TOMSK POLYTECHNIC UNIVERSITY

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)

Школа Информационных технологий и робототехники
Направление подготовки 09.04.02 Информационные системы и технологии
Отделение школы (НОЦ) Информационных технологий

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Тема работы
Модернизация кроссплатформенной библиотеки для обработки видео на базе FFmpeg

УДК 004.65:004.932

Обучающийся

Группа	ФИО	Подпись	Дата
8ИМ11	Дмитриев Василий Витальевич		

Руководитель ВКР

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР	Чердынцев Е. С.	к.т.н.		

КОНСУЛЬТАНТЫ ПО РАЗДЕЛАМ:

По разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОСГН ШБИП	Спицына Л.Ю.	к.э.н.		

По разделу «Социальная ответственность»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ООД ШБИП	Антоневич А. О.	к.б.н.		

ДОПУСТИТЬ К ЗАЩИТЕ:

Руководитель ООП	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР	Шерстнев В. С.	к.т.н.		

Томск – 2023 г.

ПЛАНИРУЕМЫЕ РЕЗУЛЬТАТЫ ОСВОЕНИЯ ООП

Код компетенции	Наименование компетенции
Универсальные компетенции	
УК(У)-1	Способен осуществлять критический анализ проблемных ситуаций на основе системного подхода, вырабатывать стратегию действий
УК(У)-2	Способен управлять проектом на всех этапах его жизненного цикла
УК(У)-3	Способен организовывать и руководить работой команды, вырабатывая командную стратегию для достижения поставленной цели
УК(У)-4	Способен применять современные коммуникативные технологии, в том числе на иностранном (-ых) языке (-ах), для академического и профессионального взаимодействия
УК(У)-5	Способен анализировать и учитывать разнообразие культур в процессе межкультурного взаимодействия
Общепрофессиональные компетенции	
ОПК(У)-1	Способен разрабатывать оригинальные алгоритмы и программные средства, в том числе с использованием современных интеллектуальных технологий, для решения профессиональных задач
ОПК(У)-2	Способен анализировать профессиональную информацию, выделять в ней главное, структурировать, оформлять и представлять в виде аналитических обзоров с обоснованными выводами и рекомендациями
ОПК(У)-3	Способен разрабатывать и модернизировать программное и аппаратное обеспечение информационных
Профессиональные компетенции	
ПК(У)-1	Способен проектировать сложные пользовательские интерфейсы; анализировать эргономические характеристики программных продуктов
ПК(У)-2	Способен осуществлять руководство разработкой комплексных проектов на всех стадиях и этапах выполнения работ
ПК(У)-3	Способен проектировать и организовывать учебный процесс по образовательным программам с использованием современных образовательных технологий

TOMSK POLYTECHNIC UNIVERSITY ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)

Школа Информационных технологий и робототехники
Направление подготовки 09.04.02 Информационные системы и технологии
Отделение школы (НОЦ) Информационных технологий

УТВЕРЖДАЮ:
Руководитель ООП

_____ _____ Шерстенев В.С.
(Подпись) (Дата) (Ф.И.О.)

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

В форме:

Магистерской диссертации

(бакалаврской работы, дипломного проекта/работы, магистерской диссертации)

Студенту:

Группа	ФИО
8ИМ11	Дмитриеву Василию Витальевичу

Тема работы:

Разработка кроссплатформенной библиотеки для обработки видео на базе FFmpeg	
Утверждена приказом директора (дата, номер)	Приказ № 40-52/с от 09.02.2023

Срок сдачи студентом выполненной работы:	15.06.2023

ТЕХНИЧЕСКОЕ ЗАДАНИЕ:

Исходные данные к работе (наименование объекта исследования или проектирования; производительность или нагрузка; режим работы (непрерывный, периодический, циклический и т. д.); вид сырья или материал изделия; требования к продукту, изделию или процессу; особые требования к особенностям функционирования (эксплуатации) объекта или изделия в плане безопасности эксплуатации, влияния на окружающую среду, энергозатратам; экономический анализ и т. д.)	Объектом исследования являются различные библиотеки кодирования и декодирования видео, а предметом исследования - реализация конкретной кроссплатформенной библиотеки кодирования и декодирования видео. В качестве исходных данных приняты низкоуровневое программное обеспечение кодирующие и декодирующие видео и исходные данные о существующих кодировщиках и

	декодеровщиках.
Перечень подлежащих исследованию, проектированию и разработке вопросов (аналитический обзор по литературным источникам с целью выяснения достижений мировой науки техники в рассматриваемой области; постановка задачи исследования, проектирования, конструирования; содержание процедуры исследования, проектирования, конструирования; обсуждение результатов выполненной работы; наименование дополнительных разделов, подлежащих разработке; заключение по работе).	Повышение удобства и эффективности работы программиста занимающегося обработкой видео на разных платформах. Описание раздела финансового менеджмента, ресурсоэффективности и ресурсосбережения. Описание раздела социальной ответственности.
Перечень графического материала (с точным указанием обязательных чертежей)	Презентация в формате *.pptx.

Консультанты по разделам выпускной квалификационной работы
(с указанием разделов)

Раздел	Консультант
Финансовый менеджмент, ресурсоэффективность и ресурсосбережение	Спицына Л. Ю., доцент ОСГН
Социальная ответственность	Антоневич О. А., доцент ООД
Английский язык	Уткина А. Н., доцент ОИЯ

Названия разделов, которые должны быть написаны на русском и иностранном языках:

Обзор технологий кодирования и декодирования видео

Дата выдачи задания для раздела по линейному графику	13.02.23

Задание выдал руководитель:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР ТПУ	Чердынцев Евгений Сергеевич	к.т.н.		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8ИМ11	Дмитриев Василий Витальевич		

TOMSK POLYTECHNIC UNIVERSITY

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)

Школа Информационных технологий и робототехники
Направление подготовки 09.04.02 Информационные системы и технологии
Уровень образования магистратура
Отделение школы (НОЦ) Информационных технологий
Период выполнения (весенний семестр 2022 /2023 учебного года)

Форма представления работы:

Магистерская диссертация

(бакалаврская работа, дипломный проект/работа, магистерская диссертация)

КАЛЕНДАРНЫЙ РЕЙТИНГ-ПЛАН
выполнения выпускной квалификационной работы

Срок сдачи студентом выполненной работы:	15.06.2023

Дата контроля	Название раздела (модуля) / вид работы (исследования)	Максимальный балл раздела (модуля)
20.02.2023	Анализ исследуемой области	*10*
06.03.2023	Проектирование программной системы	*20*
22.03.2023	Разработка программной системы	*30*
14.04.2023	Тестирование программной системы	*10*
24.04.2023	Оформление расчетно-пояснительной записки	*20*
18.05.2023	Подготовка к защите ВКР	*10*
15.06.2023	Сдача работы	*100*

СОСТАВИЛ:
Руководитель ВКР

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР	Чердынцев Е. С.	к.т.н.		

СОГЛАСОВАНО:
Руководитель ООП

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР	Шерстнев В. С.	к.т.н.		

РЕФЕРАТ

Выпускная квалификационная работа содержит 153 с., 30 рис., 19 табл., 36 источников, 5 прил.

Ключевые слова: кодирование видео, декодирование видео, библиотека, библиотека разработчика, кадрирование, масштабирование, кроссплатформенность, H264.

Объектом исследования является кроссплатформенная (Windows, Android) библиотека кодирования и декодирования видео с возможностью кадрирования и масштабирования изображения.

Цель работы – модернизация библиотеки FFmpeg путем разработки оберточной кроссплатформенной библиотеки кодирования и декодирования видео.

В процессе исследования проведен обзор существующих алгоритмов кодирования и декодирования видео, описаны существующие средства кодирования и декодирования видео. Спроектирована и разработана библиотека кодирования и декодирования видео, проведено ручное и unit-тестирование. Произведен расчет финансовой составляющей работы, определены возможные опасные и вредные факторы, влияющие на выполнение работы.

В результате исследования получена кроссплатформенная библиотека, реализующая кодирование и декодирование видео с возможностью кадрирования и масштабирования изображения. Программный интерфейс представлен на языке программирования высокого уровня.

Область применения: обработка видео.

Содержание

Введение

В настоящее время видео является одним из самых популярных и востребованных форматов контента в интернете. С развитием сетей передачи данных и увеличением доступности широкополосного интернета, видео стало доступным для потребителей по всему миру. Однако, с ростом объема потребляемого видео, возникла необходимость в более эффективных методах его передачи и хранения. Для этого были разработаны различные методы кодирования и декодирования видео.

Процесс кодирования видео заключается в уменьшении объема информации, которую необходимо передать или сохранить. Это достигается путем удаления изображений и аудио, которые несущественны для воспроизведения видео. Кодеки (сжимающие алгоритмы) используются для сжатия данных. При декодировании видео происходит обратный процесс – восстановление сжатой информации.

Для написания различных программ, в том числе и обработки видео, разработчики используют написанные другими программистами библиотеки.

Библиотека кодирования и декодирования видео – это программный инструмент, который предоставляет набор функций для сжатия и распаковки видеоданных. Библиотеки такого типа используются для кодирования и декодирования видео в различных форматах, в том числе для передачи видео через интернет, хранения видео на устройствах хранения данных и др.

Библиотеки кодирования и декодирования видео могут быть использованы различными приложениями для уменьшения размера видеофайлов и увеличения скорости передачи данных. Также они могут использоваться для повышения качества видео, уменьшения задержки и улучшения производительности при воспроизведении.

Одной из самых популярных библиотек кодирования и декодирования видео является FFmpeg. FFmpeg – это набор бесплатных программных инструментов для записи, конвертирования и потоковой передачи аудио- и видеоданных. FFmpeg предоставляет возможности для сжатия видео,

нарезки, изменения размера и поворота видео, а также преобразования форматов видео.

Однако библиотека FFmpeg, как и многие другие популярные библиотеки предоставляют пользователю не удобный интерфейс на языках программирования низкого уровня, как правило на языке C. Поэтому существует потребность в более удобных инструментах для кодирования и декодирования видео.

Поэтому в данной работе будет произведена модернизация библиотеки FFmpeg путем создания оберточной библиотеки с интерфейсом на языках высокого уровня для двух платформ: Windows и Android. В качестве алгоритма кодирования выбран H.264 (Advanced Video Coding).

1 Обзор технологий кодирования и декодирования видео

1.1 Кодирование и декодирование видео

1.1.1 Кодирование видео

Кодирование видео - это процесс сжатия видеоданных с целью уменьшения их размера для более эффективной передачи и хранения. Для этого используется кодек (кодирующий/декодирующий алгоритм), который преобразует видео в цифровой формат и сжимает его [1].

Существуют различные кодеки, каждый из которых имеет свои особенности. Например, кодек H.264, который часто используется в видеокамерах и телевизионных передачах, обеспечивает высокую степень сжатия с сохранением качества изображения. Другой популярный кодек, VP9, используется для видео в высоком разрешении на YouTube.

Для кодирования видео используются различные методы, включая пространственную и временную компрессию. Пространственная компрессия уменьшает размер файла путем удаления деталей изображения, которые не воспринимаются человеческим глазом. Временная компрессия основана на использовании различных методов сжатия данных между кадрами, таких как разница между кадрами и прогнозирование движения.

1.1.2 Декодирование видео

Декодирование видео - это процесс восстановления видеоданных из сжатого формата в оригинальный видеофайл. Для этого используется тот же кодек, который использовался для сжатия данных [1].

При декодировании видео сначала происходит декомпрессия данных, которая восстанавливает сжатый видеопоток в оригинальный видеофайл. Затем происходит декодирование данных, где видеофайл преобразуется из

цифрового формата в видео с помощью алгоритмов декодирования, которые обратны кодированию.

Для декодирования видео необходим компьютер или другое устройство с поддержкой соответствующего кодека. Для просмотра видео на устройстве также может быть необходимо установить дополнительное программное обеспечение, такое как плеер или браузерный плагин.

1.1.3 Общие положения кодирования и декодирования

В целом, кодирование и декодирование видео являются важными этапами в обработке и передаче видеоданных. Они позволяют уменьшить размер файлов, что в свою очередь облегчает их передачу через сети или хранение на устройствах. Также сжатие видео может улучшить производительность воспроизведения на устройствах с ограниченными ресурсами.

Однако при кодировании и декодировании видео возможна потеря качества изображения из-за компрессии данных. Поэтому важно выбирать подходящий кодек и настраивать его параметры таким образом, чтобы достичь наилучшего баланса между качеством и размером файла.

Кроме того, важно помнить о том, что различные устройства и программное обеспечение могут поддерживать разные форматы и кодеки, что может привести к проблемам с совместимостью. Поэтому при передаче или воспроизведении видео необходимо убедиться, что используется подходящий формат и кодек для конкретного устройства или программного обеспечения [2].

1.2 Видеокодеки

Инструменты, используемые для сжатия и воспроизведения видеофайлов, называются кодеками (или видеокодеками). Определение

кодека: программа / алгоритм сжатия (то есть уменьшения размера) видеоданных (видеофайла, видеопотока) и восстановления сжатых данных. Кодек означает кодер и декодер (co/dec) [3]. Обычно аппаратное устройство или компьютерное программное обеспечение, кодек — это видеокодер, который кодирует или декодирует поток цифровых данных или сигнал. Они сжимают необработанные видео- и аудиофайлы между аналоговым и цифровым форматами и уменьшают их размер. Существует множество различных видеокодеков, основные из них:

1. H.264/AVC (Advanced Video Coding) – это один из самых распространенных видео кодеков. Он имеет хорошую производительность, хорошее соотношение сжатия и качества видео, а также поддерживается большинством устройств и платформ [4]. Он используется для сжатия видео высокого разрешения, такого как Full HD и 4K, и может поддерживать различные битовые скорости. Широко используется на различных платформах, включая YouTube, Vimeo, Netflix, Facebook, Amazon Prime Video и многих других;
2. H.265/HEVC (High Efficiency Video Coding) – это новый стандарт сжатия видео, который заменяет H.264. Он предлагает более эффективное сжатие, что означает, что он может сжимать видео с более высоким разрешением и битовыми скоростями с меньшим размером файла [5]. Однако он требует более высокой вычислительной мощности для кодирования и декодирования, что может ограничить его использование на старых устройствах [6]. Используется на некоторых платформах, таких как Apple TV и Netflix;
3. VP9 – это открытый стандарт сжатия видео, разработанный Google. Он предлагает более эффективное сжатие, чем H.264, но менее эффективное, чем H.265. Он также требует меньше вычислительной мощности, чем H.265, но больше, чем H.264. VP9 используется в видео на YouTube и других платформах Google;

4. AV1 – это новый открытый стандарт сжатия видео, разработанный в рамках Alliance for Open Media. Он используется для сжатия видео высокого разрешения и битовых скоростей и предлагает более эффективное сжатие, чем H.264 и VP9. Однако он требует еще большей вычислительной мощности для кодирования и декодирования, чем H.265.

На рисунке 1.1 представлена гистограмма распространенности кодеков по данным отчета Bitmovin Video Developer за 2019 г [7].

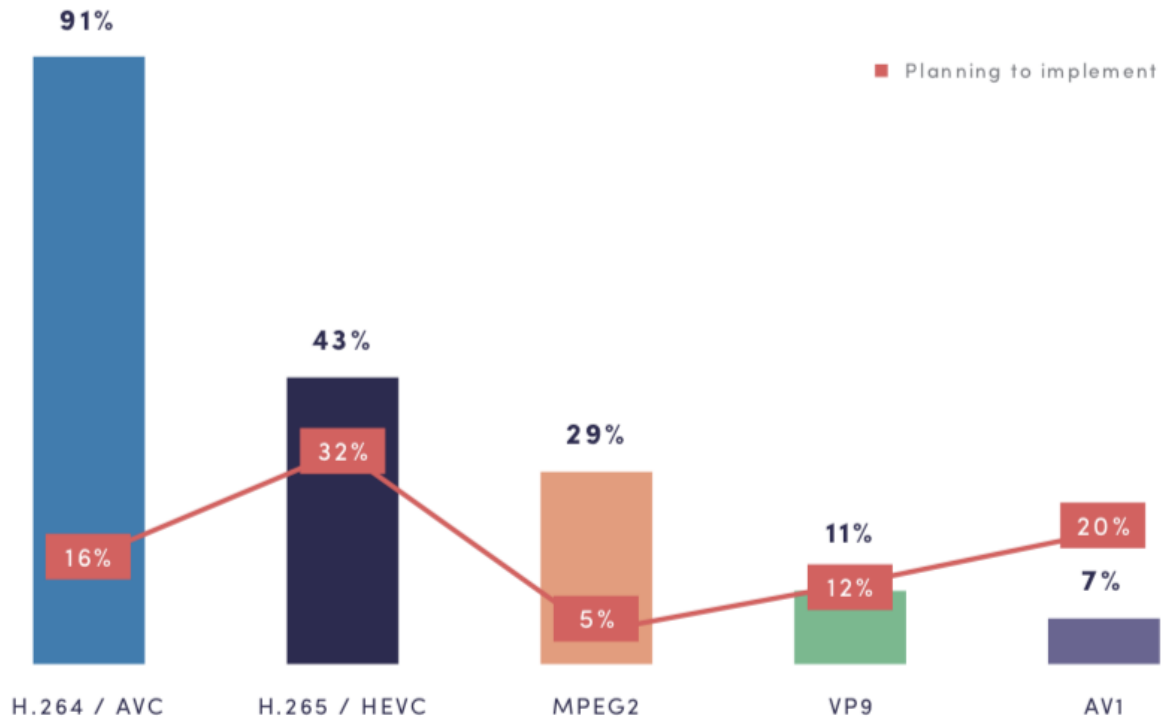


Рисунок 1.1 – Использование видеокодека (красная линия – изменение использования к 2020)

Как видно из рисунка N наиболее распространенным видеокодеком является H.264/AVC (Advanced Video Coding). Почти каждое существующее устройство поддерживает этот протокол, и он часто используется для онлайн-видео. Однако доступно несколько других кодеков, включая MPEG-2, HEVC, VP9 и AV1 [8].

В данной работе используется кодек H.264.

1.2.1 Кодек H.264

H.264, также известный как MPEG-4 Part 10 или AVC (Advanced Video Coding), является стандартом сжатия видео, разработанным Международной организацией по стандартизации (ISO) и Международным электротехническим комитетом (IEC). Он является одним из наиболее распространенных форматов сжатия видео и широко используется в трансляции потокового видео, веб-видео, цифровом видео и т.д [8].

H.264 использует алгоритм сжатия видео с потерями, который удаляет изображения, не заметные для глаза человека, для снижения объема данных. В отличие от старых стандартов сжатия видео, таких как MPEG-2, H.264 использует современные методы сжатия, такие как предсказание движения, пространственная дискретизация косинусного преобразования (DCT) и адаптивное квантование.

Кодек H.264 состоит из трех основных компонентов: энкодера, декодера и битового потока. Энкодер H.264 преобразует исходное видео в сжатый формат, который может быть передан через сеть или записан на жесткий диск. Декодер H.264 обратно преобразует сжатый поток в оригинальное видео. Битовый поток содержит все необходимые данные для воспроизведения видео, такие как кадры, информацию о цвете, частоте кадров, битовую скорость и т.д [9].

В H.264 используется ряд методов сжатия данных, включая:

- предсказание движения: метод, который анализирует движение пикселей между кадрами и использует эту информацию для определения, какие пиксели могут быть сокращены или удалены;
- адаптивное квантование: метод, который изменяет уровень квантования, используемый для сжатия каждого блока в зависимости от сложности содержимого. Это позволяет уменьшить размер файла без ухудшения качества изображения;

- пространственная дискретизация косинусного преобразования (DCT): метод, который разбивает блоки пикселей на частотные составляющие и преобразует их в сигналы частоты. Этот метод позволяет эффективно сжимать видео, удаляя информацию о высокочастотных составляющих, которые обычно не видны глазу человека.

Другой важной особенностью кодека H.264 является использование многоканальной компрессии. Это означает, что видео может быть разбито на несколько потоков, каждый из которых может быть сжат отдельно. Это улучшает производительность сжатия для видео с большим количеством движения или быстрыми изменениями кадров [10].

H.264 также поддерживает несколько профилей и уровней, которые определяют ограничения и возможности сжатия видео. Например, профиль Main позволяет использовать предсказание движения и адаптивное квантование, а профиль High поддерживает кодирование видео с высоким разрешением (1080p) и высокой битовой скоростью.

Однако, как и любой другой кодек, H.264 имеет свои ограничения. Например, он может быть требовательным к процессору и требует большой битовой скорости для сжатия видео высокого разрешения. Кроме того, H.264 не является лучшим выбором для видео с низкой движущейся картинкой или сценами со сложными эффектами, так как это может привести к существенной потере качества изображения.

Тем не менее, благодаря своей эффективности и широкому распространению, H.264 остается одним из наиболее популярных форматов сжатия видео и используется во многих приложениях, включая видеоконференции, потоковое видео, цифровое телевидение и многие другие.

1.2.1.1 Алгоритм работы кодека H.264

Краткое описание алгоритма работы кодека H.264 по этапам [11]:

1. Разбиение кадра на макроблоки: кадр разбивается на макроблоки размером от 4x4 до 16x16 пикселей;
2. Применение преобразования: каждый макроблок преобразуется в блоки коэффициентов преобразования, используя алгоритм преобразования дискретного косинусного преобразования (DCT);
3. Квантование: блоки коэффициентов преобразования квантуется с помощью квантователя, чтобы уменьшить количество информации;
4. Дифференциальное кодирование: используется метод кодирования предсказанных значений блоков, основанный на предыдущих кадрах и блоках внутри текущего кадра;
5. Удаление избыточности: данные сжимаются дополнительно путем удаления избыточности, такой как нули и дубликаты;
6. Кодирование длин серий: производится кодирование длин серий для того, чтобы уменьшить количество информации, которое необходимо передавать;
7. Упаковка в поток: каждый кадр кодируется отдельно, а затем упаковывается в поток для передачи через Интернет или другие цифровые среды.

Этот процесс повторяется для каждого кадра в видеопотоке. При декодировании, процесс обратный: поток данных разбивается на отдельные кадры, декодируется и восстанавливается в исходное изображение [12].

1.3 Аппаратное ускорение кодирования и декодирования видео

Аппаратное ускорение процесса кодирования и декодирования видео – это использование специализированных вычислительных устройств (например, видеокарты, процессоры с интегрированными графическими ядрами или специализированные чипы) для обработки видеоданных в реальном времени, с целью ускорения процесса кодирования и декодирования видео [13].

Вместо использования только центрального процессора (CPU) компьютера для выполнения этих операций, аппаратное ускорение позволяет распределить нагрузку между CPU и специализированными устройствами, которые могут обрабатывать большие объемы данных в значительно более быстром темпе [14].

Это значительно снижает время, необходимое для обработки видео, что особенно важно для задач, связанных с обработкой высококачественного потока кадров, таких как профессиональная видеопроизводство, стриминг, а также для повседневных задач, таких как просмотр видео на мобильных устройствах или компьютерах.

В целом, аппаратное ускорение процесса кодирования и декодирования видео может значительно повысить производительность компьютерной системы и обеспечить более быструю и эффективную обработку видеоданных.

Для кодирования видео используются вышеописанные форматы, такие как H.264, H.265, VP9, AV1, MPEG-4, и другие. Каждый формат имеет свои особенности, требования к ресурсам и оптимизации для конкретных устройств. Некоторые видеокарты имеют специализированные блоки кодирования и декодирования для определенных форматов, что может значительно ускорить процесс [15].

Кроме того, аппаратное ускорение может использоваться для решения проблем с производительностью, связанных с использованием видео на мобильных устройствах, где ограниченные ресурсы и батареи могут привести к низкой производительности. В таких случаях могут быть использованы специальные аппаратные компоненты, которые потребляют меньше энергии и могут обеспечить более быструю обработку видеоданных .

Аппаратное ускорение может быть использовано для различных задач, связанных с видео, таких как запись, обработка, трансляция, конвертация и проигрывание. Например, при стриминге видео в реальном времени может использоваться аппаратное ускорение для сжатия видео и оптимизации

потока данных, чтобы снизить задержки и обеспечить более плавный просмотр видео [16].

В целом, аппаратное ускорение процесса кодирования и декодирования видео является важным инструментом для обеспечения высокой производительности и качества обработки видеоданных, что делает его особенно важным для профессиональных пользователей, таких как киноиндустрия, телевидение и веб-разработчики, а также для повседневных пользователей.

На рисунке 1.2 представлено сравнение скорости декодирования изображения на ЦП и двух различных видеокартах.

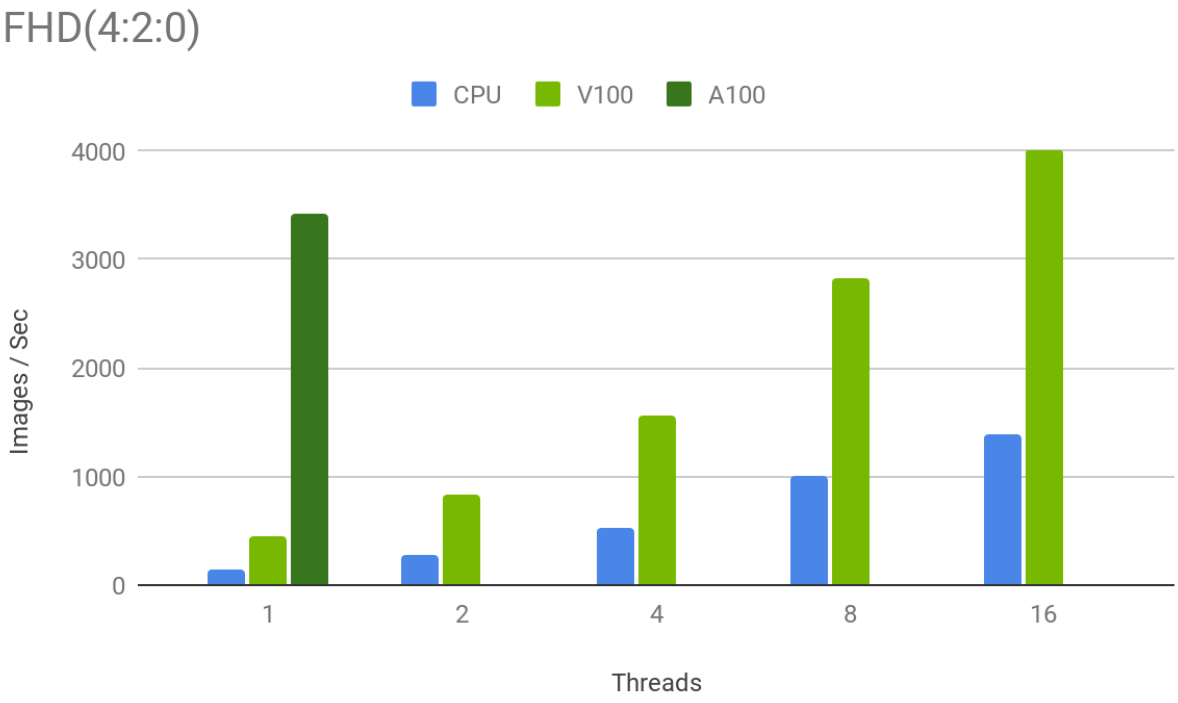


Рисунок 1.2 – Сравнение скорости декодирования изображения на ЦП и двух различных видеокартах

Так, для повышения производительности кодирования и декодирования видео необходимо использовать ГП.

В библиотеке FFmpeg используемой в процессе разработки библиотеки для Windows присутствуют аппаратные кодеры Intel Quick Sync Video, Nvidia

NVENC и другие. Для декодирования используется DirectX, методы для работы с котором также есть в FFmpeg.

Для декодирования видео на Android в FFmpeg присутствует MediaCodec.

1.3.1 Intel Quick Sync Video

Intel Quick Sync Video - это технология аппаратного ускорения видео, которая была впервые представлена компанией Intel в 2010 году. Она была разработана для ускорения операций кодирования, декодирования и обработки видео на компьютерах и других устройствах [17, с. 2].

Основная идея Intel Quick Sync Video заключается в том, чтобы использовать встроенную графическую подсистему процессора Intel для обработки видео. Для этого используется специальный блок аппаратного ускорения, который интегрирован в графический процессор процессора Intel.

Этот блок аппаратного ускорения способен обрабатывать видео с высокой скоростью, поскольку он работает параллельно с процессором и использует многопоточную обработку для ускорения операций. Это позволяет сократить время, необходимое для выполнения задач обработки видео, в несколько раз.

Intel Quick Sync Video поддерживает множество форматов видео, включая MPEG-2, H.264 и HEVC. Он может использоваться для кодирования и декодирования видео, а также для обработки видео, такой как изменение размера, наложение текста и других эффектов.

Преимущества Intel Quick Sync Video включают высокую производительность, низкое потребление энергии и возможность обработки видео без задержек. Он может использоваться в различных приложениях, таких как видео конвертеры, редакторы и потоковые сервисы [17, с. 8].

Механизм работы Intel Quick Sync Video основан на использовании специализированных аппаратных блоков внутри процессора, которые

позволяют выполнять операции кодирования и декодирования видео в режиме реального времени с использованием минимального количества вычислительных ресурсов.

Основные специализированные блоки:

1. Video Codec Engine (VCE) - это блок, который отвечает за аппаратное сжатие видео. Он поддерживает различные видеоформаты, включая H.264 и H.265, и может выполнять операции кодирования с большой скоростью;
2. Video Processing Engine (VPE) - это блок, который отвечает за обработку видео. Он может выполнять операции масштабирования, поворота, наложения эффектов и т.д.;
3. Video Quality Engine (VQE) - это блок, который отвечает за улучшение качества видео. Он может выполнять операции шумоподавления, сглаживания, улучшения контрастности и т.д.;
4. Video Analysis and Scene Detection (VASD) - это блок, который отвечает за анализ видео и определение его характеристик, таких как разрешение, кадровая частота и т.д. Он также может определять сцены и переходы между ними, что может быть полезно при редактировании видео.

Самым важным модулем является Multi-Format Codec Engine (MFX), реализующий декодирование и кодирование видео. Схема MFX представлена на рисунке 1.3.

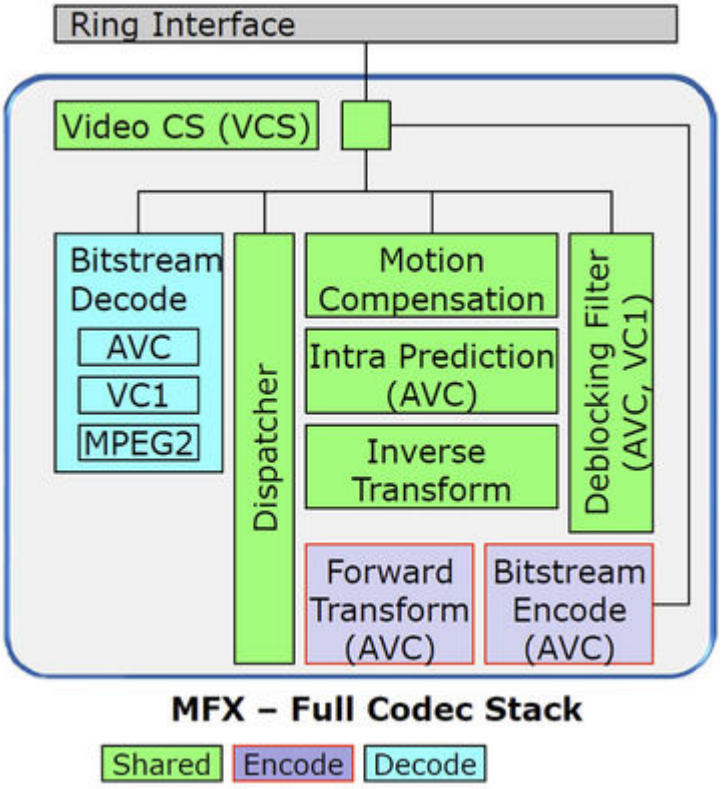


Рисунок 1.3 – Схема MFX

В целом, Intel Quick Sync Video является мощным и эффективным инструментом для обработки видео, который позволяет быстро и легко выполнять различные задачи, связанные с обработкой видео на компьютерах и других устройствах.

1.3.2 Nvidia NVENC

Nvidia NVENC (Nvidia Encoder) - это аппаратный кодировщик видео, разработанный компанией Nvidia. Он предназначен для ускорения процесса кодирования видео на компьютерах с графическими процессорами (GPU) от Nvidia [18, с. 3].

NVENC использует параллельную обработку и оптимизированные алгоритмы кодирования, чтобы ускорить процесс сжатия видео, особенно при использовании высококачественных настроек кодирования. Это

значительно уменьшает нагрузку на центральный процессор (CPU), освобождая его для выполнения других задач.

NVENC поддерживает множество форматов видео, включая H.264, H.265 (HEVC), VP8, VP9 и MPEG-2, и способен обеспечивать высокую скорость кодирования и хорошее качество видео.

Одним из основных преимуществ использования NVENC является снижение нагрузки на центральный процессор, что может улучшить производительность системы и уменьшить задержки при стриминге видео в режиме реального времени.

NVENC также поддерживает функции, такие как переменный битрейт, константный битрейт, контроль качества и поддержку нескольких потоков, что делает его удобным инструментом для различных задач видеокодирования, включая стриминг, запись игрового процесса и профессиональную видеопродукцию.

NVENC использует аппаратные возможности графического процессора Nvidia для быстрого сжатия видео, что позволяет достичь более высокой производительности и лучшего качества видео по сравнению с программным кодированием на центральном процессоре.

NVENC также поддерживает аппаратную обработку цвета и пространство цвета, что обеспечивает более точное отображение цветов в видео.

NVENC доступен во многих продуктах и платформах, включая видеокарты Nvidia серии GeForce, Quadro и Tesla, а также на мобильных устройствах с процессорами Tegra и в виртуальных машинах в облачных сервисах Nvidia [8, с. 7].

Кроме того, NVENC используется в различных программных приложениях для кодирования видео, таких как OBS Studio, Adobe Premiere Pro, CyberLink PowerDirector, XSplit и многих других.

В целом, Nvidia NVENC является мощным инструментом для быстрого и эффективного сжатия видео, который помогает улучшить производительность системы и достичь высокого качества видео.

1.3.3 MediaCodec

MediaCodec - это часть Android SDK, которая позволяет разработчикам создавать приложения для кодирования и декодирования медиаданных, таких как видео и аудио. MediaCodec является частью API для обработки медиаданных в Android.

MediaCodec поддерживает широкий спектр форматов медиаданных, включая MPEG-4, H.264, H.265, VP8, VP9, AAC, MP3 и другие. Он также предоставляет возможность обрабатывать потоковые данные в реальном времени, что делает его очень полезным для разработки приложений для видеовещания, видеоконференций и других приложений, которые требуют обработки медиаданных в реальном времени.

MediaCodec позволяет обрабатывать медиаданные в различных режимах работы, включая аппаратное ускорение, что позволяет приложениям использовать аппаратные возможности устройства для кодирования и декодирования медиаданных. Это позволяет приложениям работать более эффективно и потреблять меньше ресурсов устройства.

MediaCodec также предоставляет возможность настройки параметров кодирования и декодирования, таких как битрейт, разрешение, частота кадров и другие. Это позволяет разработчикам настраивать производительность и качество видео и аудио, чтобы соответствовать требованиям конкретного приложения.

MediaCodec является мощным инструментом для разработки приложений, которые обрабатывают медиаданные. Он предоставляет разработчикам широкий спектр возможностей для обработки видео и аудио,

что делает его незаменимым инструментом для создания высококачественных приложений для Android.

Для работы с MediaCodec необходимо создать Android Surface, который будет принимать данные от кодера.

На рисунке 1.4 представлена схема работы MediaCodec.

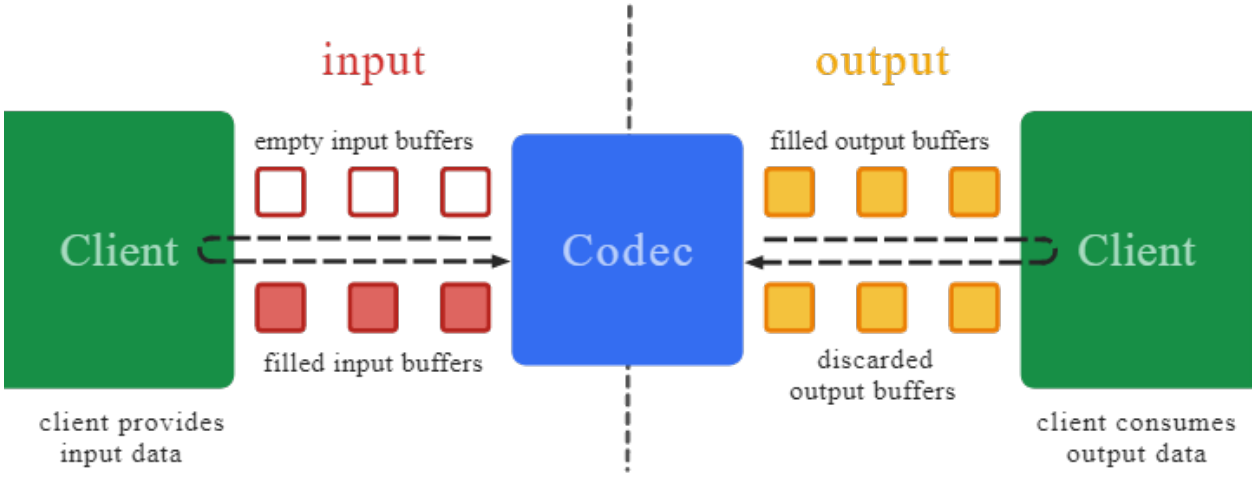


Рисунок 1.4 – Схема работы MediaCodec

Как видно из рисунке N кодек обрабатывает входные данные для создания выходных данных. Он обрабатывает данные асинхронно и использует набор входных и выходных буферов. На упрощенном уровне вы запрашиваете (или получаете) пустой входной буфер, заполняете его данными и отправляете кодеку для обработки. Кодек использует данные и преобразует их в один из своих пустых выходных буферов. Наконец, вы запрашиваете (или получаете) заполненный выходной буфер, потребляете его содержимое и возвращаете кодеку.

1.4 Выводы по разделу

Кодирование и декодирование видео – это комплексная задача требующая тщательной проработки. Для обработки видео существует множество различных видеокодеков, самым лучшим из которых является кодек H.264.

Для повышения производительности, экономии ресурсов во время кодирования и декодирования используется аппаратное ускорение, т.е. передача работы обработки видео с центрального процессора на графический процессор.

Задача кодирования и декодирования актуально, однако на данный момент существующие библиотеки либо являются низкоуровневыми, что затрудняет разработчикам работу с ними, либо являются платными и поэтому не подходят для многих программистов или компаний.

Таким образом, в рамках дипломного проектирования необходимо разработать кроссплатформенную библиотеку для Windows и Android кодирования и декодирования видео с поддержкой аппаратного ускорения, кадрирования и масштабирования видео.

2 Проектирование кроссплатформенной библиотеки кодирования и декодирования видео

2.1 Общее описание разработки библиотеки

Разрабатываемая библиотека кодирования и декодирования видео опирается на вышеупомянутую библиотеку обработки видео FFmpeg с использованием кодека H.264. При этом внутренний процесс кодирования и декодирования затронут не будут.

Планируется, что разрабатываемая библиотека будет оберткой над FFmpeg, более удобной в использовании для разработчиков. Также интерфейс библиотеки предполагает собой использование более безопасного и удобного языка C++, а не C, на котором написана FFmpeg. При это благодаря идиоме C++ Zero-Cost Abstractions потери в производительности от использования обертки ожидаются минимальными.

Предполагается, что клиент будет взаимодействовать с библиотекой через API, в котором возможно создать кодер или декодер, а также узнать какие из аппаратных кодеров и декодеров присутствуют на устройстве.

К кодеру и декодеру пользователь может добавить фильтр, который будет кадрировать или масштабировать видео.

Написанное на языке C++ ядро библиотеки подойдет для ее использования на ОС Windows, но для работы с Android необходимо будет написать обертку над основной библиотекой на языке Java. Эта обертка должна предоставлять весь тот же функционал, что и оригинальная библиотека.

2.2 Windows библиотека

Исходя из поставленной цели необходимо создать несколько сущностей: SWEncoder; SWDecoder; WinHWEncoder; WinHWDecoder. Для

того чтобы выводить видео на окно ОС Windows необходимы сущности WindowOutputManagerSW и WindowOutputManagerHW.

Для того чтобы пользователь мог задавать настройки для кодера и декодера необходимы структуры EncoderParams и DecoderParams.

Для работы с фильтрами необходима структура FilterParams, которая в себе содержит структуры CropParams и ScaleParams.

2.2.1 SWEncoder

Класс SWEncoder необходим для кодирования видео без аппаратного ускорения. Класс SWEncoder представлен на рисунке 2.1.

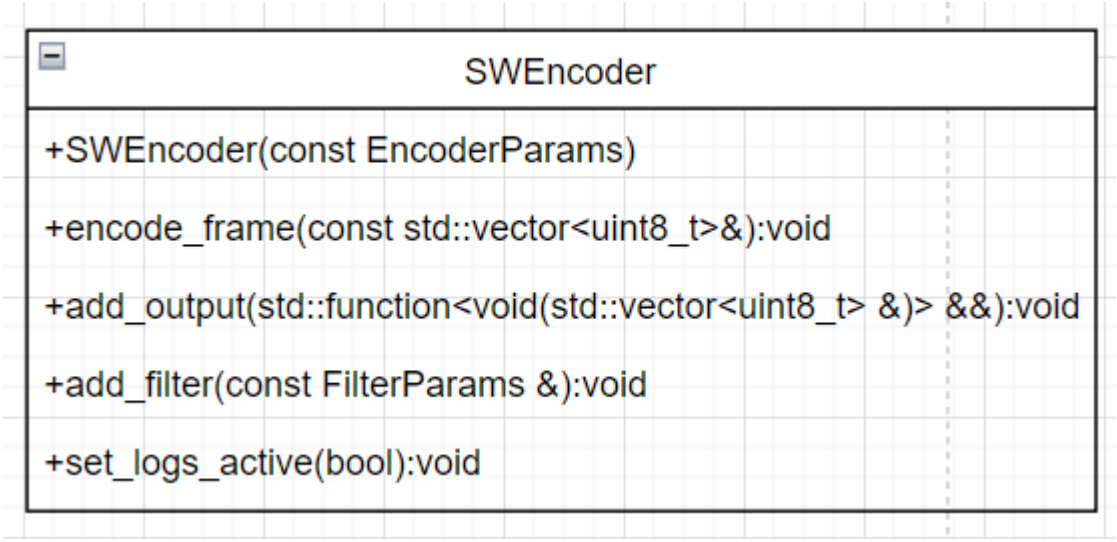


Рисунок 2.1 – Класс SWEncoder

Метод encode_frame на вход принимает массив байт, содержащий RGBA данные кадра видео. На выход метод ничего не отдает.

Метод add_output на вход принимает callback, в котором пользователь обрабатывает кодированные данные кадра. На выход метод ничего не отдает.

Метод add_filter на вход принимает параметры фильтра для кодера. На выход метод ничего не отдает.

Метод set_logs_active на вход принимает булевскую переменную, определяющую включение или выключение логов для кодера. На выход метод ничего не отдает.

Алгоритм работы SWEncoder представлен на рисунке 2.2.

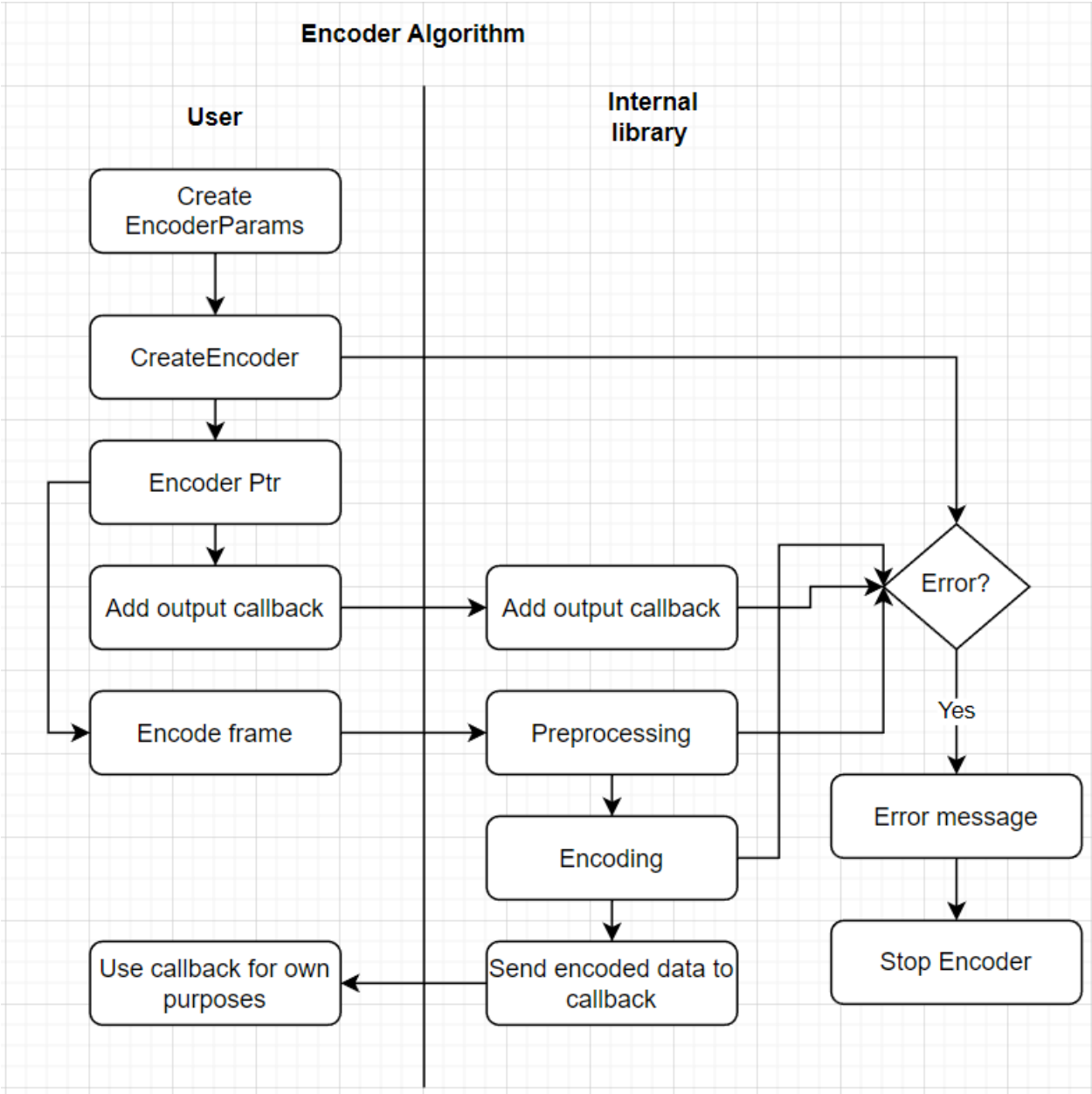


Рисунок 2.2 - Алгоритм работы SWEncoder

2.2.2 SWDecoder

Класс SWDecoder необходим для декодирования видео без аппаратного ускорения. Класс SWDecoder представлен на рисунке 2.3.

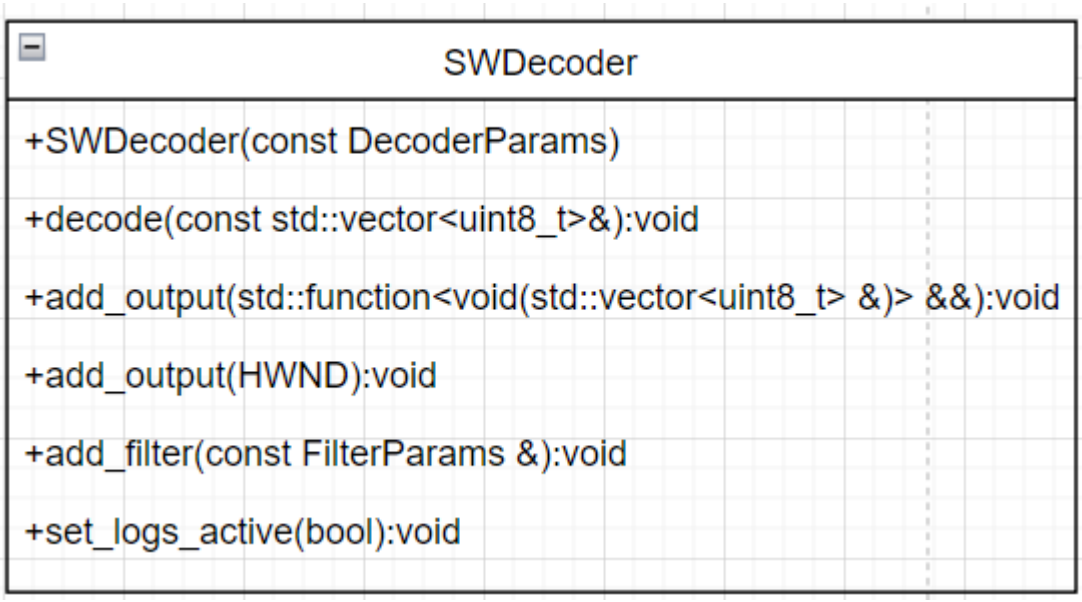


Рисунок 2.3 – Класс SWDecoder

Метод decode на вход принимает массив байт, содержащий закодированные данные видео кадра. На выход метод ничего не отдает.

Метод add_output имеет два возможных использования. В первом метод на вход принимает callback, в котором пользователь обрабатывает декодированный видео кадра. Во втором метод принимает указатель на окно ОС Windows, в котором необходимо отрисовать кадр, в тот момент, когда тот будет декодирован (данная реализация метода возможна только на ОС Windows). На выход метод ничего не отдает

Метод add_filter на вход принимает параметры фильтра для декодера. На выход метод ничего не отдает.

Метод set_logs_active на вход принимает булевскую переменную, определяющую включение или выключение логов для декодера. На выход метод ничего не отдает.

Алгоритм работы SWDecoder представлен на рисунке 2.4.

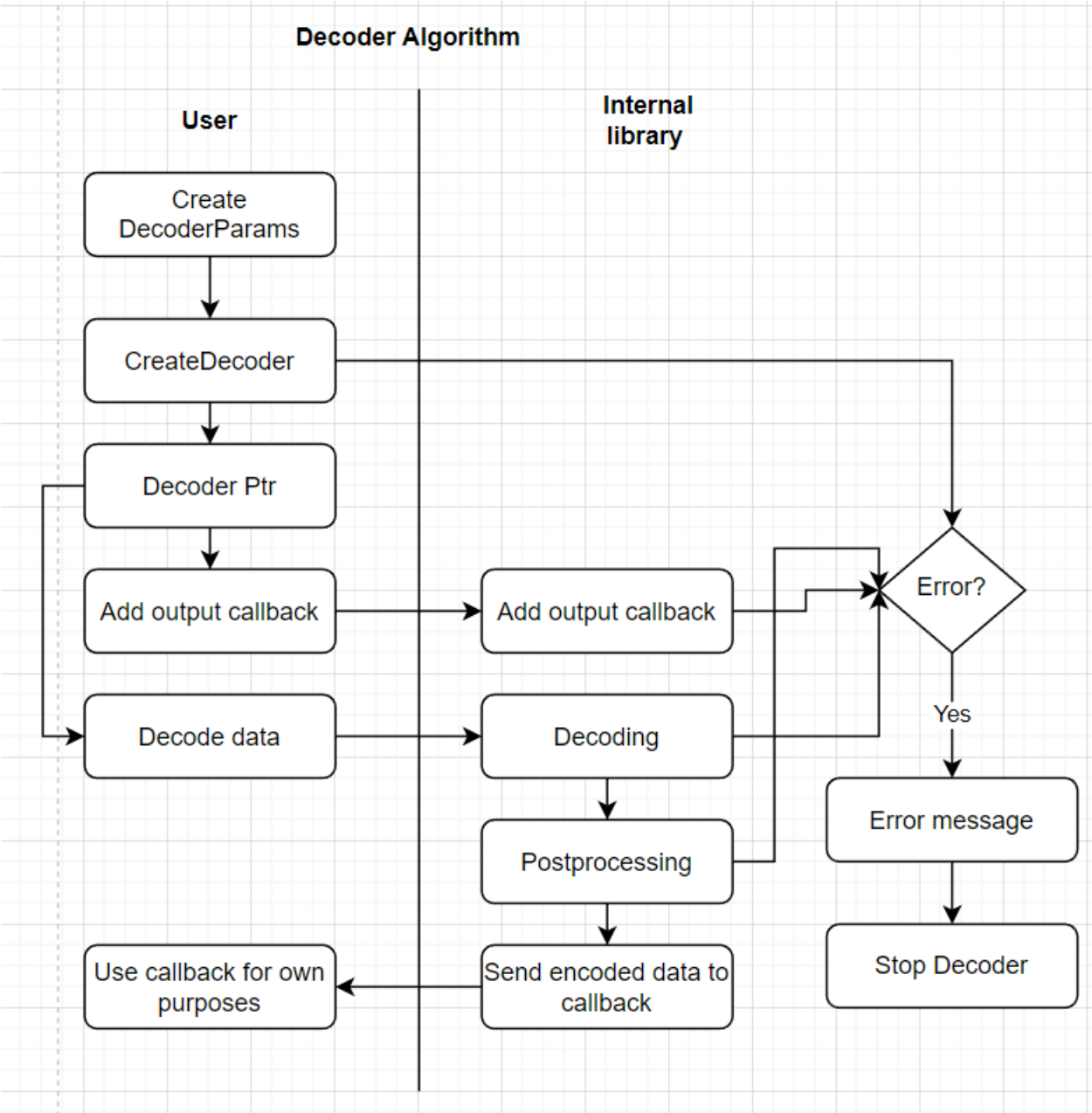


Рисунок 2.4 – Алгоритм работы SWDecoder

2.2.3 WinHWEncoder и WinHWDecoder

Сущности WinHWEncoder и WinHWDecoder с точки зрения предоставляемого интерфейса идентичны сущностям SWEncoder и SWDecoder соответственно. Данные сущности применяются, когда пользователь использует поддержку аппаратного ускорения для кодирования и декодирования видео.

Алгоритм работы аппаратного кодировщика и декодировщика не отличается от программных, однако в случае аппаратного декодера невозможно добавить callback возвращающий данные байтами, только вывести на окно ОС Windows.

2.2.4 WindowOutputManagerSW и WindowOutputManagerHW

Сущности WindowOutputManagerSW и WindowOutputManagerHW являются внутренними и пользователь библиотеки с ними не взаимодействует напрямую. Они необходимы для вывода на окно ОС Windows декодированного кадра. Если кадр был декодирован программно, используется сущность WindowOutputManagerSW, если кадр был декодирован аппаратно, используется сущность WindowOutputManagerHW.

2.2.5 EncoderParams и DecoderParams

Структуры EncoderParams и DecoderParams необходимы для инициализации кодера и декодера соотвественно.

На рисунке 2.5 представлена структура EncoderParams.

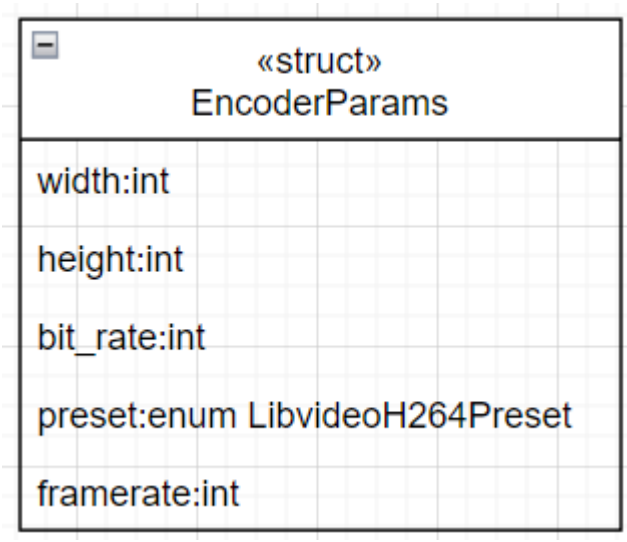


Рисунок 2.5 – Структура EncoderParams

В структуре EncoderParams содержатся следующие поля: width - ширина кадра; height - высота кадра; bit_rate - битрейт видео; preset - предустановка H.264 кодека, определяющие качество кодирования; framerate - количество кадров в секунду у видео.

На рисунке 2.6 представлена структура DecoderParams.

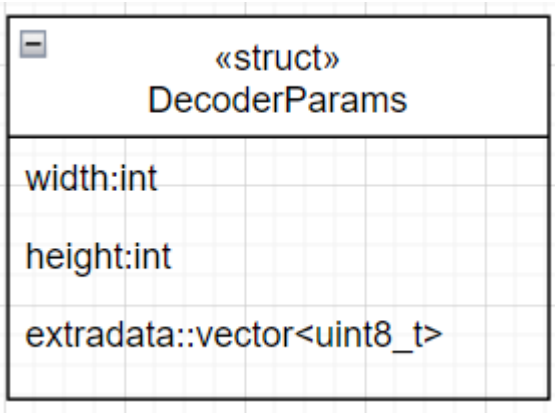


Рисунок 2.6 – Структура DecoderParams

В структуре DecoderParams содержатся следующие поля: width - ширина кадра; height - высота кадра; extradata - данные, необходимые для инициализации декодера.

2.2.6 FilterParams

Структура FilterParams содержит в себе структуры CropParams и ScaleParams и необходима для изменения параметров изображения.

На рисунке 2.7 представлена структура CropParams.

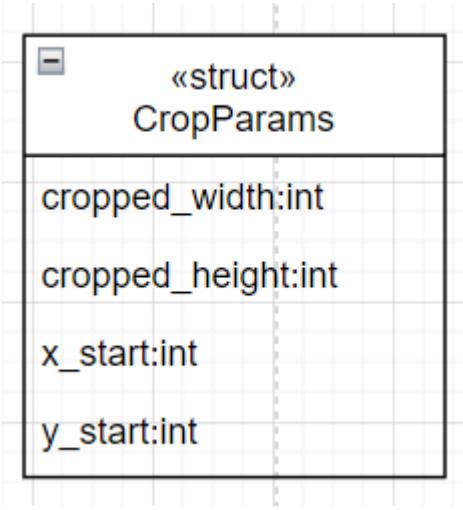


Рисунок 2.7 – Структура CropParams

В структуре CropParams содержатся следующие поля: cropped_width - ширина кадрированного изображения; cropped_height - высота кадрированного изображения; x_start - координата начала кадрирования по оси X; y_start - координата начала кадрирования по оси Y.

На рисунке 2.8 представлена структура ScaleParams.

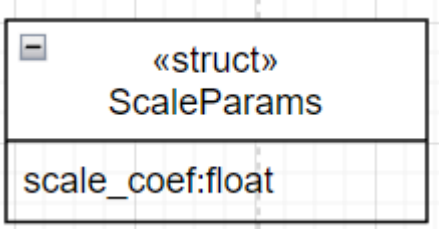


Рисунок 2.8 – Структура ScaleParams

В структуре ScaleParams содержится следующее поле: scale_coef - коэффициент масштабирования.

Алгоритм работы фильтров следующий: пользователь добавляет нужные фильтры через add_filter метод для кодера или декодера; в методе preprocessing или postprocessing фильтры берут RGBA байтовые данные и применяют к ним внутренние методы библиотеки FFmpeg и возвращают обработанные RGBA байтовые данные, которые затем кодируются или декодируются.

2.2.6 IFilter, ScaleFilter и CropFilter

Интерфейс IFilter представляет собой контракт работы с фильтрами, это необходимо, чтобы сущности Encoder и Decoder могли единообразным способом хранить в себе добавленные пользователем фильтры.

На рисунке 2.9 представлена иерархия классов фильтров.

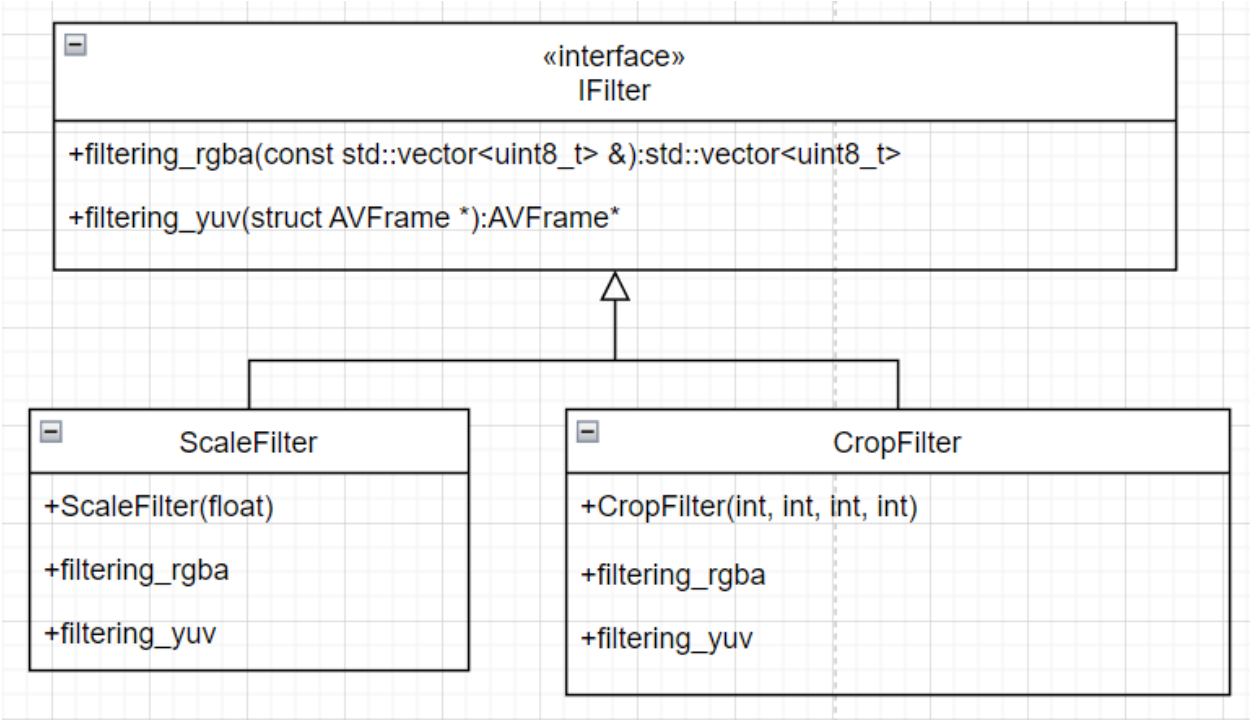


Рисунок 2.9 – Иерархия классов фильтров

Метод filtering_rgba на вход принимает RGBA данные видеокадра, на выход метод отдает отфильтрованные RGBA данные видеокадра.

Метод filtering_yuv на вход принимает YUV данные видеокадра, на выход метод отдает отфильтрованные YUV данные видеокадра.

Какой именно фильтр будет использоваться задает пользователь в FilterParams.

2.3 Android библиотека

Библиотека для кодирования и декодирования под ОС Android с точки зрения проектирования подобна библиотеки под ОС Windows. Поэтому в данной главе будут описаны различия. Под ОС Android не планируется создавать аппаратный кодер.

Созданные сущности используют в себе сущности из ядра библиотеки, написанные на языке C++ и по своей сути являются оберточными.

В Android библиотеки не планируется использование фильтров кадрирования и масштабирования.

2.3.1 SWEncoderJ

SWEncoderJ представляет собой сущность программного кодера, он представлен на рисунке 2.10.

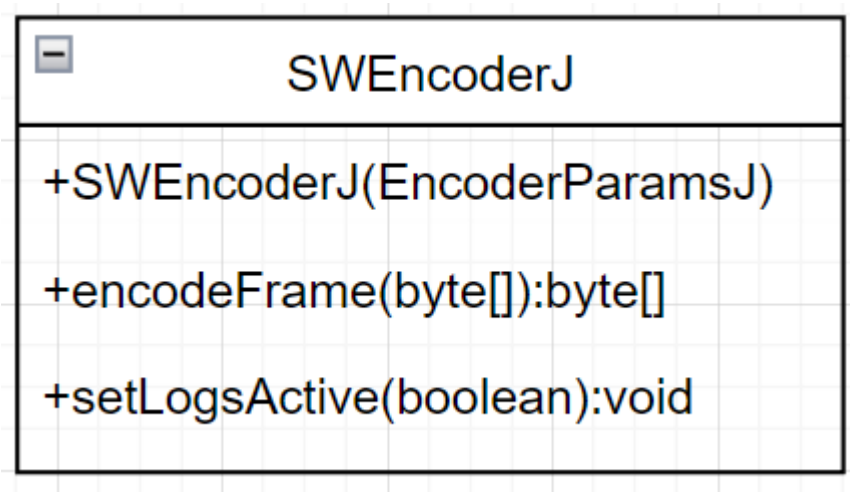


Рисунок 2.10 – Класс SWEncoderJ

Метод encodeFrame на вход принимает RGBA данные кадра, на выход метод отдает закодированные данные кадра.

Метод setLogsActive идентичен методу set_logs_active в классе SWEncoder.

2.3.2 SWDecoderJ

SWDecoderJ представляет собой сущность программного декодера, он представлен на рисунке 2.11.

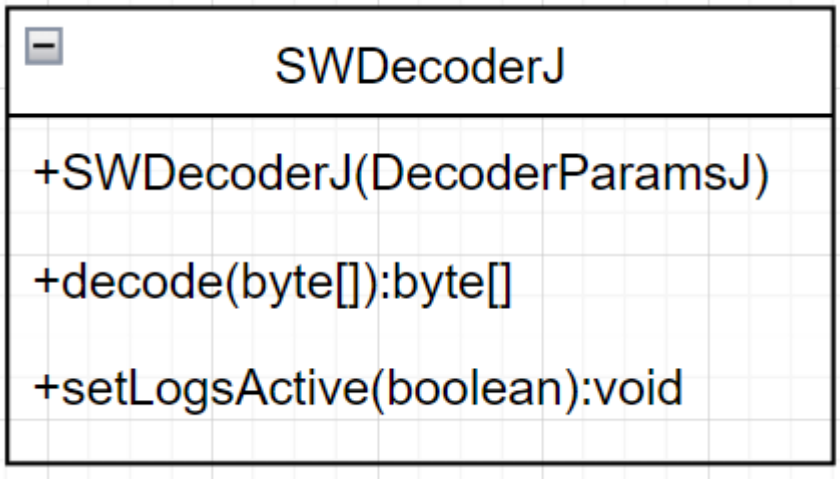


Рисунок 2.11 – Класс SWDecoder

Метод decode на вход принимает кодированные данные кадра, на выход метод отдает RGBA данные кадра.

Метод setLogsActive идентичен методу set_logs_active в классе SWDecoder.

2.3.3 HWDecoderJ

HWDecoderJ представляет собой сущность аппаратного декодера, он представлен на рисунке 2.11.

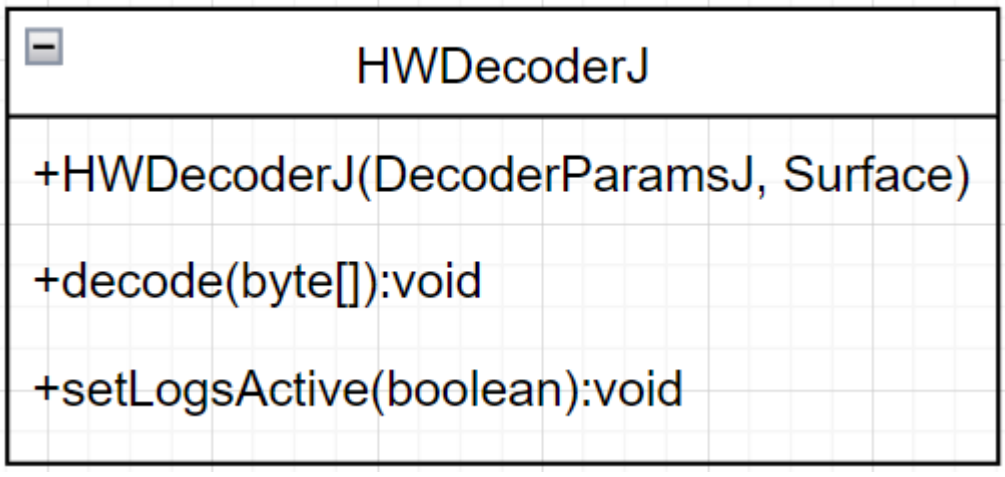


Рисунок 2.11 – Класс HWDecoderJ

В конструкторе класса HWDecoderJ принимает Android Surface, на который будет производиться отрисовка.

Метод decode на вход принимает кодированные данные кадра, на выход метод ничего не отдает. При каждом вызове метода на Surface происходит отрисовка кадра.

Метод setLogsActive идентичен методу set_logs_active в классе SWDecoder.

Сущность HWDecoderJ является оберткой для работы на ОС Android, основная обработка будет производиться в AndroidHWDecoder, он представлен на рисунке 2.12.

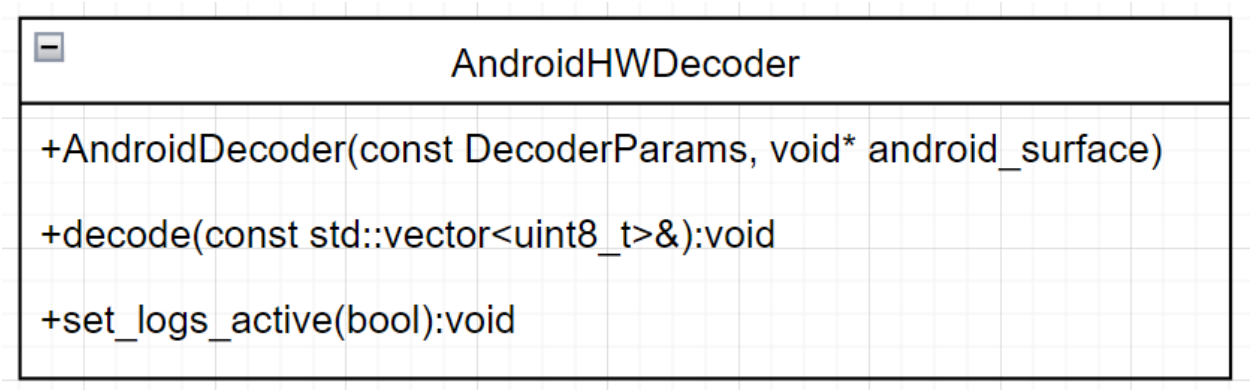


Рисунок 2.12 – Класс AndroidHWDecoder

2.4 Диаграмма классов

На рисунке 2.13 представлена диаграмма классов библиотеки.

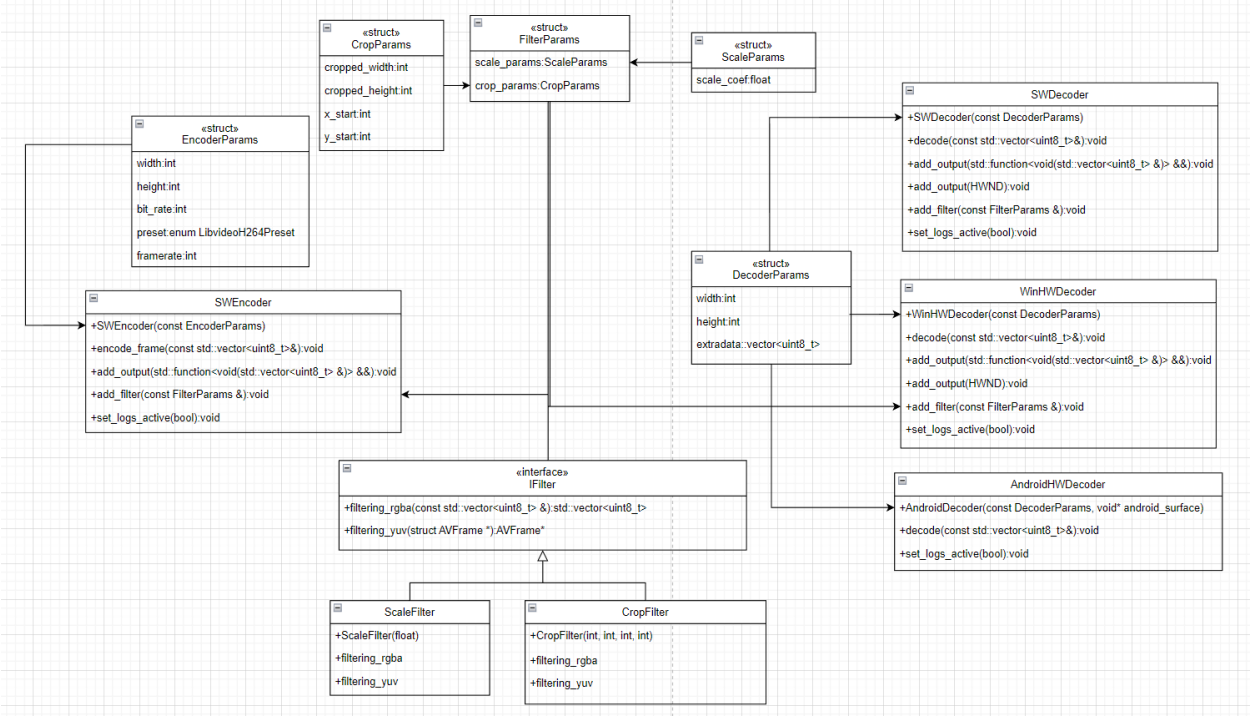


Рисунок 2.13 – Диаграмма классов библиотеки

2.5 Выводы по главе

В данной главе рассмотрены и спроектированы основные сущности, структуры и классы разрабатываемой библиотеки кодирования и декодирования видео.

Показана взаимосвязь сущностей, построена общая диаграмма классов библиотеки, представлены алгоритмы работы основных частей в виде блок-схем.

В результате проведенного детального проектирования будет создана библиотека, на основе которой будут проведены тесты производительности.

3 Программная реализация кроссплатформенной библиотеки кодирования и декодирования видео

3.1 О библиотеке

Для создания библиотеки кодирования и декодирования видео с возможностью кадрирования и масштабирования видео кадра использована среда разработки Visual Studio Code. Выбранными языками программирования стали C++ и Java. Для ОС Windows и ядра библиотеки использовался C++, для программирования под ОС Android использовался Java. В ходе реализации использовались стандартные библиотеки обоих языков, а также библиотека обработки видео FFmpeg.

Для сборки библиотеки использовалось средство автоматизации сборки программного обеспечения CMake, для подключения сторонних библиотек использовался пакетный менеджер Conan.

Для удобства сборки библиотеки на языке Python реализована сборка с необходимыми настройками.

Для автоматизации форматирования и единого code-style использована утилита clang-format. Для статической проверки кода использована утилита clang-tidy.

По определению библиотека не имеет визуального интерфейса и не может быть запущена без стартовой программы, использующей эту библиотеку.

3.2 Интерфейсы взаимодействия с библиотекой

В листинге 3.1 представлен интерфейс создания кодера и декодера.

Листинг 3.1. Интерфейс создания кодера и декодера

```
enum EncoderTypes CheckAvailableEncoders();
```

```
enum DecoderTypes CheckAvailableDecoders();

IEncoder *CreateEncoder(const EncoderParams &enc_params, enum
EncoderTypes enc_type);

#ifdef __ANDROID_API__
#if __ANDROID_API__ > 20
IDecoder *CreateDecoder(const DecoderParams &dec_params, enum
DecoderTypes dec_type, void *surface = nullptr);
#endif
#else
#ifdef WIN32
IDecoder *CreateDecoder(const DecoderParams &dec_params, enum
DecoderTypes dec_type);
#endif
#endif
```

Из листинга видно, что метод создания декодера разный для ОС Windows и ОС Android. Пользователь в своем коде может узнать какие именно кодеры и декодеры присутствуют сейчас на устройстве, если никаких кроме программных нет, то использоваться будут программные. После создания IEncoder или IDecoder, пользователю возвращается указатель на конкретную реализацию кодера или декодера.

Реализация методов листинга представлена в Приложении Б.

В листинге 3.2 представлен интерфейс кодера.

Листинг 3.2. Интерфейс декодера

```
class IEncoder {
  public:
      virtual void add_output(std::function<void(std::vector<uint8_t>
&)> &&out_callback) = 0;

      virtual void add_filter(const struct FilterParams &f_params) = 0;
```

```
    virtual void encode_frame(std::vector<uint8_t> &video_frame) = 0;

    virtual void set_logs_active(bool enable_logs) = 0;

    virtual ~IEncoder() = default;
};
```

Пользователь в кодере может добавить callback для выходных данных, добавить фильтр кадрирования и масштабирования, провести кодирование кадра, включить и выключить логирование.

Реализация кодера на примере SWEncoder представлена в Приложении Б.

В листинге 3.3 представлен интерфейс декодера.

Листинг 3.3. Интерфейс декодера

```
class IDecoder {
  public:
#ifdef WIN32
    virtual void add_output(HWND hwnd) = 0;
#endif
    virtual  void  add_output(std::function<void(std::vector<uint8_t>
&)> &&out_callback) = 0;

    virtual void add_filter(const struct FilterParams &f_params) = 0;

    virtual void decode(std::vector<uint8_t> &encoded_data) = 0;

    virtual void set_logs_active(bool enable_logs) = 0;

    virtual ~IDecoder() = default;
};
```

Пользователь в декодере может добавить callback для выходных данных, добавить фильтр кадрирования и масштабирования, провести декодирование кадра, включить и выключить логирование.

Реализация кодера на примере SWDecoder представлена в Приложении Г.

3.3 Реализация сборки библиотеки

Для удобства сборки библиотеки пользователем на языке Python написан скрипт, позволяющий в командной строке прописать необходимые параметры сборки.

В листинге 3.4 представлена команда, собирающая под ОС Windows библиотеку в релизной сборке.

Листинг 3.4. Команда, собирающая библиотеку в релизной сборке

```
py .ci/prebuild_thirdparty.py --build-release
```

В таблице 3.1 приведены все параметры командной строки с описанием.

Таблица 3.1 – Параметры сборки библиотеки с описанием

Параметр командной строки	Описание
--build-release / --build-debug	Взаимозаменяемые параметры определяющие какой тип сборки необходим
--enable-amf / --disable-amf	Взаимозаменяемые параметры включающие и выключающие аппаратный кодировщик AMD AMF
-cflags	При включенном аппаратном кодировщике AMD AMF необходимо ввести данный параметр и через пробел написать путь до включений кодировщика
--android	При использовании собирает библиотеку под ОС

	Android
--enable-tidy / --disable-tidy	Взаимозаменяемые параметры включающие и выключающие статическую проверку кода

Код скрипта представлен в Приложении Д.

3.4 Выводы по разделу

В данной главе приведено описание последовательной программной реализации спроектированной ранее кроссплатформенной библиотеки кодирования и декодирования видео с возможностью кадрирования и масштабирования видео. Библиотека представляет собой .dll файл для ОС Windows и .aar файл для ОС Android, написанные на языках C++ и Java.

В главе подробно описан интерфейс взаимодействия с библиотекой, основанный на UML-диаграммах предшествующего раздела, главные функции доступные пользователю (Приложение В - Г), а также средство сборки написанное на языке Python (Приложение Д).

4 Тестирование кроссплатформенной библиотеки кодирования и декодирования видео

Для проверки правильной работы основных компонентов системы необходимо провести ручное тестирование посредством создания приложений на Windows, Android и создать unit-тесты.

4.1 Windows тестовое приложение

Для реализации тестового приложения с помощью Windows API отдельно от основного проекта создано Windows окно, на которое будет выводится кодированный и декодированный кадр. Также на окне выводится статистика по видео: разрешение (resolution); битрейт (bitrate); количество кадров в секунду (framerate); отметка времени презентации (pts).

Тестирование будет для четырех разных режимов работы: программный кодер и декодер; программный кодер и аппаратный декодер; аппаратный декодер и программный кодер; аппаратный кодер и декодер.

На рисунке 4.1 отображена статистика для кодирования и декодирования кадра.

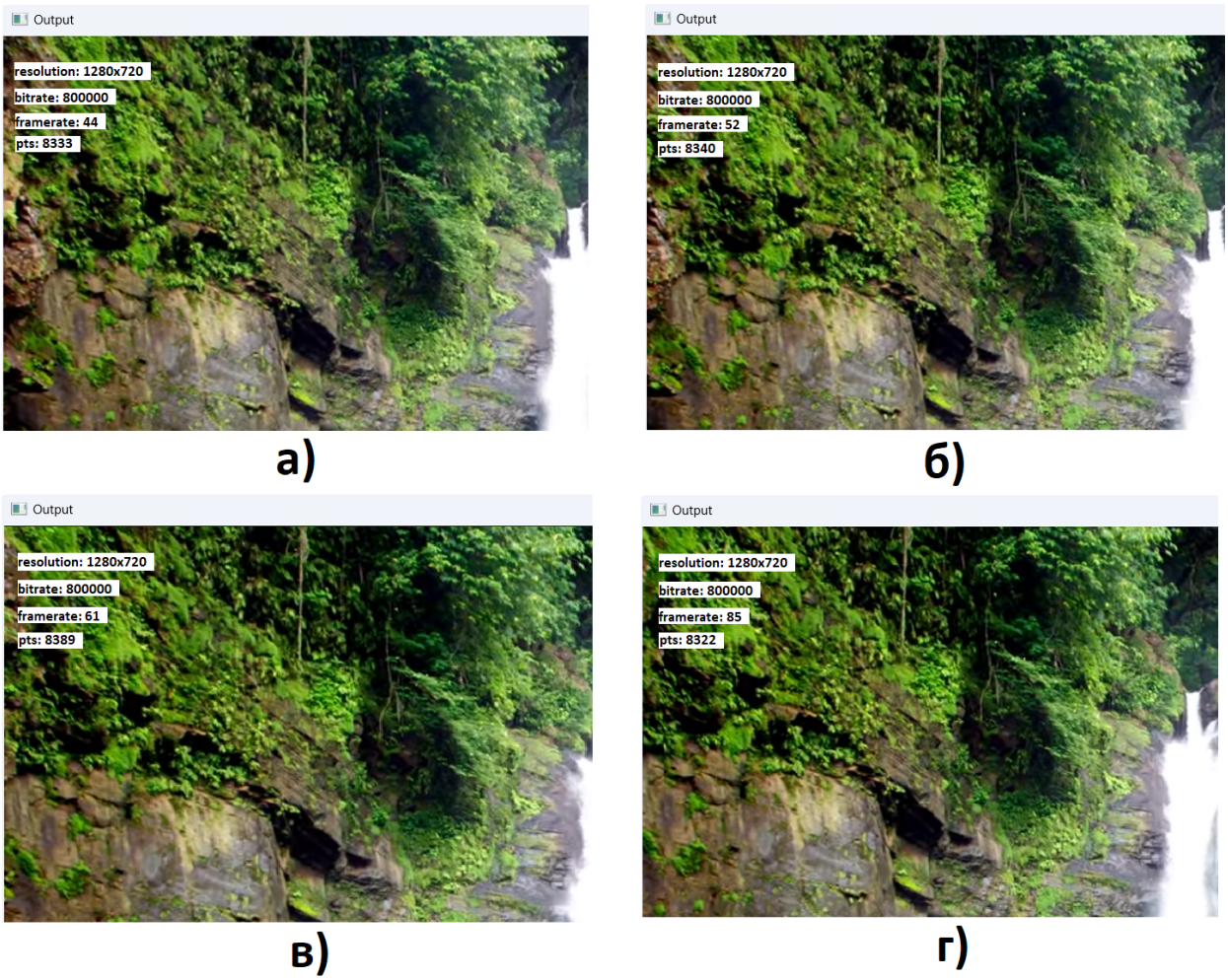
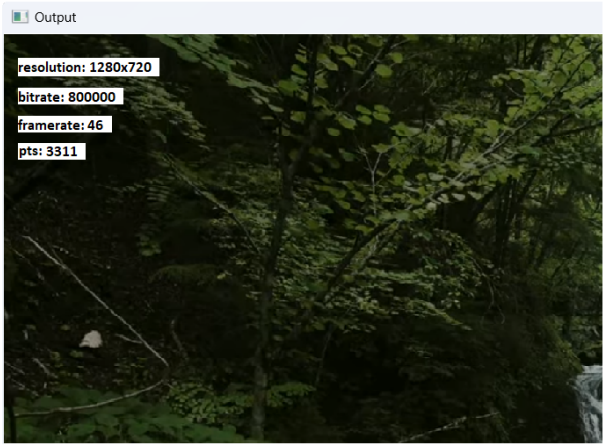


Рисунок 4.1 – Скриншоты работы библиотеки при разных режимах работы: а – программный кодер и декодер; б – программный кодер и аппаратный декодер; в – аппаратный декодер и программный кодер; г – аппаратный кодер и декодер

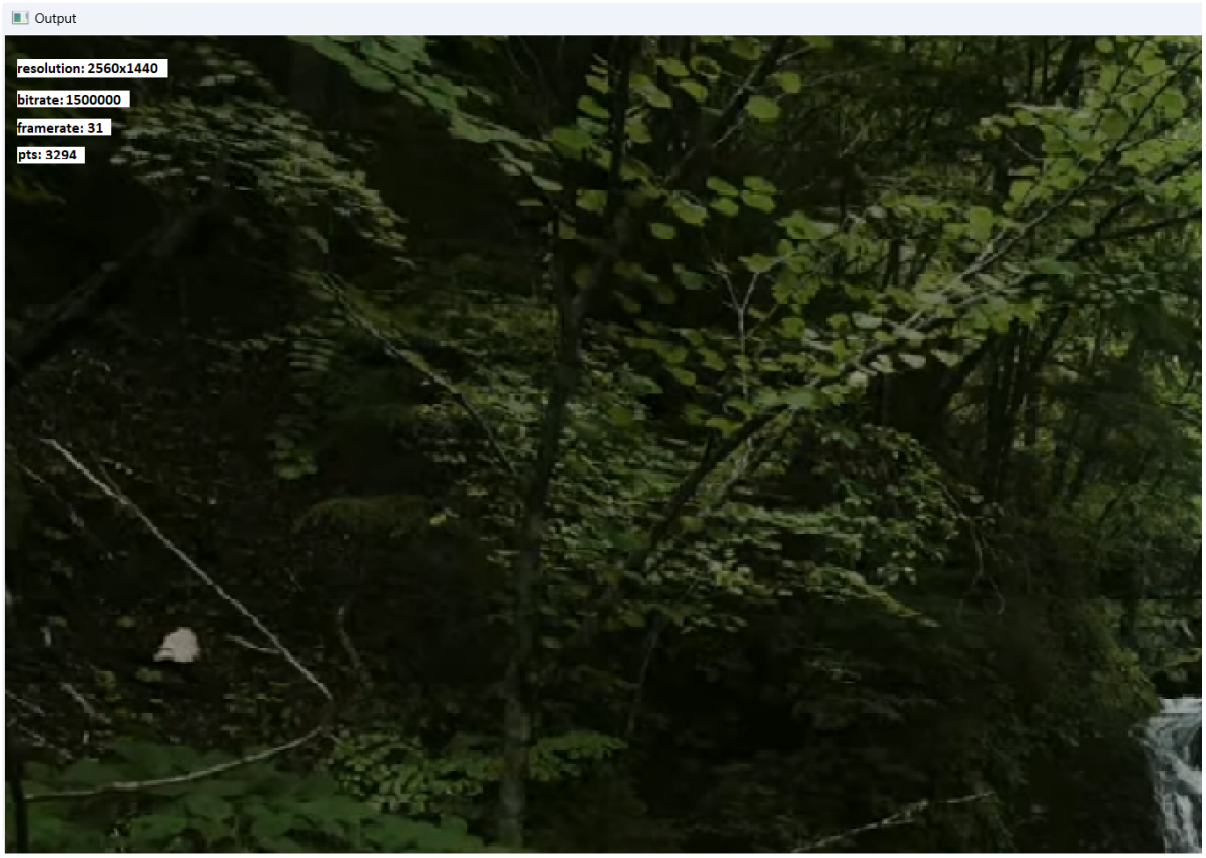
Как видно из рисунка худший результат показывает программный кодер и декодер, а лучший аппаратный кодер и декодер. Также из рисунка видно, что использование аппаратного декодера на данной системе позволяет получить более высокую производительность, чем использование аппаратного кодера.

В сравнении с данными кодера x264 прироста или падения производительности не обнаружено.

Затем проведено тестирование фильтра масштабирования видео. На рисунке 4.2 представлено два кадра: изначальный и увеличенный в два раза.



а)



б)

Рисунок 4.2 – Скриншоты работы библиотеки: а – изначальный видеокадры; б – масштабированный в 2 раза видеокадр.

Как видно из рисунка при увеличении видео в 2 раза качество изображение падает, уменьшается производительность, так как системе необходимо кодировать и декодировать кадр с большим разрешением.

4.2 Android тестовое приложение

Для Android написано два тестовых приложения, использующие основные методы библиотеки.

Первое приложение имеет два окна. На верхнем окне отображается кадр, взятый с камеры телефона, на нижнем кодированный и декодированный кадр. На рисунке 4.3 представлен результат работы первого тестового приложения.

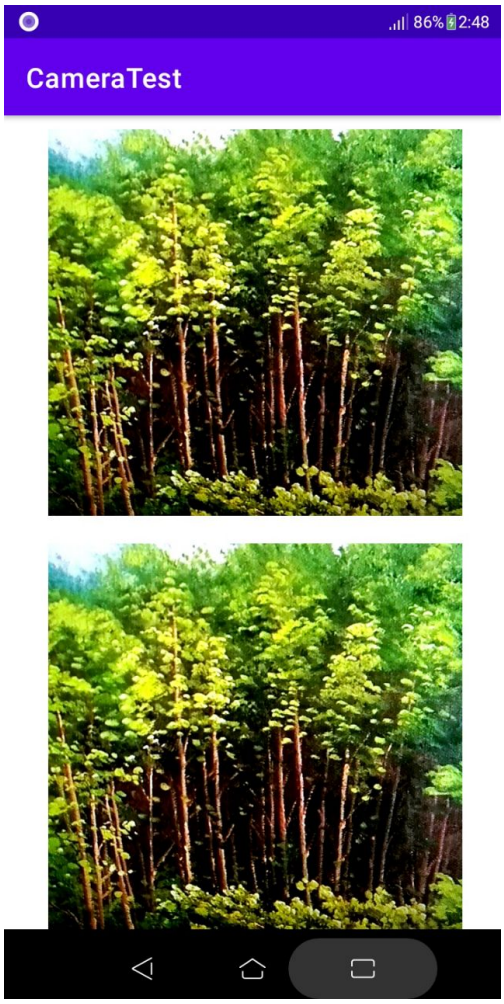


Рисунок 4.3 – Android тестовое приложение с программным кодированием декодированием

Второе приложение имеет одно окно, созданное как Android Surface, на которое выводит аппаратно декодированный кадр. На рисунке 4.4 представлен результат работы второго тестового приложения.

Рисунок 4.4 – Android тестовое приложение с аппаратным декодированием

4.3 Unit-тестирование

Создано 9 unit-тестов проверяющих основные компоненты системы. Для тестов используется две фикустуры, одна используется для подготовки данных с использованием фильтров, вторая для подготовки данных без использования фильтров.

В таблице 4.1 представлены названия тестов и их краткое описание.

Таблица 4.1 – Unit-тесты библиотеки

Имя теста	Описание
is_scaled_frame_has _scaled_size_enc_yuv	Проверка правильности работы фильтра масштабирования для кодера в YUV цветовом пространстве
is_cropped_frame_has _cropped_size_enc_yuv	Проверка правильности работы фильтра кадрирования для кодера в YUV цветовом пространстве
is_scaled_frame_has	Проверка правильности работы фильтра масштабирования

_scaled_size_enc_rgba	для кодера в RGBA цветовом пространстве
is_cropped_frame_has _cropped_size_enc_rgba	Проверка правильности работы фильтра кадрирования для кодера в RGBA цветовом пространстве
is_scaled_frame_has _scaled_size_dec_yuv	Проверка правильности работы фильтра масштабирования для декодера в YUV цветовом пространстве
is_cropped_frame_has _cropped_size_dec_yuv	Проверка правильности работы фильтра кадрирования для декодера в YUV цветовом пространстве
is_scaled_frame_has _scaled_size_dec_rgba	Проверка правильности работы фильтра масштабирования для декодера в RGBA цветовом пространстве
is_cropped_frame_has _cropped_size_dec_rgba	Проверка правильности работы фильтра кадрирования для декодера в RGBA цветовом пространстве
is_decoded_and_ initial_frame_has_same_size	Проверка правильности работы кодирования и декодирования без использования фильтров

На рисунке 4.5 представлено сообщение, выведенное gtest о успешном завершении всех тестов.

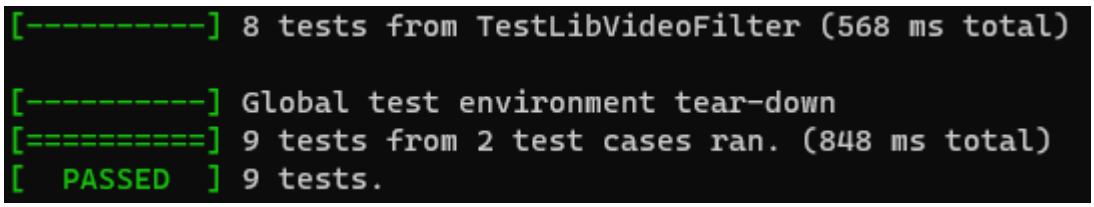


Рисунок 4.5 – Успешное завершение всех тестов

4.4 Сравнение производительности библиотеки с библиотекой x264

При разработке библиотеки для кодирования и декодирования использовалась открытая библиотека x264 с помощью FFmpeg.

Проведем сравнение среднего количества кадров в секунду за 30-ти секундное видео для созданной библиотеки и библиотеки x264.

Сравнение проведено на ПК с центральным процессором AMD Ryzen 5 5600G и видеокартой Radeon RX 590.

На рисунке 4.6 представлена диаграмма сравнения количества кадров в секунду для 30-ти секундного видео для созданной библиотеки (libvideo) и библиотеки x264 для разрешения 1280x720.

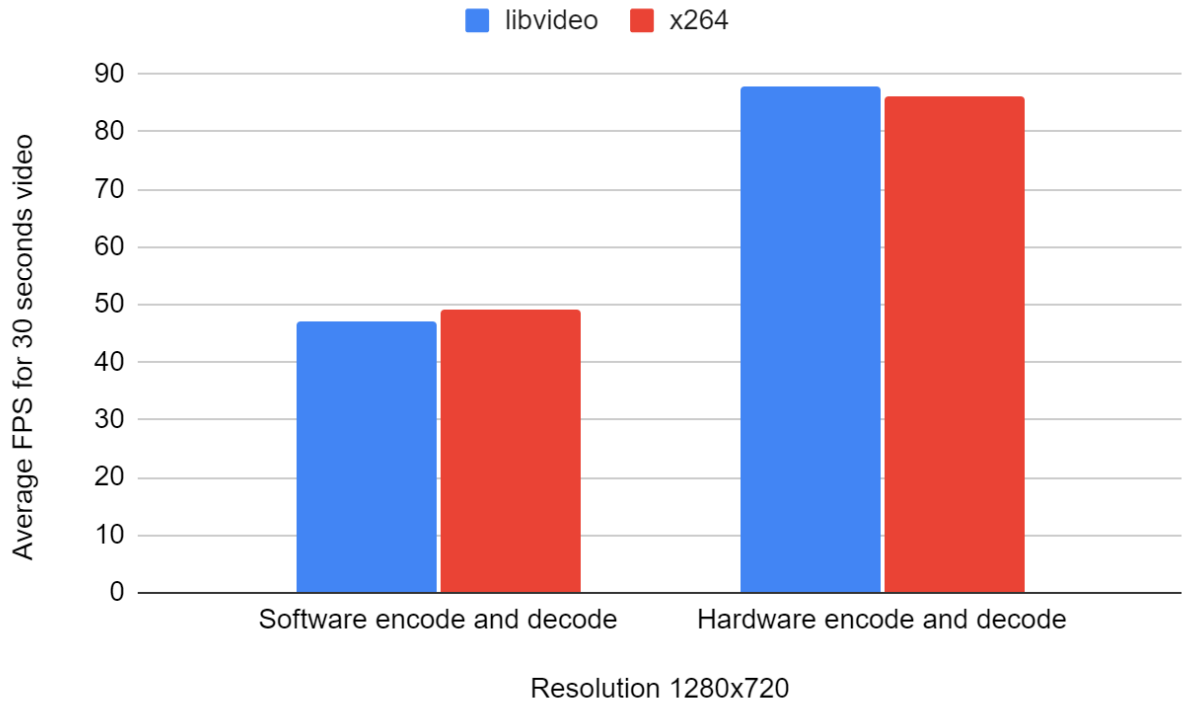


Рисунок 4.6 – Диаграмма сравнения количества кадров в секунду для 30-ти секундного видео для созданной библиотеки (libvideo) и библиотеки x264 для разрешения 1280x720

На рисунке 4.7 представлена диаграмма сравнения количества кадров в секунду для 30-ти секундного видео для созданной библиотеки (libvideo) и библиотеки x264 для разрешения 1920x1080.

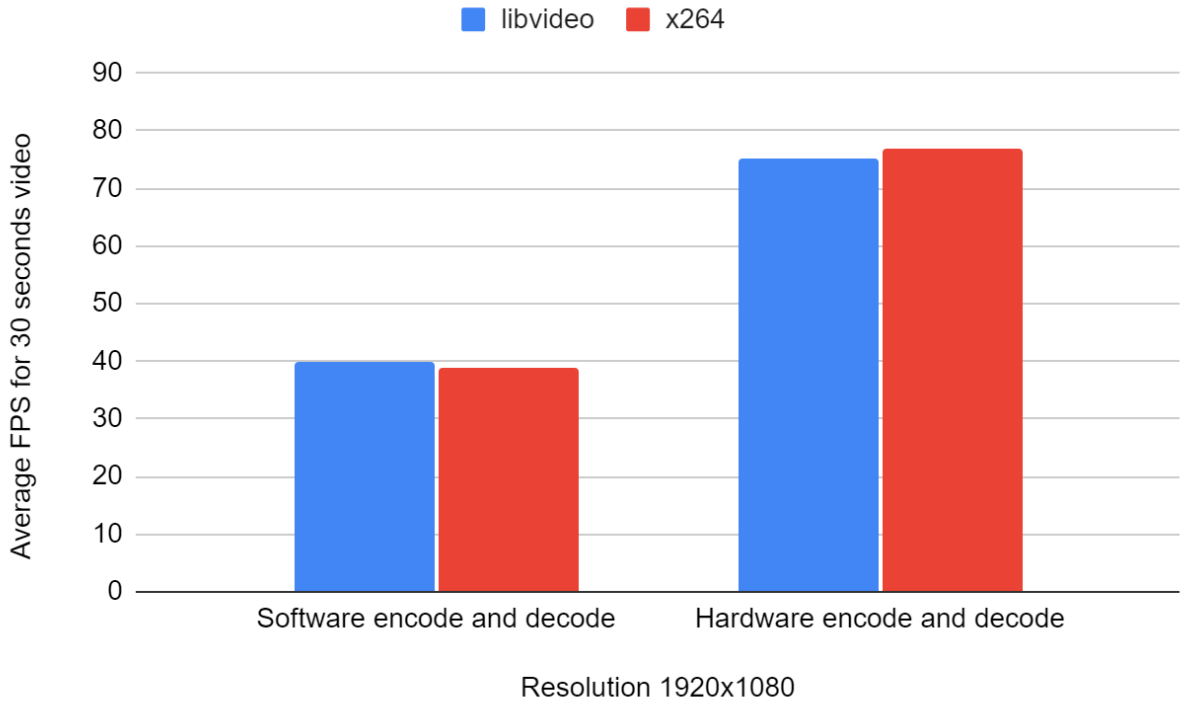


Рисунок 4.7 – Диаграмма сравнения количества кадров в секунду для 30-ти секундного видео для созданной библиотеки (libvideo) и библиотеки x264 для разрешения 1920x1080

Как видно из рисунков 4.6 и 4.7 количество кадров в секунду близки, различие не превышает 2,5% и могут быть объяснены сторонними процессами в системе.

Такое поведение объясняется идиомой Zero Cost Abstraction.

4.5 Выводы по разделу

В данной главе описано создание приложения для Windows, двух приложений для Android и девяти unit-тестов и показана их работы.

Windows приложение подразумевает под собой Winodws окно, на котором выводятся кодированные и декодированные видеокадры вместе с необходимой статистикой. Как было показано, наилучшая

производительность наблюдается при использовании аппаратного кодера и декодера. Показана работа фильтров масштабирования.

Android приложения кодируют и декодируют кадры с камеры телефона, а затем отображают их на экране. В одном случае декодирование программное, в другом аппаратное. Аппаратное кодирование показало лучшую производительность.

Создано 9 unit-тестов, проверяющих правильность работы библиотеки. Все тесты прошли успешно.

Произведено сравнение производительности созданной библиотеки и библиотеки x264 при кодировании и декодировании одного и того же видео. Выяснено, что различия несущественны и объясняются погрешностью. Таким образом, дополнительные абстракции в созданной не несут издержек по производительности.

ЗАДАНИЕ ДЛЯ РАЗДЕЛА

«ФИНАНСОВЫЙ МЕНЕДЖМЕНТ, РЕСУРСОЭФФЕКТИВНОСТЬ И РЕСУРСОСБЕРЕЖЕНИЕ»

Студенту:

Группа	ФИО
8ИМ11	Дмитриеву Василию Витальевичу

Школа	ИШИТР	Отделение школы (НОЦ)	ОИТ
Уровень образования	Магистратура	**Направление/ специальность**	09.04.02 Информационные системы и технологии

Исходные данные к разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»:	
1. Стоимость ресурсов научного исследования (НИ): материально-технических, энергетических, финансовых, информационных и человеческих	Бюджет проекта – не более 500000 руб., в т.ч. затраты по оплате труда – не более 300000 руб.
2. Нормы и нормативы расходования ресурсов	Значение показателя интегральной ресурсоэффективности – не менее 3,9 баллов из 5
3. Используемая система налогообложения, ставки налогов, отчислений, дисконтирования и кредитования	Коэффициент отчислений на уплату во внебюджетные фонды – 30%.
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. Оценка коммерческого и инновационного потенциала НТИ	Анализ конкурентных технических решений; SWOT-анализ.
2. Планирование процесса управления НТИ: структура и график проведения, бюджет, риски и организация закупок	Формирование плана и графика разработки: - определение структуры и трудоемкости работ; - создание диаграммы Ганта. Формирование бюджета затрат на разработку: - затраты на амортизацию оборудования; - заработная плата (основная и дополнительная); - отчисления во внебюджетные фонды; - накладные расходы.
3. Определение ресурсной, финансовой, экономической эффективности	Определение потенциального эффекта разработки; Расчет рисков.
Перечень графического материала (с точным указанием обязательных чертежей):	
1. Оценка конкурентоспособности технических решений *2.* Матрица SWOT *3.* График проведения и бюджет НТИ *4.* Оценка ресурсной, финансовой и экономической эффективности НТИ *5.* Потенциальные риски	

Дата выдачи задания для раздела по линейному графику	

Задание выдал консультант:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОСГН ШБИП ТПУ	Спицына Любовь Юрьевна	к.э.н.		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8ИМ11	Дмитриев Василий Витальевич		

5 Финансовый менеджмент, ресурсоэффективность и ресурсосбережение

Целью выполнения раздела «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение» является комплексное описание и анализ финансово-экономических аспектов проекта. Необходимо оценить полные денежные затраты на проект и дать приближенную реалистичную экономическую оценку результатов его внедрения, что в свою очередь позволит оценить экономическую целесообразность выполненной работы, используя традиционные показатели эффективности инвестиций.

Для достижения данной цели были поставлены и решены следующие задачи:

- провести предпроектный анализ;
- спланировать продолжительность этапов работ и график выполнения этих работ в рамках проекта;
- рассчитать смету затрат на выполнение проекта;
- определить стоимость разработки проекта;
- оценить экономическую эффективность проекта.

Потребителями данного продукта могут быть юридические и физические лица заинтересованные в разработке ПО направленного на обработку видео. Существует множество компаний, как в РФ, так и по всему миру, которые занимаются обработкой видео, они все будут являться потенциальными клиентами. Потенциальными клиентами могут быть и обычные люди, занимающиеся обработкой видео, так как данный проект является Open Source решением.

5.1 Предпроектный анализ

5.1.1 Анализ конкурентных технических решений

Результатом разработки является кроссплатформенная библиотека для кодирования и декодирования видео под операционные системы Windows и Android. Прямыми конкурентами являются следующие решения: FFmpeg - набор свободных библиотек с открытым исходным кодом, которые позволяют записывать, конвертировать и передавать цифровые аудио- и видеозаписи в различных форматах; x264 - свободная библиотека программных компонентов для кодирования видеопотоков H.264; openh264 - бесплатная программная библиотека для кодирования и декодирования видеопотоков в реальном времени в формате H.264.

Выбраны следующие критерии конкурентоспособности:

- легкость работы с библиотекой (легкость работы);
- количество предоставляемых функций (функциональность);
- количество поддерживаемых платформ (кроссплатформенность);
- простота начала работы с библиотекой (простота запуска).

Для каждого из критериев был оценен его вес, далее все конкурентные решения экспертным методом были оценены по десятибалльной шкале по приведенным критериям, на основании чего были получены итоговые оценки конкурентоспособности. В таблице 5.1 представлена оценочная карта конкурентных технических решений. Данную работу именуем, как “Проект”.

Таблица 5.1 – Оценочная карта конкурентных технических решений

Критерий оценки	Вес критерия	Баллы				Конкурентоспособность			
		Проект	FFmpeg	x264	openh264	Проект	FFmpeg	x264	openh264
Технические критерии оценки ресурсоэффективности									
Легкость работы	0,2	9	4	6	6	2,7	1,2	1,8	1,8
Функциональность	0,2	5	8	6	6	1	1,6	1,2	1,2

Кроссплатформенность	0,2	6	8	7	7	1,2	1,6	1,4	1,4
Простота запуска	0,2	8	4	5	5	2,4	1,2	1,5	1,5
Экономические критерии оценки эффективности									
Конкурентоспособность	0,1	6	8	7	7	0,6	0,8	0,7	0,7
Послепродажное обслуживание	0,1	8	6	7	7	0,8	0,6	0,7	0,7
Итого:	1	9	4	6	6	7	6,2	6,2	6,2

Из таблицы 5.1 видно, что предлагаемый в работе подход имеет наибольшую конкурентоспособность, основным преимуществом является легкость работы с библиотекой, сочетаемой с простатой начала работы с библиотекой.

5.1.2 SWOT-анализ

Для формирования стратегии позиционирования продукта (в нашем случае, библиотеки кодирования и декодирования видео) необходимо учитывать следующие данные ситуационного анализа – SWOT-анализ. Предварительно вникнув в сильные и слабые стороны организации, была составлена матрица SWOT для проекта представленная на рисунке 5.1.

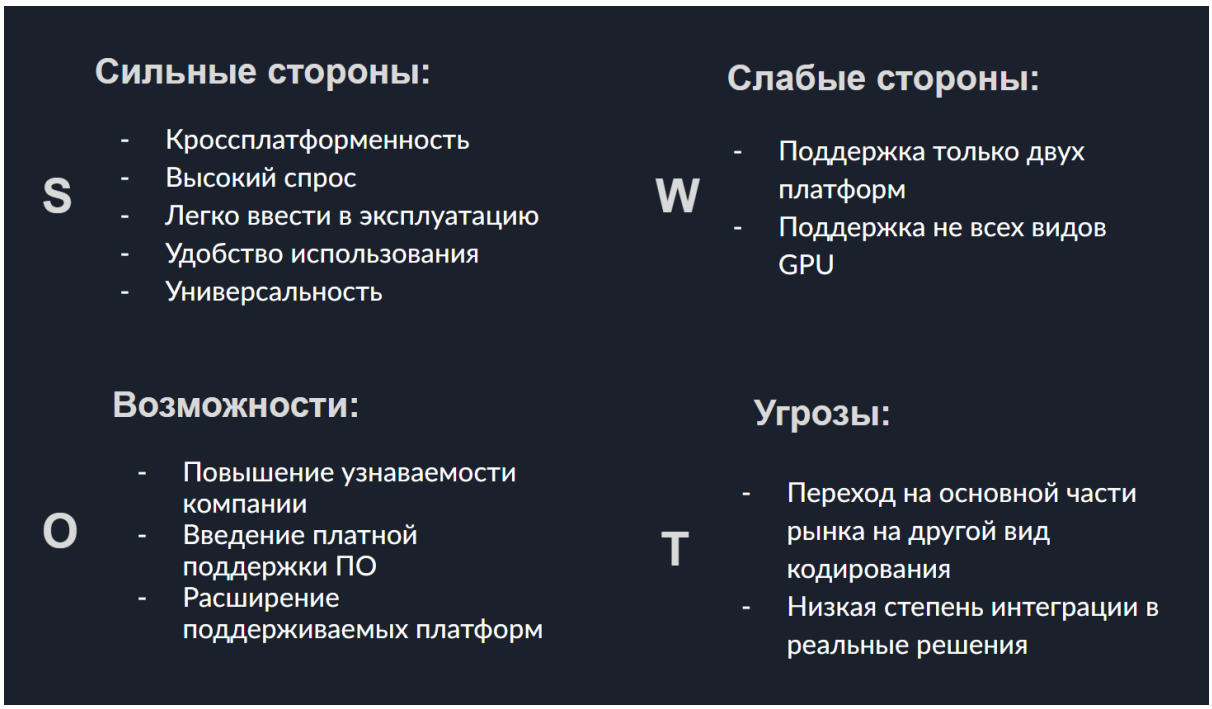


Рисунок 5.1 – Матрица SWOT

Рассмотрим все возможные сочетания сильных сторон с возможностями и угрозами, а также слабых сторон с возможностями и угрозами. Оценим по пятибалльной шкале степень взаимодействия слабых и сильных сторон, возможностей и угроз. На пересечении строки и столбца укажем балльную оценку значимости. Балльная оценка позволяет более точно понять значимость отдельных факторов. Результаты балльной оценки сильных и слабых сторон внутренней среды, а также угроз и возможностей внешней среды представлены в таблице 5.2.

Также в таблице 5.2, в строке и столбце «Итого» приведена совокупная балльная оценка сильных и слабых сторон внутренней среды предприятия, а также возможностей и угроз внешней среды.

Таблица 5.2 – Оценка взаимодействия сторон с возможностями и угрозами

Факторы	**Сильные стороны**					**Слабые стороны**		**Итого: (из 35)**
	Кроссплатформенность	Высокий спрос	Легко ввести в	Удобство исполь	Универсальнос	Поддержка только	Поддержка не	

				эксплуатацию	зования	ть	двух платформ	всех видов GPU	
Возможности	Повышение узнаваемости компании	5	5	5	4	4	3	5	31
	Введение платной поддержки ПО	5	4	5	4	4	2	2	27
	Расширение поддерживаемых платформ	5	5	4	5	4	5	2	30
Угрозы	Переход на основной части рынка на другой вид кодирования	3	2	2	2	4	2	1	16
	Низкая степень интеграции в реальные решения	4	5	4	4	5	3	2	27
Итого (из 25):		22	21	20	19	21	15	12	

По результатам анализа данных, представленных в таблице можно сделать вывод, что в проекте более выражены сильные стороны. Наибольшее значение среди сильных сторон имеет кроссплатформенность, а также расширение количества поддерживаемых платформ и возможность ввести платную поддержку ПО. Среди слабых сторон следует обратить внимание на поддержку только ограниченного количества GPU (графический процессор) для кодирования и декодирования.

Лидирующую позицию в возможностях занимает повышение узнаваемости компании, что является важным в современном мире. Основной угрозой является переход большей части рынка на принципиально

другой вид кодирования и декодирования видео, но в ближайшее время это произойдет с малой вероятностью, следует следить за рынком.

Таким образом, проведенный SWOT-анализ показал, что реализация рассматриваемого проекта будет способствовать повышению эффективности эффективности сектора обработки видео и может быть рентабельным.

5.1.3 Оценка готовности проекта к коммерциализации

В таблице 5.3 представлена оценка готовности проекта к коммерциализации.

Таблица 5.3 – Оценка готовности проекта к коммерциализации

	Наименование	Степень проработанности научного проекта	Уровень имеющихся знаний у разработчика
1.	Определен имеющейся научно-технический раздел	4	3
2.	Определены перспективные направления коммерциализации научно-технического задела	4	4
3.	Определены отрасли и технологии (товары, услуги) для предложения на рынке	4	3
4.	Определена товарная форма научно-технического задела для представления на рынок	3	4
5.	Определены авторы и осуществлена охрана их прав	5	5
6.	Проведена оценка стоимости интеллектуальной собственности	5	4
7.	Проведены маркетинговые исследования рынков сбыта	2	2
8.	Разработан бизнес-план коммерциализации научной разработки	3	3
9.	Определены пути продвижения	3	4

	научной разработки		
10.	Разработана стратегия (форма) реализации научной разработки	4	4
11.	Проработаны вопросы международного сотрудничества и выхода на зарубежный рынок	1	2
12.	Проработаны вопросы использования услуг инфраструктуры поддержки, получения льгот	2	2
13.	Проработаны вопросы финансирования коммерциализации научной разработки	4	4
14.	Имеется команда для коммерциализации научной разработки	5	5
15.	Проработан механизм реализации научного проекта	4	4
Итого:		53	53

Как видно из таблицы 5.3 по двум показателям сумма баллов находится между 59 и 45 баллами, поэтому коммерческая перспективность разработки выше среднего.

5.2 Инициация проекта

В реализации данного проекта заинтересованы несколько сторон, стороны представлены в таблице 5.4.

Таблица 5.4 – Заинтересованные стороны проекта

Заинтересованные стороны проекта	**Ожидания заинтересованных сторон**
Исполнитель	Получение одобрения от компании и университета о успешном завершении работ
Университет, научный руководитель	Правильно подготовленный и

	составленный отчет о научном-техническом проекте
Организация, руководитель от организации	Разработанный в соответствии с поставленными требованиями программный продукт

У проекта есть цели, ожидаемые результаты, критерии приемки результата проекта и требования, представленные в таблице 5.5

Таблица 5.5 – Цели и результат проекта

Цели проекта:	- Создать конкурентное ПО - Выйти на рынок Open Source решений
Ожидаемые результаты проекта:	- готовая к распространению кроссплатформенная библиотека с поддержкой заявленного функционала
Критерии приемки результата проекта:	- Количество багов, изъянов - Грамотность документации - Производительность - Соответствие реального функционала заявленному
Требования к результату проекта:	**Требование:**
	Поддержка заявленных платформ
	Выполнение заявленного функционала

В проекте задействованы несколько участников, представленные в таблице 5.6.

Таблица 5.6 – Рабочая группа проекта

№ п/п	ФИО, основное место работы, должность	Роль в проекте	Функции	Трудозатраты, час.
1.	Трошин Максим Вадимович, ООО “Томсксофт-Р”, технический директор	Руководитель от предприятия	Контроль, консультирование по вопросам разработки	15,66
2.	Чердынцев Евгений Сергеевич, НИ ТПУ,	Научный руководитель	Контроль, консультирование по	20,56

	доцент		вопросам ВКР	
3.	Дмитриев Василий Витальевич, ООО “Томскcофт” и НИ ТПУ, программист-инженер, студент	Исполнитель	Разработка проекта, оформление необходимых документов	91,20
Итого:				127,42

Ограничения и допущения проекта представлены в таблице 5.7.

Таблица 5.7 – Ограничения проекта

Фактор	**Ограничения/допущения**
Бюджет проекта	до 500000 руб.
Источник финансирования	Компания, НИ ТПУ
Сроки проекта	06.02.2023 - 18.05.2023

5.3 Организация и планирование работ

При организации процесса был определен полный перечень необходимых работ, а также их исполнители и рациональная продолжительность. В таблице 5.7 представлен полный перечень производимых работ, данные в котором хронологически упорядочены.

Таблица 5.7 – Перечень работ и продолжительность их выполнения

Этапы работы	Исполнители	Загрузка исполнителей
Постановка целей и задач	НР, РП	НР – 100% РП – 100%
Составление и утверждение ТЗ	НР, РП, И	НР – 100% И – 10% РП – 30%
Подбор и изучение литературы и аналогов по тематике	НР, И	НР – 10% И – 100%
Разработка календарного плана	НР, РП, И	НР – 100% И – 10% РП – 10%
Анализ исследуемой	И, РП	И – 100%

области		РП - 20%
Проектирование программной системы	И, РП	И – 100% РП - 20%
Разработка программной системы	И, РП	И – 100% РП - 20%
Тестирование программной системы	И, РП	И – 100% РП – 20%
Оформление расчетно-пояснительной записки	И, НР	И – 100% НР - 40%
Подготовка к защите ВКР	НР, И	НР – 20% И – 100%
Примечание: НР – научный руководитель, РП – руководитель от предприятия, И – исполнитель		

5.2.1 Продолжительность этапов работ

Расчет продолжительности этапов работ проведен опытно-статическим методом с помощью экспертного способа. Для определения вероятных (ожидаемых) значений продолжительности работ применена следующая формула (5.2.1):

$$t_{ож} = \frac{3 * t_{min} + 2 * t_{max}}{5}, \quad (5.2.1)$$

где t_{min} - минимальная продолжительность, дн.;

t_{max} - максимальная продолжительность работы, дн.

Для построения линейного графика также была рассчитана длительность этапов в рабочих днях, а после и в календарных. Расчет продолжительности выполнения каждого этапа в рабочих днях ($T_{РД}$) проводился по формуле (5.2.2):

$$T_{РД} = \frac{t_{ож}}{K_{ВН}} * K_{Д}, \quad (5.2.2)$$

где $K_{ВН}$ - коэффициент выполнения работ, учитывающий влияние внешних факторов на соблюдение предварительно определенных. Пусть $K_{ВН} = 1$;

где $K_Д$ - коэффициент, учитывающий дополнительное время на компенсацию непредвиденных задержек и согласование работ. Пусть $K_{ВН}=1{,}1$.

Формула расчета продолжительности этапа в календарных днях (5.2.3) представлена ниже:

$$T_{КД}=T_{РД}*T_К, \quad (5.2.3)$$

где $T_К$ - коэффициент календарности, позволяющий перейти от длительности работ в рабочих днях к их аналогам в календарных днях, и рассчитываемый следующим образом для шестидневной недели (5.2.4):

$$T_{К6}=\frac{T_{КАЛ}}{T_{КАЛ}-T_{ВД}-T_{ПД}}=\frac{365}{365-52-15}=1{,}22, \quad (5.2.4)$$

где $T_{КАЛ}$ - календарные дни, дн.;

$T_{ВД}$ - выходные дни, дн.;

$T_{КАЛ}$ - календарные дни, дн.

Коэффициент календарности для пятидневной недели (5.2.5):

$$T_{К5}=\frac{T_{КАЛ}}{T_{КАЛ}-T_{ВД}-T_{ПД}}=\frac{365}{365-118}=1{,}48. \quad (5.2.5)$$

Полученные результаты трудозатрат на выполнение проекта отображены в таблице 5.9, а график Ганта на рисунке 5.2.

Таблица 5.9 – Перечень работ и продолжительность их выполнения

Этап	Исполнители	Продолжительность работ, дн.			Трудоемкость работ по исполнителям, чел-дн.					
					$T_{РД}$			$T_{КД}$		
		t_{min}	t_{max}	$t_{ож}$	НР	РП	И	НР	РП	И
Постановка целей и задач	НР, РП	1	2	1,4	1,54	1,54	-	1,88	2,28	-
Составление и утверждение ТЗ	НР, РП, И	2	4	2,8	3,08	0,92	0,31	3,76	1,37	0,46
Подбор и изучение литературы и аналогов по	НР, И	3	6	4,2	4,62	-	4,62	5,64	-	6,84

тематике										
Разработка календарного плана	НР, РП, И	1	2	1,4	1,54	0,15	0,15	1,88	0,23	0,23
Анализ исследуемой области	И	6	8	6,8	-	1,50	7,48	-	2,21	11,07
Проектирование программной системы	И	8	14	10,4	-	2,29	11,44	-	3,39	16,93
Разработка программной системы	И	8	18	12	-	2,64	13,20	-	3,91	19,54
Тестирование программной системы	И	5	10	7	-	1,54	7,70	-	2,28	11,40
Оформление расчетно-пояснительной записки	И	10	16	12,4	5,46	-	13,64	6,66	-	20,19
Подготовка к защите ВКР	НР, И	2	4	2,8	0,62	-	3,08	0,75	-	4,56
			Итого:	61,2	16,85	10,58	61,62	20,56	15,66	91,20

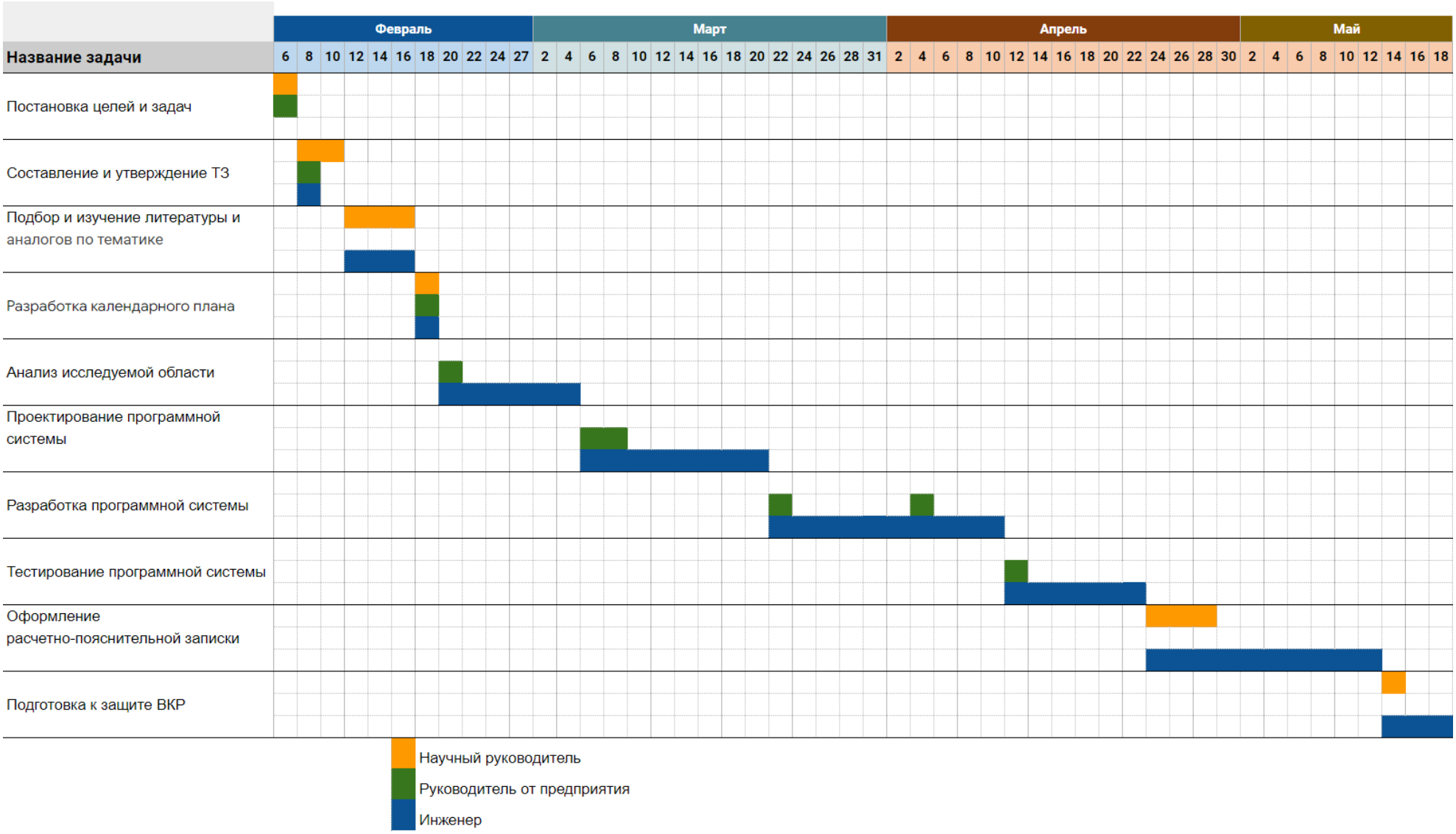


Рисунок 5.2 – Линейный график работ

5.3 Расчет сметы затрат на выполнение проекта

В состав затрат на создание проекта включается величина всех расходов, необходимых для реализации комплекса работ, составляющих содержание данной разработки. Расчет сметной стоимости ее выполнения производится по следующим статьям затрат:

- материалы и покупные изделия;
- заработная плата;
- социальный налог;
- расходы на электроэнергию (без освещения);
- амортизационные отчисления;
- прочие (накладные расходы) расходы.

Общая себестоимость проекта представлена в таблице 5.14 пункта 5.6.7.

5.3.1 Расчет затрат на материалы

К данной статье расходов относится стоимость материалов, покупных изделий, полуфабрикатов и других материальных ценностей, расходуемых непосредственно в процессе выполнения работ над объектом проектирования. Сюда же относятся специально приобретенное оборудование, инструменты и прочие объекты, относимые к основным средствам, стоимостью до 40 000 руб. включительно.

В процессе разработки программной системы использовались библиотеки с открытым исходным кодом и свободно распространяемый редактор кода. Поэтому расходы на использование лицензионного ПО равны нулю.

Расчет затрат конкретно данного проекта приведен в таблице 5.10.

Таблица 5.10 – Расчет затрат на материалы

Наименование материалов	Цена за ед., руб.	Кол-во	Сумма, руб.
Бумага для принтера формата А4	300	1 уп.	300
Картридж для принтера	1500	1 шт.	1500
		Итого:	1800

Допустим, что ТЗР составляют 5% от отпускной цены материалов, тогда расходы на материалы с учетом ТЗР равны $C_{мат} = 1800 * 1{,}05 = 1890$ руб.

5.3.2 Расчет заработной платы

Данная статья расходов включает заработную плату научного руководителя и исполнитель проекта, а также премии, входящие в фонд заработной платы.

Общая заработная плата равна:

$$C_{зп} = З_{осн} + З_{доп}, \quad (5.3.1)$$

где $З_{осн}$ – основная заработная плата,

$З_{доп}$ – дополнительная заработная плата.

Основная заработная плата рассчитывается по формуле:

$$З_{осн} = З_{дн} \cdot T_{раб}, \quad (5.3.2)$$

где $З_{дн}$ – среднедневная заработная плата,

$T_{раб}$ – продолжительность работ.

Среднедневная заработная плата рассчитывается по формуле:

$$З_{дн} = \frac{З_{м} \cdot M}{F_{д}}, \quad (5.3.3)$$

где $З_{м}$ – месячный оклад работника,

M – количество месяцев работы без отпуска в течении года (для 5-ти дневной рабочей недели 11,2, для 6-ти дневной рабочей недели 10,4),

$F_{д}$ – действительный годовой фонд рабочего времени работника.

Дополнительная заработная плата рассчитывается по формуле:

$$З_{доп} = k_{доп} \cdot З_{осн}, \quad (5.3.4)$$

где $k_{доп}$ – коэффициент дополнительной заработной платы.

Коэффициент $k_{доп}$ может быть различным, примем $k_{доп}$ равным 0,1.

Расчет затрат на полную заработную плату приведен в таблице 5.11.

Таблица 5.11 – Расчет затрат на заработную плату

Исполнитель	Оклад, руб./мес.	$З_{дн}$, руб./раб.дн.	Затраты времени, раб.дн.	$З_{осн}$, руб	$З_{доп}$, руб	$С_{зп}$, руб
НР	50000	1866,66	20,56	38378,53	3837,85	42216,38
РП	100000	4210,53	15,66	65936,90	6593,69	72530,59
И	35000	1473,68	91,20	134399,62	13439,96	147839,58
					Итого:	262586,55

5.3.3 Расчет затрат на социальный налог

Затраты на единый социальный налог (ЕСН), включающий в себя отчисления в пенсионный фонд, на социальное и медицинское страхование, составляют 30 % от полной заработной платы по проекту. Соответственно, для текущего проекта затраты на социальный налог равны (5.3.4):

$$С_{соц} = 262586{,}55 \cdot 0{,}30 = 78775{,}96. \quad (5.3.4)$$

5.3.4 Расчет затрат на электроэнергию

Данный вид расходов включает в себя затраты на электроэнергию, потраченную в ходе выполнения проекта на работу используемого оборудования, рассчитываемые по формуле (5.3.5):

$$С_{эл.об} = P_{об} * t_{об} * С_{э}, \quad (5.3.5)$$

где $P_{об}$ - мощность, потребляемая оборудованием, кВт;

$С_{э}$ – тариф на 1 кВт·час (3,16 руб./кВт·час);

$t_{об}$ - время работы оборудования, час.

Время работы оборудования вычисляется на основе итоговых данных таблицы 5.12 для исполнителя ($T_{РД}$) из расчета, что продолжительность рабочего дня равна 8 часов.

$$t_{об} = T_{РД} * K_t, \quad (5.3.6)$$

где $K_t \leq 1$ - коэффициент использования оборудования по времени, равный отношению времени его работы в процессе выполнения проекта к $T_{РД}$, определяется исполнителем самостоятельно.

Мощность, потребляемая оборудованием, определяется по формуле 5.3.7:

$$P_{об} = P_{ном} * K_C, \quad (5.3.7)$$

где $P_{ном}$ - номинальная мощность оборудования, кВт;

$K_C \leq 1$ - коэффициент загрузки, зависящий от средней степени использования номинальной мощности.

Определим время работы персонального компьютера (5.3.8), принимая коэффициент использования равным 1, так как все работы выполняются за ним:

$$t_{об} = (62 * 8) * 1 = 496. \quad (5.3.8)$$

Расчет затрат на электроэнергию для технологических целей приведен в таблице 5.12.

Таблица 5.12 – Расчет затрат на электроэнергию

Наименование оборудования	Время работы оборудования $t_{об}$, час	Потребляемая мощность $P_{об}$, кВт.	Затраты $C_{эл.об}$, руб.
Компьютер	496	0,75	1175,52
Принтер	1	0,3	0,95
		Итого:	1176,47

5.3.5 Расчет амортизационных расходов

В данной статье представлен расчет амортизации используемого оборудования за время выполнения проекта по формуле 5.3.9:

$$C_{АМ} = \frac{H_A * t_{об} * C_{об} * n}{F_д}, \quad (5.3.9)$$

где H_A - годовая норма амортизации единицы оборудования;

$C_{об}$ - балансовая стоимость единицы оборудования с учетом ТЗР;

$F_д$ - действительный годовой фонд времени работы соответствующего оборудования, берется из специальных справочников или фактического режима его использования в текущем календарном году ($F_д = 300 * 8 = 2400\,ч$);

$t_{об}$ - фактическое время работы оборудования в ходе выполнения проекта, учитывается исполнителем проекта;

n - число задействованных однотипных единицы оборудования.

Расчет для компьютера следующий:

$$C_{АМ} = \frac{0{,}4 * 496 * 100000 * 1}{2400} = 8266{,}66\,руб. \quad (5.3.10)$$

Расчет для принтера:

$$C_{АМ} = \frac{0{,}5 * 1 * 4600 * 1}{300} = 7{,}82\,руб. \quad (5.3.11)$$

Итого начислено амортизации 8274,48 руб.

5.3.6 Расчет прочих расходов

В статье «Прочие расходы» отражены расходы на выполнение проекта, которые не учтены в предыдущих статьях. Их следует принять равными 10% от суммы всех предыдущих расходов (5.3.12):

$$C_{проч} = (C \| мат + C_{зп} + C_{соц} + C_{эл.об} + C_{ам}) * 0{,}1. \quad (5.3.12)$$

Таким образом, прочие расходы на реализацию настоящего проекта составили:

$$C_{проч} = ¿$$

$$+ 8274{,}48 * 0{,}1 = 20051{,}36\,руб.$$

5.3.7 Расчет общей себестоимости

Проведя расчет по всем статьям сметы затрат на разработку, можно определить общую себестоимость проекта «Разработка метода оценки состояния тяжести пациента с использованием инструментария семантических сетей» (таблица 5.13).

Таблица 5.13 – Расчет себестоимости разработки проекта

Статья затрат	**Условное обозначение**	**Сумма, руб.**
Материалы и покупные изделия	$C_{мат}$	1800
Основная заработная плата	$C_{зп}$	262586,55
Отчисления в социальные фонды	$C_{соц}$	78775,96
Расходы на электричество	$C_{эл.об}$	1176,47
Амортизационные отчисления	$C_{ам}$	8274,48
Прочие расходы	$C_{проч}$	20051,36
Итого:		372664,82

5.3.8 Расчет прибыли

Данный проект разрабатывается для компании и предполагается к распространению через Open Source модель, компания не планирует коммерциализировать данный проект. Поэтому рассчитаем гипотетическую прибыль.

Всего в РФ на данный момент находится около 73000 IT компаний [19], предположим, что работой с видео занимается 5% компаний, то есть 3650 компаний. Предположим, что получится продавать одну копию ПО проекта 1/6 всех компаний, то есть примерно 600. Зададим цену равную 5000 руб. за единицу ПО. Тогда прибыль равна:

$$C_{приб} = 600 * 5000 = 3000000\ руб$$

5.3.9 Расчет НДС

НДС составляет 20% от суммы затрат на разработку и прибыли:

$$C_{НДС} = (3000000 + 220565) * 0{,}2 = 644113\ руб$$

5.3.10 Цена разработки НИР

Цена равна сумме полной себестоимости, прибыли и НДС:

$$Ц_{НИР} = 3000000 + 220565 + 644113 = 3864678\ руб$$

5.4 Определение ресурсной, финансовой, бюджетной, социальной и экономической эффективности исследования

В данном подразделе сравнивается разработка, описанная в данной работе, со наиболее прямым конкурентом – openh264, описанным в подразделе 5.1.1, так как данный аналог решает похожую задачу.

5.4.1 Определение финансовой эффективности исследования

Интегральный финансовый показатель разработки для i-го варианта исполнения рассчитывается как:

$$I_{ф}^{исп.i} = \frac{Ф_i}{Ф_{max}}, \qquad (5.4.1)$$

где $Ф_i$ – стоимость i-го варианта исполнения,

$Ф_{max}$ – максимальная стоимость исполнения из всех вариантов.

Реализованный проект не будет являться прямым конкурентом openh264, так как openh264 более функционален и его разработка по времени

и стоимости будет выше. Допустим, что для разработки аналога потребуется на 50% больше ресурсов, тогда:

$$I_{ф}^{П}=\frac{Ф_{П}}{Ф_{max}}=\frac{Ф_{П}}{Ф_{П}}=1, \quad (5.4.2)$$

$$I_{ф}^{openh264}=\frac{Ф_{openh264}}{Ф_{max}}=\frac{Ф_{openh264}}{Ф_{П}}=\frac{(1+0{,}5)\cdot Ф_{П}}{Ф_{П}}=1{,}5, \quad (5.4.3)$$

5.4.2 Определение показателя ресурсоэффективности

Интегральный показатель ресурсоэффективности для i-го варианта исполнения рассчитывается как:

$$I_{ф}^{исп.i}=\sum_{j=1}^{n} a_j \cdot b_j, \quad (5.4.4)$$

где a_j – весовой коэффициент j-го параметра сравнения,

b_j – оценка j-го параметра сравнения по пятибалльной шкале,

n – число параметров сравнения.

В качестве параметров сравнения были взяты критерии конкурентоспособности из подраздела 5.1.1 с теми же весовыми коэффициентами. Результаты определения показателя ресурсоэффективности для разработки и аналоги приведены в таблице 5.14.

Таблица 5.14 – Показатель ресурсоэффективности для разработки и аналога

Параметр	Вес критерия	Объект исследования	
		Проект	о
Легкость работы	0,20	5	
Функциональность	0,20	3	
Кроссплатформенность	0,20	4	
Простота запуска	0,20	4	

Конкурентоспособность	0,1	3	
Послепродажное обслуживание	0,1	4	
Итого:	1	3,90	

5.4.3 Интегральный показатель эффективности

Интегральный показатель эффективности *i*-го варианта исполнения вычисляется как:

$$I_{исп.i} = \frac{I_{П}^{исп.i}}{I_{ф}^{исп.i}}. \quad (5.4.5)$$

Для сравнения *i*-го и *j*-го вариантов исполнения разработки используется сравнительная эффективность проекта:

$$Э_{ср} = \frac{I_{исп.i}}{I_{исп.j}}, \quad (5.4.6)$$

Взяв за *i*-ый вариант разработку, описанную в данной работе, а за *j*-ый – прямого конкурента, можно сделать вывод о целесообразности предлагаемого варианта: если Эср > 1, то предлагаемая разработка более целесообразна. В таблице 5.15 приведены все рассчитанные значения интегральных показателей и сравнительная эффективность вариантов исполнения.

Таблица 5.15 – Сравнение интегральных показателей

Показатель	Проект	openh264
Интегральный финансовый показатель разработки	1	1,50
Интегральный показатель ресурсоэффективности разработки	3,90	2,40
Интегральный показатель эффективности	3,90	1,60
Сравнительная эффективность вариантов исполнения	2,44	

Таким образом, значение сравнительной эффективности показывает ее большую целесообразность.

Анализируя экономическую целесообразность можно сказать, что на данный момент существует множество различных решений, позволяющих кодировать и декодировать видео. Однако большая часть является платными или представляют неудобный и низкоуровневый интерфейс для разработки ПО.

Проект не задумывался, как коммерческий, но при его коммерциализации можно добиться прибыли. Клиенты использующие данное ПО не будут вынуждены искать несколько различных решений под разные платформы.

Повысить экономическую эффективность возможно расширяя количество поддерживаемых платформ. Разработка под дополнительную платформу не будет сложна, однако, при увеличении количества поддерживаемых платформ предполагается, что количество клиентов, которые захотят использовать у себя данный продукт, а значит и потенциальная прибыль увеличится в разы.

5.5 Риски научно-исследовательского проекта

При разработке научно-исследовательского проекта следует понимать и учитывать возможные риски. Представленная таблица 5.16 содержит результаты анализа возможных рисков.

Таблица 5.16 – Реестр рисков

Риск	**Потенциальное воздействие**	**Вероятность наступления (1-5)**	**Влияние риска (1-5)**	**Уровень риска**	**Способы смягчения риска**	**Условия наступления**
Политический риск	Отключение GitHub в РФ	2	4	высокий	Переход на другие платформы	Негативное изменение в политической ситуации
Технический	Некорректная работа на	4	3	высокий	Тестирование на	Запуск многими

риск	некоторых устройствах				разных смартфонах	пользователями на разных устройствах
Кадровый риск	Отсутствие заинтересованных исполнителей проекта	2	4	средний	Повышение мотивации исполнителей проекта	Потеря интереса исполнителей к деятельности проекта

Из анализа реестров следует, что на первый риск обойти тяжело, так как не существует полных российских аналогов GitHub. Второй риск тяжело предотвратить, так как требуется большое количество устройств с ОС Android, но на выполнение научно-исследовательского проекта он мало влияет. Третий риск присутствует почти в каждом проекте.

5.6 Выводы по разделу

В ходе работы над разделом проведен предпроектный анализ, включающий анализ конкурентных решений и SWOT-анализ. Анализ конкурентных решений показал преимущество разрабатываемой библиотеки по сравнению с конкурентами. SWOT-анализ позволил определить потенциальные пути улучшения разработки: увеличение количества поддерживаемых платформ и увеличение количества поддерживаемых GPU.

Также в ходе инициации проекта составлен календарный план проекта, составлен перечень работ и рассчитана их продолжительность (таблица 5.10). График-план проекта был представлен в виде диаграммы Гантта (рисунок 5.2).

Сформирован бюджет разработки, включающий материальные затраты, затраты на амортизацию, основную заработную плату исполнителям, дополнительную заработную плату, отчисления во

внебюджетные фонды и накладные расходы (таблица 5.8). Общий бюджет разработки составил 372664,82 руб.

Проведено сравнение эффективности выполнения для данной разработки и ее прямого аналога на основании сравнения значений интегральных показателей эффективности. Предлагаемый в данной работе подход оказался более целесообразным.

ЗАДАНИЕ ДЛЯ РАЗДЕЛА

«СОЦИАЛЬНАЯ ОТВЕТСТВЕННОСТЬ»

Студенту:

Группа	ФИО
8ИМ11	Дмитриеву Василию Витальевичу

Школа	ИШИТР	Отделение школы (НОЦ)	ОИТ
Уровень образования	Магистратура	**Направление/специальность**	09.04.02 Информационные системы и технологии

Тема ВКР:

Модернизация кроссплатформенной библиотеки для обработки видео на базе FFmpeg	
Исходные данные к разделу «Социальная ответственность»:	
Введение - Характеристика объекта исследования (вещество, материал, прибор, алгоритм, методика) и области его применения. - Описание рабочей зоны (рабочего места) при разработке проектного решения/при эксплуатации	*Объект исследования:* библиотека кодирования и декодирования видео (программное обеспечение) *Область применения:* обработка видео *Рабочая зона*: офис *Размеры помещения:* $35\ м^2$ *Количество и наименование оборудования рабочей зоны:* персональные компьютеры (ПК) в количестве 6 штук *Рабочие процессы, связанные с объектом исследования, осуществляющиеся в рабочей зоне:* работа за ПК в сидячем положении
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. Правовые и организационные вопросы обеспечения безопасности: - специальные (характерные при эксплуатации объекта исследования, проектируемой рабочей зоны) правовые нормы трудового законодательства; - организационные мероприятия при компоновке рабочей зоны.	ГОСТ 12.2.032-78 Рабочее место при выполнении работ сидя; IC CSR-08260008000: 2011 «Социальная ответственность организации»; Федеральный закон от 30.03.1999 N 52-ФЗ (ред. от 04.11.2022) "О санитарно-эпидемиологическом благополучии населения" Трудовой кодекс Российской Федерации" от 30.12.2001 N 197-ФЗ (ред. от 19.12.2022, с изм. От 11.04.2023), статьи: 91; 111; 142; 212.
2. Производственная безопасность: - Анализ выявленных вредных и опасных факторов - Расчет уровня опасного или вредного производственного фактора	**Вредные факторы:** Повышенный уровень шума; Отсутствие или недостаток необходимого искусственного освещения; Наличие электромагнитных полей радиочастотного диапазона; Вредные производственные факторы, связанные с аномальными микроклиматическими параметрами воздушной среды на местонахождении работающего. **Опасные факторы:** Повышенное образование электростатических зарядов. Произведен расчет освещенности рабочей зоны.
3. Экологическая безопасность при разработке проектного решения:	Воздействие на литосферу путем утилизации люминесцентных ламп, компьютеров и другой оргтехники
4. Безопасность в чрезвычайных ситуациях при разработке проектного решения:	Возможные ЧС: Пожар, Обрушение здания Наиболее типичная ЧС: Пожар

Дата выдачи задания для раздела по линейному графику	

Задание выдал консультант:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ООД ШБИП ТПУ	Антоневич Ольга Алексеевна	к.б.н.		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8ИМ11	Дмитриев Василий Витальевич		

6 Социальная ответственность

Представление понятия «Социальная ответственность» сформулировано в международном стандарте (МС) IC CSR-08260008000: 2011 «Социальная ответственность организации» [20].

В соответствии с МС социальная ответственность – это ответственность организации за воздействие ее решений и деятельности на общество и окружающую среду через прозрачное и этичное поведение, которое:

- содействует устойчивому развитию, включая здоровье и благосостояние общества;
- учитывает ожидания заинтересованных сторон;
- соответствует применяемому законодательству и согласуется с международными нормами поведения (включая промышленную безопасность и условия труда, экологическую безопасность);
- интегрировано в деятельность всей организации и применяется во всех ее взаимоотношениях (включая промышленную безопасность и условия труда, экологическую безопасность).

Выполняемая работа заключается в разработке библиотеки кодирования и декодирования видео, таким образом, работу можно классифицировать как работу разработчика программного обеспечения.

В разделе будут рассмотрены опасные и вредные факторы, оказывающие влияние на производственную деятельность технологического персонала, работающего с персональным компьютером, рассмотрены воздействия разрабатываемой системы на окружающую среду, правовые и организационные вопросы, а также мероприятия в чрезвычайных ситуациях.

6.1 Правовые и организационные вопросы обеспечения безопасности

6.1.1 Специальные правовые нормы трудового законодательства

Правовое регулирование трудовых отношений между работодателем, работником и государством регулируется Трудовым кодексом Российской Федерации от 30.12.2001 N 197-ФЗ. В ТК РФ в соответствии с Конституцией РФ, признаются свобода труда, выбор и согласие на него, а также выбор профессии и деятельности [21].

В соответствии со ст. 91 ТК РФ, Нормальная продолжительность рабочего времени не может превышать 40 часов в неделю.

В соответствии со ст. 111 ТК РФ Всем работникам предоставляются выходные дни (еженедельный непрерывный отдых). При пятидневной рабочей неделе работникам предоставляются два выходных дня в неделю, при шестидневной рабочей неделе - один выходной день.

В соответствии со ст. 142 ТК РФ, в случае задержки выплаты заработной платы на срок более 15 дней работник имеет право, известив работодателя в письменной форме, приостановить работу на весь период до выплаты задержанной суммы, кроме ряда перечисленных случаев.

В соответствии со ст. 212 ТК РФ, работодатель обязан обеспечить безопасные условия труда, а также обязательное социальное страхование 64 работников от несчастных случаев на производстве и профессиональных заболеваний.

В ФЗ от 30.03.1999 N 52-ФЗ (ред. от 04.11.2022) указано, что условия работы с машинами, механизмами, установками, устройствами, аппаратами, которые являются источниками физических факторов воздействия на человека (шума, вибрации, ультразвуковых, инфразвуковых воздействий, теплового, ионизирующего, неионизирующего и иного излучения), не должны оказывать вредное воздействие на человека.

Для комфортного времяпрепровождения на территории рабочего места, организации необходимо соблюдать ряд правил для офисного помещения и

персональных компьютеров, изложенных в трудовом праве российской федерации.

6.1.2 Организационные мероприятия при компоновке рабочей зоны

Ниже будут перечислены предъявляемые требования к расположению и компоновке рабочего места.

Высота рабочей поверхности стола для взрослых пользователей должна регулироваться в пределах (680-800) мм, при отсутствии такой возможности высота рабочей поверхности стола должна составлять 725 мм [22].

Модульными размерами рабочей поверхности стола для ПК, на основании которых должны рассчитываться конструктивные размеры, следует считать: ширину 800, 1000, 1200 и 1400 мм, глубину 800 и 1000 мм при нерегулируемой его высоте, равной 725 мм [22].

Рабочий стол должен иметь пространство для ног высотой не менее 600 мм, шириной – не менее 500 мм, глубиной на уровне колен – не менее 450мм и на уровне вытянутых ног – не менее 650 мм [22].

Конструкция рабочего стула должна обеспечивать:

- ширину и глубину поверхности сиденья не менее 400 мм;
- поверхность сиденья с закругленным передним краем;
- регулировку высоты поверхности сиденья в пределах (400 550) мм и углам наклона вперед до 15 град, и назад до 5 град.;
- высоту опорной поверхности спинки (300±20) мм, ширину – не менее 380 мм и радиус кривизны горизонтальной плоскости – 400 мм;
- угол наклона спинки в вертикальной плоскости в пределах ±30 градусов; регулировку расстояния спинки от переднего края сиденья в пределах (260 400) мм;

- стационарные или съемные подлокотники длиной не менее 250мм и шириной – (50 70) мм;
- регулировку подлокотников по высоте над сиденьем в пределах (230±30) мм и внутреннего расстояния между подлокотниками в пределах (350 500) мм [22].

При выборе компьютеров для сотрудников важным является возможность монитора компьютера изменять положение в различных плоскостях (горизонтальные или вертикальные), с возможной устойчивой фиксацией в положении, которая удобна пользователю. Экран монитора должен содержать регулировку яркости и контрастности, что каждый работник мог установить нужный режим, которые будет соответствовать чувствительности глаз.

Освещенность на поверхности стола в зоне размещения рабочего документа должна быть 300-500 лк. Освещение не должно создавать бликов на поверхности экрана. Освещенность поверхности экрана не должна быть более 300 лк [22].

Рабочие столы следует размещать таким образом, чтобы видеодисплейные терминалы были ориентированы боковой стороной к световым проемам, чтобы естественный свет падал преимущественно слева [22].

Конструкция рабочего стула (кресла) должна обеспечивать поддержание рациональной рабочей позы при работе на ПЭВМ, позволять изменять позу с целью снижения статического напряжения мышц шейноплечевой области и спины для предупреждения развития утомления. Тип рабочего стула (кресла) следует выбирать с учетом роста пользователя, характера и продолжительности работы с ПЭВМ [22].

Контролируемыми гигиеническими параметрами персональных цифровых ЭВМ (в т.ч. портативных) являются: уровни электромагнитных полей (ЭМП), акустического шума, концентрация вредных веществ в воздухе, визуальные показатели ВДТ, мягкое рентгеновское излучение.

Помещение должно быть оборудовано системами вентиляции, кондиционирования и отопления.

Создание благоприятных условий труда и правильное эстетическое оформление рабочих мест на производстве имеет большое значение как для облегчения труда, так и для повышения его привлекательности, положительно влияющей на производительность труда.

6.2 Производственная безопасность

В данном пункте анализируются вредные и опасные факторы, которые могут возникать при разработке или эксплуатации проектируемого решения.

В процессе исследования с точки зрения возникающих вредных или опасных факторов можно выделить один основной этап: работа сидя за ЭВМ.

6.2.1 Анализ вредных и опасных факторов, которые может создать объект исследования

Вредные и опасные факторы регулирует ГОСТ 12.0.003-2015 [23].

Таблица 6.1 – Возможные опасные и вредные производственные факторы на рабочем месте при выполнении НИР

№	Факторы (ГОСТ 12.0.003-2015)	Нормативные документы
1	Повышенный уровень шума	ГОСТ 12.1.003-2014 Система стандартов безопасности труда. Шум. Общие требования безопасности.
2	Отсутствие или недостаток необходимого искусственного освещения	СНиП 23-05-95. Естественное и искусственное освещение.
3	Наличие электромагнитных полей радиочастотного диапазон	ГОСТ 12.1.009-2017 Система стандартов безопасности труда. Электробезопасность.
4	Вредные производственные факторы, связанные с	СанПиН 1.2.3685-21 Гигиенические нормативы и требования к обеспечению

	аномальными микроклиматическими параметрами воздушной среды	безопасности и (или) безвредности для человека факторов среды обитания
5	Повышенное образование электростатических зарядов	ГОСТ 12.1.030-81 Система стандартов безопасности труда. Электробезопасность. Защитное заземление, зануление.

6.2.2 Повышенный уровень шума

Превышение уровня шума является распространенным вредным фактором на рабочем месте. Его нарушение влечет за собой негативное воздействие не только на органы слуха, но и на весь организм человека в целом, через центральную нервную систему.

На рабочем месте за компьютером присутствует несколько различных источников шума: вращающиеся компоненты компьютера, а именно приводы жестких дисков, вентиляторы охлаждения центрального процессора, корпуса и блока питания; помехи из наушников; посторонние звуки в аудитории и из коридора.

Выполняемые работы оцениваются как научная деятельность, конструирование и проектирование, программирование, следовательно, согласно и с точки зрения возгораемого шума регулируются ГОСТ 12.1.003-2014 [24].

Снижению уровня шума способствует установка звукопоглощающих материалов (плиты, панели), подвесных акустических потолков, а также установка малошумного оборудования.

6.2.3 Отсутствие или недостаток необходимого искусственного освещения

Освещение рабочего места специалиста складывается из естественного и искусственного освещения. Естественное освещение достигается

установкой оконных проемов с коэффициентом естественного освещения (КЕО) не ниже 1,2% в зонах с устойчивым снежным покровом и не ниже 1,5% на остальной территории [25].

Нормируемые показатели естественного, искусственного и совмещенного освещения в соответствии с СанПиНом 1.2.3685-21 [26].

Для искусственного освещения помещений с персональными компьютерами следует применять светильники типа ЛПО36. Допускается применять светильники прямого света, преимущественно отраженного света типа ЛПО13, ЛПО5, ЛСО4, ЛПО34, ЛПО31 с люминесцентными лампами типа ЛБ.

В утреннее и вечернее время вводится общее искусственное освещение. Основными источниками искусственного освещения являются лампы белого света ЛБ-20. Для обеспечения нормируемых значений освещенности по СанПиН 1.2.3685-21 [26] в помещениях для работы за ПК следует проводить чистку стекол оконных рам и светильников не реже двух раз в год и проводить своевременную замену перегоревших ламп.

Расчет системы искусственного освещения проводится для прямоугольного помещения, размерами: длина А = 5 м, ширина В = 7 м, высота Н = 4 м, количество ламп N = 12 шт.

Вычисления будут, производится по методу светового потока, предназначенного для расчета освещенности общего равномерного освещения горизонтальных поверхностей. Согласно отраслевым нормам освещенности уровень рабочей поверхности над полом составляет 0,8 м и установлена минимальная норма освещенности Е = 300 Лк.

Световой поток лампы накаливания или группы люминесцентных ламп светильника определяется по формуле:

$$\Phi = \frac{E_н \cdot S \cdot K_з \cdot Z \cdot 100}{n \cdot \eta}, \qquad (6.2.1)$$

где $E_н$ – нормируемая минимальная освещенность по СНиП 23-05-95, (Лк);

S – площадь освещаемого помещения, $м^2$;

$K_з$ – коэффициент запаса, учитывающий загрязнение источника света, стен, прочих отражающих поверхностей, наличие в атмосфере цеха дыма, пыли;

Z – коэффициент неравномерности освещения. Для люминесцентных ламп при расчётах берётся равным $Z=1{,}1$;

n – число светильников;

η – коэффициент использования светового потока, %;

$Ф$ – световой поток, излучаемый светильником.

Коэффициент использования светового потока показывает, какая часть светового потока ламп попадает на рабочую поверхность. Он зависит от индекса помещения i, типа светильника, высоты светильников над рабочей поверхностью h и коэффициентов отражения стен ($\rho_{ст}$) и потолка ($\rho_п$). Индекс помещения определяется по формуле (6.2.2).

$$i=\frac{S}{h\cdot(A+B)}. \qquad (6.2.2)$$

Произведем расчет:

$$h=H-0{,}8=4-0{,}8=3{,}2. \qquad (6.2.3)$$

где h – расчетная высота подвеса светильников над рабочей поверхностью.

Экономичность осветительной установки зависит от отношения, представленного в формуле:

$$l=\frac{L}{h}. \qquad (6.2.4)$$

где L – расстояние между рядами светильников, м.

Рекомендуется размещать люминесцентные лампы параллельными рядами, принимая $l=1{,}4$, отсюда расстояние между рядами светильников:

$$L=l\cdot h=1{,}4\cdot 3{,}2=4{,}48 \text{ м}.$$

Два ряда светильников будут расположены вдоль длинной стены помещения. Расстояние между двумя рядами светильников и стенами вычисляется по формуле:

$$Л = \frac{B - L}{4} = \frac{7 - 4{,}48}{4} = 0{,}63\,м. \quad (6.2.5)$$

Определим индекс помещения вычисляя по формуле (6.6) получаем:

$$i = \frac{35}{3{,}2 \cdot 12} = 0{,}91. \quad (6.2.6)$$

Найдем коэффициенты отражения поверхностей стен, пола и потолка. Так как поверхность стен недавно побелена и окрашена в серый цвет, а помещение с окнами без штор, то коэффициент отражения поверхности стен $P_{ст} = 50\%$. Поверхность потолка тоже светлая, поэтому коэффициент отражения поверхности потолка $P_п = 30\%$. Учитывая коэффициенты отражения поверхностей стен, потолка и индекс помещения i, определяем значение коэффициента $\eta = 41\%$.

Подставив все значения в формулу (6.2.1), по которой рассчитывается световой поток одного источника света, получаем:

$$Ф = \frac{300 \cdot 35 \cdot 1{,}5 \cdot 1{,}1}{12 \cdot 0{,}41} = 3521\,Лм.$$

По полученному световому потоку подбираем лампу, наиболее подходящей является лампа LUNA250 со световым потоком 3520 Лм (F).

Выразим E:

$$E = \frac{F \cdot N \cdot \eta}{k} = \frac{3520 \cdot 12 \cdot 0{,}41}{1{,}5 \cdot 35 \cdot 1{,}1} = 299{,}5\,Лм. \quad (6.2.7)$$

Как видно из расчета, минимальная освещенность в пределах нормы. Для того чтобы доказать, что использование люминесцентной лампы LUNA250 является наиболее рациональным, рассчитаем необходимое количество светильников по формуле:

$$N = \frac{E \cdot k \cdot S \cdot Z}{n \cdot \eta \cdot F}, \quad (6.2.8)$$

где E – норма освещенности $E = 300\,Лк$;

k – коэффициент запаса учитывающий старение ламп и загрязнение светильников, $k=1{,}5$;

S – площадь помещения;

Z – коэффициент неравномерности освещения, $Z=1{,}1$;

n – число рядов светильников, $n=4$;

η – коэффициент использования светового потока, $\eta=0{,}41$;

F – световой поток, излучаемый светильником, $F=3520$ для LUNA250.

Подставим численные значения в формулу (6.2.8), получим количество светильников в одном ряду:

$$N=\frac{300\cdot 1{,}5\cdot 35\cdot 1{,}1}{4\cdot 0{,}41\cdot 3520}\approx 3\,шт.$$

Длина одного светильника равна 0,5 м, в одном светильнике 4 лампы LUNA250.

На рисунке 6.1 представлена аудитория, в которой проводились работы с расположением ламп.

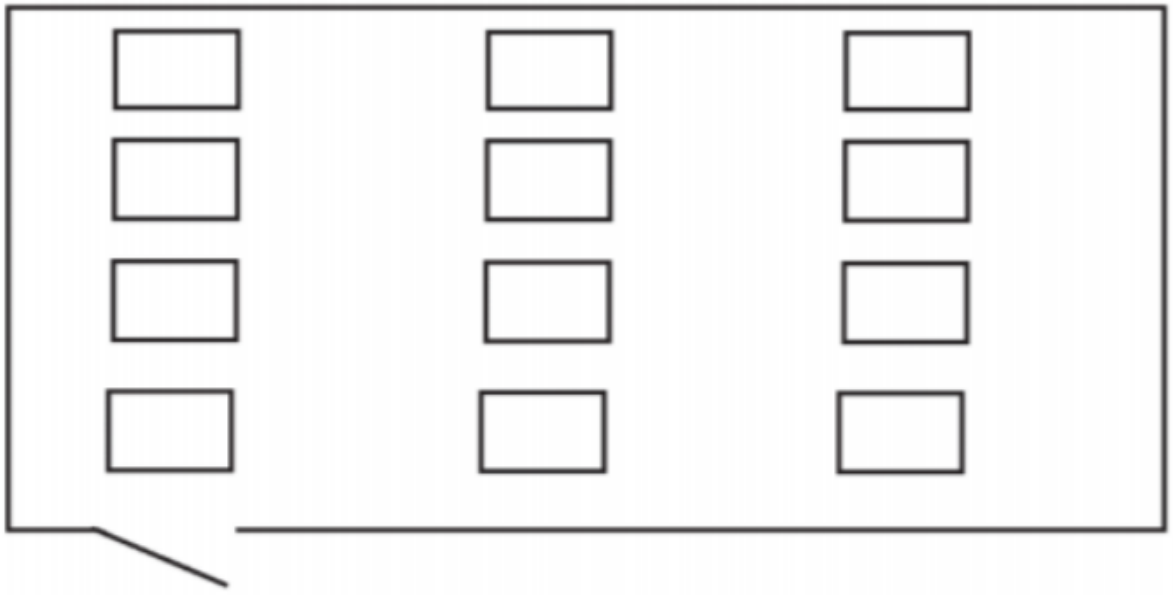


Рисунок 6.1 – Схема расположения ламп в аудитории КЦ НИ ТПУ

Так как в рассматриваемом помещении количество ламп 12 (шт), по три светильника в четырех рядах, следовательно, нормы безопасности по искусственному освещению в данном случае соблюдены.

6.2.4 Наличие электромагнитных полей радиочастотного диапазона

Электромагнитным излучением называется излучение, прямо или косвенно вызывающее ионизацию среды. Контакт с электромагнитными излучениями представляет серьезную опасность для человека, по сравнению с другими вредными производственными факторами (повышенное зрительное напряжение, психологическая перегрузка, сохранение длительное время неизменной рабочей позы).

Когда все устройства персонального компьютера включены, в районе рабочего места программиста, формируется сложное по структуре электромагнитное поле. Реальную угрозу для пользователя компьютера представляют электромагнитные поля. Влияние их на организм человека не обходится без последствий. Исследования показали, что в организме человека под влиянием электромагнитного излучения монитора происходят значительные изменения гормонального состояния, специфические изменения биотоков головного мозга, изменение обмена веществ. Пыль, притягиваемая электростатическим полем монитора, иногда становится причиной дерматитов лица, обострения астматических симптомов, раздражения слизистых оболочек.

Для снижения воздействия электромагнитного излучения следует применять мониторы с пониженным уровнем излучения, также устанавливать защитные экраны, придерживаться регламентированного режима труда и отдыха, а также проводить регулярную гигиеническую уборку помещения [27].

Для оценки соблюдения уровней необходим производственный контроль (измерения). В случае превышения уровней необходимы организационно-технические мероприятия (защита временем, расстоянием, экранирование источника, либо рабочей зоны, замена оборудования, использование СИЗ).

6.2.5 Вредные производственные факторы, связанные с аномальными микроклиматическими параметрами воздушной среды

Значимым физическим фактором является микроклимат рабочей зоны (температура, влажность и скорость движения воздуха). Температура, относительная влажность и скорость движения воздуха влияют на теплообмен и необходимо учитывать их комплексное воздействие. Нарушение теплообмена вызывает тепловую гипертермию, или перегрев.

Оптимальные нормы температуры, относительной влажности и скорости движения воздуха производственных помещений для работ, производимых сидя и не требующих систематического физического напряжения (категория Iа), согласно с СанПиН 1.2.3685-21 [26].

В рабочем помещении для выполнения данного научно исследовательского проекта температурные замеры в холодный период года– февраль 2023 – колебались от 19 до 22 градусов, температура в теплое время года – апрель 2023 – от 22 до 26, что соответствует допустимым значениям.

Для обеспечения установленных норм микроклиматических параметров и чистоты воздуха на рабочих местах и в помещениях применяют вентиляцию.

В холодное время года предусматривается система отопления. Для отопления помещений используются водяные системы центрального отопления. Радиаторы должны устанавливаться в нишах, прикрытых деревянными или металлическими решетками. Применение таких решеток способствует также повышению электробезопасности в помещениях.

6.2.6 Повышенное образование электростатических зарядов

Среди распространенных опасностей в рабочей зоне находится и поражение электрическим током. Опасность поражения определяется

величиной тока проходящего через тело человека I или напряжением прикосновения U. Напряжение считается безопасным при напряжении прикосновения $U < 42$ В.

При получении человеком разряда электрического тока могут быть получены электротравмы, электрические удары и даже летальный исход (согласно ГОСТ 12.1.009-2017 [28]). Для защиты от поражения электрическим током следует выполнить следующие пункты:

- обеспечить недоступность токоведущих частей от случайных прикосновений;
- соблюдать эстетичность производства;
- устранить опасность поражения при появлении напряжения на разных частях.

Токи статического электричества, наведенные в процессе работы компьютера на корпусах монитора, системного блока и клавиатуры, могут приводить к разрядам при прикосновении к этим элементам. Такие разряды опасности для человека не представляют, но могут привести к выходу из строя оборудования.

На рабочем месте пользователя размещены дисплей, клавиатура и системный блок. Перед началом работы следует убедиться в отсутствии свешивающихся со стола или висящих под столом проводов электропитания, в целостности вилки и провода электропитания, в отсутствии видимых повреждений аппаратуры и рабочей мебели, в отсутствии повреждений и наличии заземления приэкранного фильтра.

Методы защиты от воздействия статического электричества [29]:

- влажная уборка, чтобы уменьшить количество пылинок в воздухе и на предметах офиса;
- использование увлажнителей воздуха;
- защитное заземление;
- применение средств индивидуальной защиты, таких как антистатические спреи и браслеты.

6.3 Экологическая безопасность

Рассмотрим загрязнения литосферы в результате исследовательской деятельности бытовым мусором, на примере люминесцентных ламп. Их эксплуатация требует осторожности и четкого выполнения инструкции по обращению с данным отходом (код отхода 35330100 13 01 1, класс опасности – 1 [30]). В данной лампе содержится опасное вещество ртуть в газообразном состоянии. При неправильной утилизации, лампа может разбиться и пары ртути могут попасть в окружающую среду. Вдыхание паров ртути может привести к тяжелому повреждению здоровья.

При перегорании ртутьсодержащей лампы её замену осуществляет лицо, ответственное за сбор и хранение ламп (обученное по электробезопасности и правилам обращения с отходом). Отработанные люминесцентные лампы сдаются только на полигон токсичных отходов для захоронения. Запрещается сваливать отработанные люминесцентные лампы с мусором [30].

Бытовой мусор помещений организаций несортированный, образованный в результате деятельности работников предприятия (код отхода 91200400 01 00 4). Агрегатное состояние отхода твердое; основные компоненты: бумага и древесина, металлы, пластмассы и др [30]. Для сбора мусора рабочее место оснащается урной. При заполнении урны, мусор выносится в контейнер бытовых отходов. Предприятие заключает договор с коммунальным хозяйством по вывозу и размещению мусора на организованных свалках.

Согласно ФЗ от 10.01.2002 № 7-ФЗ «Об охране окружающей среды» [31] аудиторию, в которой проводятся работы стоит рассматривать, как объект относящийся к IV категории опасности.

6.4 Безопасность в чрезвычайных ситуациях

6.4.1 Анализ вероятных ЧС, которые может инициировать объект исследований

Перечень возможных ЧС, которые могут возникнуть на рабочем месте, достаточно широк. Ограничиваясь местоположением объекта и условиями его эксплуатации, его можно представить следующим ориентировочным вариантом:

- обрушение здания;
- пожар на объекте;

В этом разделе наиболее актуальным будет рассмотрение такого вида ЧС как пожар, определение категории помещения, где ведется научно исследовательская работа, на пожаровзрывобезопасность (аудитория КЦ НИ ТПУ) и регламентирование мер противопожарной безопасности.

Рабочее место разработчика ПО должно соответствовать требованиям ФЗ Технический регламент по ПБ и норм пожарной безопасности (НПБ 105-03) и удовлетворять требованиям по предотвращению и тушению пожара по ГОСТ 12.1.004-91 [32] и СНиП 21-01-97 [33]. По пожарной, взрывной, взрывопожарной опасности помещение относится к категории Д, т.е. к помещению, в котором находятся негорючие вещества и материалы в холодном состоянии. Основным поражающим фактором пожара для помещений данной категории является наличие открытого огня и отравление ядовитыми продуктами сгорания оборудования.

Пожар в помещении, где работает рассматриваемый тип сотрудника (программист), может возникнуть вследствие причин неэлектрического и электрического характера.

К причинам неэлектрического характера относятся халатное и неосторожное обращение с огнем (курение, оставление без присмотра нагревательных приборов).

К причинам электрического характера относятся:

- короткое замыкание;
- перегрузка проводов;
- большое переходное сопротивление;
- искрение;
- статическое электричество.

Режим короткого замыкания – появление в результате резкого возрастания силы тока, электрических искр, частиц расплавленного металла, электрической дуги, открытого огня, воспламеняющейся изоляции.

Причины возникновения короткого замыкания:

- ошибки при проектировании;
- старение изоляции;
- увлажнение изоляции;
- механические перегрузки.

6.4.2 Обоснование мероприятий по предотвращению ЧС и разработка порядка действия в случае возникновения ЧС

Пожарная безопасность объекта должна обеспечиваться системами предотвращения пожара и противопожарной защиты, в том числе организационно-техническими мероприятиями.

Пожарная защита должна обеспечиваться применением средств пожаротушения, а также применением автоматических установок пожарной сигнализации.

Должны быть приняты следующие меры противопожарной безопасности:

- обеспечение эффективного удаления дыма, т.к. в помещениях, имеющих оргтехнику, содержится большое количество пластиковых веществ, выделяющих при горении летучие ядовитые вещества и едкий дым;

- обеспечение правильных путей эвакуации;
- наличие огнетушителей и пожарной сигнализации;
- соблюдение всех противопожарных требований к системам отопления и кондиционирования воздуха.

Для тушения пожаров на участке производства необходимо применять углекислотные (ОУ-5 или ОУ-10) и порошковые огнетушители (например, типа ОП-10).

Помещение КЦ НИИ ТПУ оборудовано пожарными извещателями, которые позволяют оповестить дежурный персонал о пожаре. В качестве пожарных извещателей в помещении устанавливаются дымовые фотоэлектрические извещатели типа ИДФ-1 или ДИП-1.

Выведение людей из зоны пожара должно производиться по плану эвакуации. План эвакуации представляет собой заранее разработанный план (схему), в которой указаны пути эвакуации, эвакуационные и аварийные выходы, установлены правила поведения людей, порядок и последовательность действий в условиях чрезвычайной ситуации по п. 3.14 ГОСТ Р 12.2.143-2009 [34].

Согласно Правилам пожарной безопасности, в Российской Федерации ППБ 01-2003 (п. 16) в зданиях и сооружениях (кроме жилых домов) при единовременном нахождении на этаже более 10 человек должны быть разработаны и на видных местах вывешены планы (схемы) эвакуации людей в случае пожара. План эвакуации людей при пожаре из помещения, где расположена аудитория КЦ НИ ТПУ, представлен на рисунке 6.2.

Рисунок 6.2 – План эвакуации при пожаре

6.5 Выводы по разделу

Все производственные факторы на изучаемом рабочем месте соответствуют нормам, описанным в данном разделе.

Категория помещения по электробезопасности согласно ПУЭ соответствует первому классу – «помещения без повышенной опасности» [29].

Согласно правилам по охране труда при эксплуатации электроустановок персонал должен обладать I группой допуска по электробезопасности. Присвоение группы I по электробезопасности производится путем проведения инструктажа, который должен завершаться проверкой знаний в форме устного опроса и (при необходимости) проверкой приобретенных навыков безопасных способов работы или оказания первой помощи при поражении электрическим током [35].

Категория тяжести труда в лаборатории по СанПиН 1.2.3685-21 "Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания" относится к категории Iб (работы, производимые сидя, стоя или связанные с ходьбой и сопровождающиеся физическим напряжением) [26].

Рабочее место разработчика ПО должно соответствовать требованиям ФЗ Технический регламент по ПБ и норм пожарной безопасности (НПБ 105-03) и удовлетворять требованиям по предотвращению и тушению пожара по ГОСТ 12.1.004-91 [32] и СНиП 21-01-97 [33]. По пожарной, взрывной, взрывопожарной опасности помещение относится к категории Д, т.е. к помещению, в котором находятся негорючие вещества и материалы в холодном состоянии. Основным поражающим фактором пожара для помещений данной категории является наличие открытого огня и отравление ядовитыми продуктами сгорания оборудования.

Рассмотренный объект, оказывающий незначительное негативное воздействие на окружающую среду, относится к объектам IV категории [36].

Заключение

Основные результаты, полученные в процессе выполнения работы:

1. Выполнен анализ существующих видеокодеков и методов кодирования, необходимых для сферы обработки видео. Признано, что создание кроссплатформенной библиотеки для кодирования и декодирования видео актуально.

2. Спроектированы и описаны основные компоненты библиотеки, включающие: кодер (программный и аппаратный); декодер (программный и аппаратный); фильтр кадрирования; фильтр масштабирования. Построено представление о необходимых тестирующих сущностях.

3. Реализована библиотека кодирования и декодирования в виде файла .lib на языке C++ для ОС Windows, и .aar на языке Java для ОС Android. Создан скрипт на языке Python для сборки библиотеки.

4. Проведено тестирование библиотеки кодирования и декодирования видео. На ОС Windows создано приложение, в котором на окне Windows отображалось кодированное и декодированное видео с примененными фильтрами. На окне и в консоли отображалось логирование основной информации. Для ОС Android создано два приложения, выводящих на экран телефона кодированное и декодированное видео с камеры. В первом приложении использовался программный кодер, во втором аппаратный.

Создано 9 unit-тестов проверяющих работу основных компонентов библиотеки.

Произведено сравнение производительности кодирования и декодирования видео с библиотекой x264. Выяснено, что различие в производительности не превышает 2,5 % и объясняются сторонними процессами в системе.

Рекомендации:

– созданная библиотека может быть использована, как программное решение в качестве замены существующих библиотек, предоставляющий пользователю низкоуровневый интерфейс;

– созданная библиотека может быть развита, как open-source решение для разработчиков по всему миру.

В рамках данной работы были разработаны разделы «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение», «Социальная ответственность», а также раздел на иностранном языке (английский) – «Overview of Different Types of Video Codecs», размещенный в Приложении А.

Список использованных источников

1. Mohammed Ghanbari. Video Coding: An Introduction to Standard Codecs. Institution of Electrical Engineers (1999). pp. 107–120. ISBN 9780852967621
2. Сэломон Д. Сжатие данных, изображений и звука. – М.: Техносфера, 2004. – 368 с
3. Iain Richardson. Video Codec Design: Developing Image and Video Compression Systems. Wiley; 1st edition (May 13, 2002). pp. 35–54. ISBN 978-0471485537
4. Advanced video coding for generic audiovisual services // Recommendation ITU-T H.264. – 2011. – P. 686.
5. Bross B. et al. High-Efficiency Video Coding (HEVC) text specification draft 9 // JCT-VC: 11th meeting. – Shanghai, CN, 10–19 October, 2012. – 390 p. URL: http://phenix.int-evry.fr/jct/doc_end_user/documents/11_Shanghai/wg11/JTVC-K1003-v13.zip (дата обращения: 20.04.2023).
6. G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wie-gand, “Overview of the high efficiency video cod-ing (hevc) standard,” Transactions on circuits and systems for video technology, vol. 22, no. 12, pp.1649–1668, 2012.
7. Living in a Multi-Codec World – Future Codecs revisited. Paul MacDougall. July 2015. URL: https://bitmovin.com/multi-codec-world-2020/ (дата обращения: 14.11.2022)
8. Generic coding of moving pictures and associated audio information: Video // MPEG-2 ISO/IEC 13818–2. – 1998. – P. 202.
9. Mohammed Ghanbari. Standard Codecs: Image Compression to Advanced Video Coding (Telecommunications). The Institution of Engineering and Technology (August 31, 2003). pp. 55–72. ISBN 978-0852967102

10. Richardson I. White Paper: H.264/AVC Intra Prediction // Vcodex. – 2011. – P. 1–7. URL: http://www.vcodex.com/files/H264_intrapred_wp.pdf (дата обращения: 12.03.2023).
11. T. Wiegand, G. J. Sullivan, G. Bjontegaard, andA. Luthra, “Overview of the h. 264/avc video cod-ing standard,” Transactions on circuits and systems for video technology, vol. 13, pp. 560–576, 2003.
12. The H.264 Advanced Video Compression Standard, Second Edition. Wiley (August, 2010). pp. 29–33. ISBN 9780470516928
13. J. Kufa and T. Kratochvil, “Software and hardware hevc encoding,” in Int. Conf. on Systems, Signals and Image Processing, 2017, pp. 1–5.
14. Fisher Y. Fractal Image Compression – Theory and Application. – N.Y.: Springer"Verlag, 1994. – 341 p.
15. S. Zhang, Z. Fan, L. Chen, C. Qian, and X. Gao,“Realizing hardware accelerated avs video playback on the godson platform,” in Int. Conf. on Electron-ics, Circuits, and Systems, 2013, pp. 177–180.
16. Hardware and software video encoding comparison. Ramil Safin, Emilia Garipova, Roman Lavrenov. Proceedings of the SICE Annual Conference 2020 (September 23-26 2020). pp. 925-927. URL: https://www.researchgate.net/publication/344375475_Hardware_and_software_video_encoding_comparison. (дата обращения: 16.11.2022)
17. Intel QuickSync Video and FFmpeg. Intel Corporation. January 2016. pp. 5-6. URL: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cloud-computing-quicksync-video-ffmpeg-white-paper.pdf. (дата обращения: 21.11.2022)
18. NVIDIA VIDEO CODEC SDK – ENCODER. Application Note. NVIDIA Corporation. URL: https://docs.nvidia.com/video-technologies/video-codec-sdk/pdf/NVENC_Application_Note.pdf. (дата обращения: 02.12.2022)
19. Интерфакс. Численность и оборот ИТ-компаний в РФ растут, рентабельность активов снижается. 5 октября 2022. – Текст:

электронный URL: https://www.interfax.ru/business/866443. (дата обращения: 15.03.2022)
20. IC CSR-08260008000:2011 Социальная ответственность организации
21. Трудовой кодекс Российской Федерации" от 30.12.2001 N 197-ФЗ (ред. от 19.12.2022, с изм. от 11.04.2023) (с изм. и доп., вступ. в силу с 01.03.2023)
22. ГОСТ 12.2.032-78 Рабочее место при выполнении работ сидя
23. ГОСТ 12.0.003-2015 ССБТ. Опасные и вредные производственные факторы. Классификация
24. ГОСТ 12.1.003-2014 Система стандартов безопасности труда. Шум. Общие требования безопасности.
25. СНиП 23-05-95. Естественное и искусственное освещение.
26. СанПиН 1.2.3685-21 "Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания"
27. Безопасность жизнедеятельности. Учебник. Под ред. Э.А. Арустамова / 10-е изд., перераб. и доп. – М.: Изд-во «Дашков и К°», 2006. – 476 с.
28. ГОСТ 12.1.009-2017 Система стандартов безопасности труда. Электробезопасность.
29. ГОСТ 12.1.030-81 Система стандартов безопасности труда. Электробезопасность. Защитное заземление, зануление.
30. Федеральный классификационный каталог отходов [Электронный ресурс]. – 2013. – Режим доступа: http://www.ecoguild.ru/faq/fedwastecatalog. html, свободный. - (Дата обращения: 10.05.2023)
31. Федерального закона от 10.01.2002 № 7-ФЗ «Об охране окружающей среды»
32. ГОСТ 12.1.004-91 ССБТ. Пожарная безопасность. Общие требования.
33. СНиП 21-01-97. Пожарная безопасность зданий и сооружений.

34. ГОСТ Р 12.2.143-2009 Система стандартов безопасности труда. Системы фотолюминесцентные эвакуационные. Требования и методы контроля
35. Приложение к приказу Министерства труда и социальной защиты Российской Федерации от 15 декабря 2020 года N 903н “Правила по охране труда при эксплуатации электроустановок”
36. Федеральный закон от 10.01.2002 № 7-ФЗ «Об охране окружающей среды»

Приложение А

(справочное)

Overview of Different Types of Video Codecs

Студент

Группа	ФИО	Подпись	Дата
8ИМ11	Дмитриев Василий Витальевич		

Руководитель ВКР

Должность	ФИО	Ученая степень, звание	Подпись	Дата
доцент	Чердынцев Евгений Сергеевич	к.т.н.		

Консультант-лингвист отделения иностранных языков ШБИП

Должность	ФИО	Ученая степень, звание	Подпись	Дата
доцент	Уткина Анна Николаевна	к.филос.н.		

7 Video coding and decoding

Coding and decoding, also known as video compression and decompression, are techniques used to reduce the amount of data needed to represent a video signal.

Video coding involves compressing a video signal by removing redundant or irrelevant information, so that it takes up less storage space or bandwidth when transmitted over a network. There are various video coding standards available, such as H.264, H.265, VP9, AV1, etc., each of which uses different techniques to compress video.

Video decoding, on the other hand, involves decompressing a compressed video signal back into its original form so that it can be viewed or further processed. Video decoding requires a decoder, which is a software or hardware component that can understand the compression format used to encode the video and can reconstruct the original video signal.

Both coding and decoding are essential components of video processing and are used in a variety of applications, including video streaming, video conferencing, video surveillance, and digital television broadcasting.

Video codecs are essential tools that enable digital video files to be compressed and encoded in a way that makes them smaller and more manageable. In this way, video codecs help make video files easier to store, transmit, and playback on a range of devices.

In general, video codecs operate by using mathematical algorithms to compress and decompress video data. This involves removing redundant or unnecessary data from video files, while retaining the key visual and auditory information that is necessary for playback.

There are many different video codecs available, each with its own strengths and weaknesses. Some of the most popular codecs include:

- H.264/AVC - This codec is widely used for online video streaming and digital TV broadcasts. It offers high compression rates and excellent image quality, making it a popular choice for many video application;
- HEVC/H.265 - This is a newer codec that is even more efficient than H.264, offering even higher compression rates while maintaining good image quality. It is particularly well-suited for 4K and 8K video, which require large amounts of data to be compressed;
- VP9 - Developed by Google, this codec is designed for online video playback and is used by platforms like YouTube. It offers good compression rates and high image quality, and is particularly well-suited for streaming video over slower internet connections;
- AV1 - This is a newer, open-source codec that is gaining popularity due to its excellent compression rates and high image quality. It is particularly well-suited for streaming high-quality video over slower internet connections.

In figure 1 you can see the prevalence of codecs.

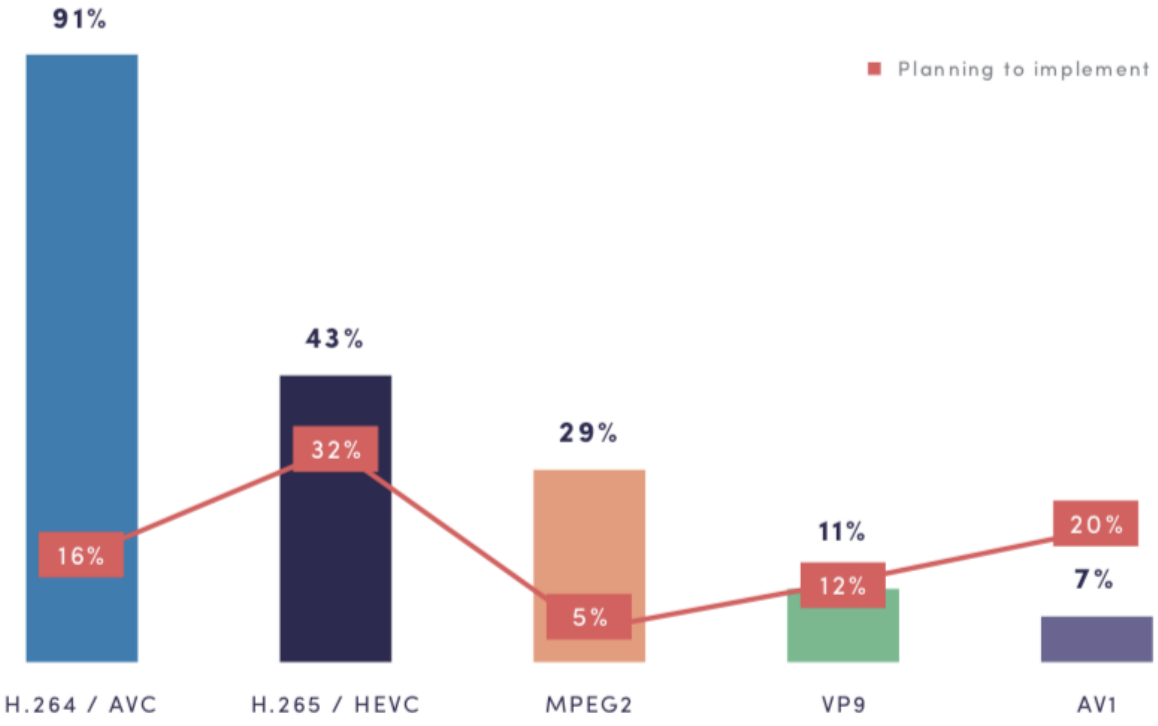


Figure 1 - Video codec usage (red line - usage change by 2020)

In addition to these codecs, there are many others available, including proprietary codecs developed by companies like Apple, Microsoft, and Sony. When choosing a video codec, it is important to consider factors like image quality, compression rate, compatibility with different devices and platforms, and licensing costs.

Video codecs play a critical role in modern video production and distribution. Without them, digital video files would be much larger and more difficult to work with, making it much harder to produce and distribute high-quality video content. Whether you are a professional video producer, a casual video enthusiast, or a regular user of online video platforms, understanding video codecs and how they work is essential for making the most of today's digital video landscape.

Another important aspect of video codecs is their ability to handle different types of video content. For example, some codecs may be better suited for handling fast-moving action sequences, while others may be better for handling subtle changes in lighting or color. This means that choosing the right codec for a particular video project can make a big difference in the final image quality and playback experience.

In addition to compression and image quality, video codecs can also affect things like file size, bandwidth usage, and playback performance. For example, codecs that use more aggressive compression algorithms may result in smaller file sizes, but may also require more processing power to decode and playback. Similarly, codecs that use more bandwidth may be able to deliver higher-quality video, but may also require faster internet connections to avoid buffering or other playback issues.

One important consideration when working with video codecs is compatibility. Not all devices and platforms support all codecs, so it's important to choose a codec that will work well across a range of devices and platforms. This

may involve testing different codecs on different devices or using tools that can convert video files between different formats.

Overall, video codecs are essential tools for modern video production and distribution. They allow us to compress and encode video files in a way that makes them more manageable, while still maintaining high image quality and playback performance. Whether you are producing professional video content or just watching videos online, understanding the role of video codecs can help you make the most of this exciting and dynamic medium.

7.1 H.264 Codec

H.264, also known as AVC (Advanced Video Coding), is a widely used video codec that was developed by the Joint Video Team (JVT), a collaboration between the International Telecommunication Union (ITU) and the ISO/IEC Moving Picture Experts Group (MPEG). H.264 is known for its high compression rates, which allow for high-quality video to be streamed or stored in smaller file sizes.

The H.264 codec works by using a combination of spatial and temporal compression techniques. Spatial compression involves removing redundant data within a single video frame, while temporal compression involves removing redundant data between frames. This combination of techniques allows H.264 to achieve high compression rates while still maintaining good image quality.

At a high level, the H.264 encoding process involves the following steps:

1. Dividing the video into macroblocks: The video is divided into rectangular blocks of 16x16 pixels, called macroblocks;
2. Performing motion estimation: The encoder analyzes each macroblock and determines how much it has moved from the previous frame. This information is used to reduce redundancy between frames;

3. Performing intra-frame prediction: Within each macroblock, the encoder uses neighboring pixels to predict the values of other pixels. This reduces redundancy within a single frame;
4. Encoding the residuals: The difference between the predicted and actual pixel values (known as the residuals) are encoded using variable-length coding (VLC) or arithmetic coding;
5. Performing entropy coding: The encoded residuals are further compressed using entropy coding techniques such as Huffman coding;
6. Adding metadata: The encoded video stream is then supplemented with metadata that describes the video's format, resolution, bit rate, and other parameters.

On the decoding side, the process is essentially the reverse of the encoding process. The decoder uses the metadata to determine the video format and other parameters, and then performs entropy decoding, residual decoding, and inverse prediction to reconstruct the original video frames.

H.264 is widely supported by many hardware and software platforms, which means it can be played back on a variety of devices, including smartphones, tablets, computers, and gaming consoles. Its popularity has also led to the development of many H.264-based video formats, such as MP4, which are widely used for online video distribution.

One of the key advantages of H.264 is its ability to support a range of different video resolutions and bitrates, which makes it suitable for both high-quality video content as well as lower-quality video content for mobile devices or slower internet connections. It also supports a range of different video profiles, including baseline, main, and high profiles, which can be used to balance image quality and file size depending on the specific use case.

However, one potential downside of H.264 is that it can be computationally intensive to encode and decode, which may make it less suitable for real-time video applications or devices with limited processing power. This has led to the

development of newer codecs, such as H.265 (also known as HEVC), which offer similar compression rates as H.264 but with lower computational requirements.

Overall, H.264 is a highly efficient video codec that has become the de facto standard for many applications, including online video streaming, digital TV broadcasts, and Blu-ray discs. Its combination of spatial and temporal compression techniques allows it to achieve high compression rates while still maintaining good image quality, making it a popular choice for many video applications.

7.2 Hardware video codecs

A hardware video codec is a dedicated hardware component that is designed to perform video compression and decompression tasks. It is optimized to handle the processing requirements of video data, and is designed to deliver high-quality video content with fast and efficient processing. The hardware codec may be integrated into various devices such as smartphones, tablets, cameras, and set-top boxes, and is used to encode and decode video in real-time. The use of hardware video codecs has become increasingly popular in recent years due to the increasing demand for high-quality video content and the need for efficient processing of video data. They are well-suited for applications such as video conferencing, live streaming, and gaming, as they can provide real-time video processing and fast data transfer rates.

Hardware video codecs are generally faster than software video codecs because they are optimized for video processing tasks and use dedicated hardware components to perform the encoding and decoding of video data. Here are some reasons why hardware codecs are faster:

1. Dedicated hardware components: Hardware codecs use dedicated hardware components such as specialized circuits, processing units, and memory to perform video encoding and decoding tasks. These components are optimized for video processing and can perform these tasks much faster than general-purpose processors used in software codecs;

2. Parallel processing: Hardware codecs can perform multiple processing tasks in parallel, which increases the overall processing speed. This is because they have multiple processing units that work together to encode or decode video data simultaneously;
3. Reduced software overhead: Hardware codecs offload the processing tasks from the CPU and other general-purpose processors, which reduces the software overhead and frees up processing resources for other tasks. This leads to faster video processing and overall system performance;
4. Energy efficiency: Hardware codecs are often designed to be more power-efficient than software codecs, which reduces the amount of energy required to perform video processing tasks. This makes them well-suited for mobile devices and other low-power applications.

Hardware video codecs are dedicated hardware components that are designed to perform video compression and decompression tasks. They are typically integrated into devices such as smartphones, tablets, cameras, and set-top boxes, and are used to encode and decode video in real-time.

Hardware video codecs are designed to be highly optimized for video processing tasks, which makes them much faster and more power-efficient than software codecs running on general-purpose processors. They are often designed to support specific video formats and features, such as H.264 or HEVC, and may include additional features such as image stabilization or noise reduction.

In Figure 2 displayed a comparison in decoding between software and hardware decoding.

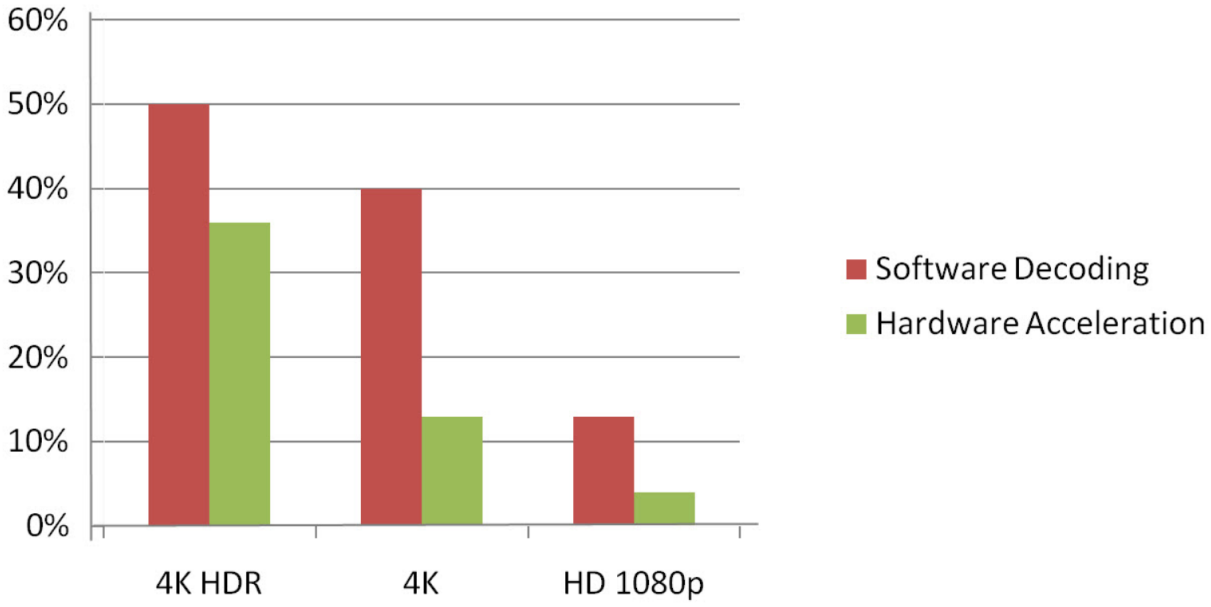


Figure 2 - Comparison between software and hardware decoding

As can be seen from the figure above, hardware acceleration for decoding in some cases exceeds software decoding by several times.

The use of hardware video codecs has become increasingly common in recent years, as video content has become more prevalent and the demand for high-quality video has grown. By using dedicated hardware components to perform video processing tasks, hardware codecs are able to deliver high-quality video content while minimizing power consumption and optimizing device performance.

One of the key advantages of hardware video codecs is their ability to support real-time video processing, which makes them ideal for applications such as video conferencing, live streaming, and gaming. They are also well-suited for mobile devices, where power consumption and performance are critical considerations.

Despite their many advantages, there are some limitations to hardware video codecs. For example, they are typically designed to support specific video formats and features, which can limit their flexibility compared to software codecs. They may also be more expensive to implement, which can make them less accessible for smaller companies or individual developers.

Overall, the use of hardware video codecs has become an essential tool for delivering high-quality video content across a wide range of devices and applications. Their ability to deliver fast, power-efficient video processing makes them a critical component of modern video technology.

7.3 Programming tools for encoding and decoding video

For developing video codecs programmers use special tools which help them to write code. These tools are called frameworks.

A framework is a software development tool or platform that provides a structured and standardized way to build applications or software systems. It consists of pre-defined structures, libraries, reusable components, and guidelines that help developers create software efficiently and consistently.

There are several popular frameworks and libraries available for video coding and decoding. Here are a few examples:

- FFmpeg: FFmpeg is a powerful multimedia framework that includes a vast collection of libraries and tools for handling multimedia data. It supports video encoding, decoding, transcoding, and streaming. FFmpeg is widely used and supports a wide range of video codecs;
- GStreamer: GStreamer is an open-source multimedia framework that provides a pipeline-based architecture for handling multimedia data. It offers a wide range of plugins for video encoding, decoding, and processing. GStreamer is highly modular and extensible, allowing developers to create custom multimedia pipelines;
- Video Toolbox Framework (iOS/macOS): Video Toolbox is a framework provided by Apple for video encoding and decoding on iOS and macOS platforms. It leverages the hardware acceleration capabilities of Apple devices, such as the Video Core APIs and dedicated video encoders/decoders. It supports various video formats and codecs;

- MediaCodec (Android): MediaCodec is an Android framework that provides a low-level API for video and audio encoding and decoding. It allows developers to interact directly with the device's hardware codecs, enabling efficient video processing on Android devices;
- libavcodec: libavcodec is a library part of the FFmpeg project, specifically focused on video and audio encoding and decoding. It provides a powerful and flexible API for handling multimedia streams and supports a wide range of video codecs;
- OpenCV: While primarily known for computer vision, OpenCV (Open Source Computer Vision Library) also includes functionalities for video processing, including video decoding, encoding, and manipulation. It is a popular choice for multimedia applications that require computer vision capabilities alongside video processing.

These frameworks and libraries offer different levels of abstraction and support for various platforms. Developers can choose the one that best fits their requirements, programming language preference, and target platform. It's important to refer to the documentation and examples provided by each framework to understand their specific usage and capabilities.

7.4 AMD Advanced Media Framework (AMF)

AMD Advanced Media Framework (AMF) is a powerful software development kit (SDK) developed by Advanced Micro Devices (AMD). It provides a comprehensive set of tools and libraries for developers to leverage the capabilities of AMD GPUs (Graphics Processing Units) in multimedia applications.

AMF is designed to accelerate multimedia processing tasks, including video encoding, decoding, and transcoding, by harnessing the parallel processing capabilities of AMD GPUs. It enables developers to take advantage of hardware-

accelerated video processing, resulting in improved performance, reduced CPU utilization, and enhanced video quality.

One of the key features of AMF is its support for industry-standard video codecs such as H.264 (AVC) and H.265 (HEVC). Developers can utilize the SDK to efficiently encode and decode video streams using these codecs, leveraging the power of AMD GPUs to achieve faster processing times and lower latency.

In addition to video encoding and decoding, AMF also provides capabilities for video transcoding. Transcoding involves converting a video from one format to another while preserving its quality. AMF enables developers to leverage the hardware acceleration capabilities of AMD GPUs to perform high-quality, high-performance transcoding operations, making it an excellent choice for applications that require efficient video format conversion.

AMF supports various programming languages, including C/C++, .NET, and Python, making it accessible to a wide range of developers. The SDK provides a comprehensive set of APIs, allowing developers to integrate AMF functionality seamlessly into their applications and workflows.

Furthermore, AMF is designed to be platform-agnostic, supporting different operating systems such as Windows and Linux. This flexibility allows developers to create multimedia applications that can run on a variety of platforms, making it easier to reach a broader audience.

Overall, AMD Advanced Media Framework (AMF) is a powerful SDK that empowers developers to leverage the advanced capabilities of AMD GPUs for multimedia processing. By providing efficient video encoding, decoding, and transcoding capabilities, AMF enables developers to create high-performance, high-quality multimedia applications across various platforms.

7.6 Nvidia NVENC

NVIDIA NVENC (NVIDIA Encoder) is a hardware-accelerated video encoding technology developed by NVIDIA. It is specifically designed to offload

video encoding tasks from the CPU to the NVIDIA GPU (Graphics Processing Unit), resulting in improved performance, reduced CPU utilization, and faster video encoding times.

NVENC is primarily used for real-time video encoding, commonly employed in applications such as video streaming, game capture, video editing, and video transcoding. By leveraging the dedicated video encoding hardware present in NVIDIA GPUs, NVENC allows for high-performance video encoding with minimal impact on system resources.

NVENC supports various video codecs, including H.264 (AVC) and H.265 (HEVC), which are widely used for video compression. These codecs offer efficient compression algorithms that significantly reduce the file size of videos without significant loss in visual quality. NVENC's hardware-accelerated encoding capabilities ensure that the encoding process is performed swiftly, even at high resolutions and bitrates.

One of the notable advantages of NVENC is its ability to provide a balance between video quality and performance. While software-based encoding solutions utilize CPU resources for video encoding, NVENC takes advantage of the GPU's parallel processing capabilities to deliver fast and efficient video encoding. This allows users to encode videos without a substantial impact on gaming performance or other CPU-intensive tasks.

NVENC is supported on a wide range of NVIDIA GPUs, including those from the GeForce, Quadro, and Tesla series. NVIDIA provides a software development kit (SDK) called NVENC SDK, which allows developers to integrate NVENC into their applications and leverage its video encoding capabilities. The NVENC SDK provides a programming interface that enables developers to access and control the encoding process using NVIDIA's hardware-accelerated encoding capabilities.

Overall, NVIDIA NVENC is a powerful video encoding technology that utilizes the dedicated video encoding hardware present in NVIDIA GPUs to accelerate video encoding tasks. By offloading the encoding process to the GPU,

NVENC provides improved performance and efficiency, making it a valuable tool for applications that require real-time video encoding and transcoding.

7.6 Findings

Video encoding and decoding is a complex task requiring careful study. There are many different video codecs for video processing, the best of which is the H.264 codec.

To improve performance, save resources during encoding and decoding, hardware acceleration is used, i.e. transfer of video processing work from the CPU to the GPU.

The task of encoding and decoding is relevant, but at the moment the existing libraries are either low-level, which makes it difficult for developers to work with them, or are paid and therefore not suitable for many programmers or companies.

Thus, as part of the graduation project, it is necessary to develop a cross-platform library for Windows and Android for encoding and decoding video with support for hardware acceleration, cropping and scaling video.

References

1. Living in a Multi-Codec World – Future Codecs revisited. Paul MacDougall. July 2015. Access mode: https://bitmovin.com/multi-codec-world-2020/
2. Mohammed Ghanbari. Video Coding: An Introduction to Standard Codecs. Institution of Electrical Engineers (1999). pp. 107–120. ISBN 9780852967621
3. The H.264 Advanced Video Compression Standard, Second Edition. Wiley (August, 2010). pp. 29–33. ISBN 9780470516928
4. Hardware and software video encoding comparison. Ramil Safin, Emilia Garipova, Roman Lavrenov. Proceedings of the SICE Annual Conference

2020 (September 23-26 2020). pp. 925-927. Access mode: https://www.researchgate.net/publication/344375475_Hardware_and_software_video_encoding_comparison

Приложение Б

Листинг части библиотеки

Файл «libvideo.cpp»

```
enum EncoderTypes CheckAvailableEncoders() {
	const std::array<std::string, 3> hw_encoders = {"h264_amf", "h264_qsv", "h264_nvenc"};

	std::vector<std::string> avalible_hw_encoders;

	for (const auto &enc : hw_encoders) {
		void *i = 0;
		const AVCodec *cur_codec = av_codec_iterate(&i);
		while (cur_codec) {
			cur_codec = av_codec_iterate(&i);
			if (!cur_codec)
				continue;
			if (strcmp(enc.c_str(), cur_codec->name) == 0)
				avalible_hw_encoders.emplace_back(enc);
		}
	}

	if (!avalible_hw_encoders.empty()) {
		LibvideoLog() << "Available hardware encoders: ";
		for (const auto &hw_enc : avalible_hw_encoders)
			LibvideoLog() << hw_enc << " ";

		LibvideoLog() << "\n";
		LibvideoLog() << "Input name of hardware encoder\nor input 'software' for "
				"software encoding: ";

		std::string encoder_name;
		std::cin >> encoder_name;
```

```
            if (encoder_name == "software")
                  return EncoderTypes::Software;

            auto    result    =    std::find(begin(avalible_hw_encoders),
end(avalible_hw_encoders), "h264_amf");
            if    (encoder_name    ==    "h264_amf"    &&    result    !=
std::end(avalible_hw_encoders))
                  return EncoderTypes::AMF;

            result       =       std::find(begin(avalible_hw_encoders),
end(avalible_hw_encoders), "h264_qsv");
            if    (encoder_name    ==    "h264_qsv"    &&    result    !=
std::end(avalible_hw_encoders))
                  return EncoderTypes::QSV;

            result       =       std::find(begin(avalible_hw_encoders),
end(avalible_hw_encoders), "h264_nvenc");
            if    (encoder_name    ==    "h264_nvenc"    &&    result    !=
std::end(avalible_hw_encoders))
                  return EncoderTypes::CUDA;
      }

      LibvideoLog() << "Can't find any HW encoder. Initialize software
encoder\n";
      return EncoderTypes::Software;
}

enum DecoderTypes CheckAvailableDecoders() {
      std::vector<std::string> avalible_hw_decoders;

      enum AVHWDeviceType type = AV_HWDEVICE_TYPE_NONE;

      LibvideoLog() << "Available hardware decoding device types: ";
```

```
      while     ((type     =     av_hwdevice_iterate_types(type))     !=
AV_HWDEVICE_TYPE_NONE) {
            LibvideoLog() << av_hwdevice_get_type_name(type) << " ";


avalible_hw_decoders.emplace_back(av_hwdevice_get_type_name(type));
      }

      LibvideoLog() << "\n";

      if (!avalible_hw_decoders.empty()) {
            LibvideoLog()  <<  "Input  name  of  device  for  decoding\nor
input 'software' "
                                    "for software decoding: ";
            std::string device_name;
            std::cin >> device_name;

            if (device_name == "software")
                  return DecoderTypes::Software;

            for (const auto &hw_decoder : avalible_hw_decoders) {
                  if (hw_decoder != device_name)
                        continue;

                  type                                                    =
av_hwdevice_find_type_by_name(device_name.c_str());

                  switch (type) {
#ifdef WIN32
                  case AV_HWDEVICE_TYPE_CUDA:
                        return DecoderTypes::CUDA;

                  case AV_HWDEVICE_TYPE_DXVA2:
                        return DecoderTypes::DX9;
```

```
                case AV_HWDEVICE_TYPE_D3D11VA:
                      return DecoderTypes::DX11;

                case AV_HWDEVICE_TYPE_QSV:
                      return DecoderTypes::QSV;

                case AV_HWDEVICE_TYPE_OPENCL:
                      return DecoderTypes::OPENCL;
#endif

#ifdef __ANDROID_API__
#if __ANDROID_API__ > 20
                case AV_HWDEVICE_TYPE_MEDIACODEC:
                      return DecoderTypes::MEDIACODEC;
#endif
#endif

                default:
                      return DecoderTypes::Software;
                }
           }
      }

      LibvideoLog() << "Can't find any HW decoder. Initialize software
decoder\n";
      return DecoderTypes::Software;
}

IEncoder   *CreateEncoder(const   EncoderParams   &enc_params,   enum
EncoderTypes enc_type) {
      if (enc_type == EncoderTypes::Software)
           return new SWEncoder(enc_params);
#ifdef WIN32
```

```
      else
            return new WinHWEncoder(enc_params, enc_type);
#elif __linux__
      return new SWEncoder(enc_params);
#endif
}
#ifdef __ANDROID_API__
#if __ANDROID_API__ > 20
IDecoder   *CreateDecoder(const   DecoderParams   &dec_params,   enum
DecoderTypes dec_type, void *surface)
#endif
#else
IDecoder   *CreateDecoder(const   DecoderParams   &dec_params,   enum
DecoderTypes dec_type)
#endif
{
      if (dec_type == DecoderTypes::Software)
            return new SWDecoder(dec_params);
#ifdef WIN32
      else
            return new WinHWDecoder(dec_params, dec_type);
#elif __ANDROID_API__
#if __ANDROID_API__ > 20
      else
            return new AndroidHWDecoder(dec_params, surface);
#endif
#elif __linux__
      return new SWDecoder(dec_params);
#endif
}
```

Приложение В

Листинг части библиотеки

Файл «SWEncoder.cpp»

```
SWEncoder::SWEncoder(EncoderParams params)
	:                                                    m_params(params),
m_convertor(std::make_unique<RGBAtoYUVConvertor>(m_params.width,
m_params.height)),
                                        m_filtered_width(m_params.width),
m_filtered_height(m_params.height) {
	codec = avcodec_find_encoder(AV_CODEC_ID_H264);
	if (!codec) {
		LibvideoLog() << "Codec not found\n";
		return;
	}

	codec_ctx = avcodec_alloc_context3(codec);
	if (!codec_ctx) {
		LibvideoLog() << "Could not allocate video codec context\
n";
		return;
	}

	codec_ctx->bit_rate = m_params.bit_rate;
	codec_ctx->time_base = {1, m_params.framerate + 1};
	codec_ctx->framerate = {m_params.framerate + 1, 1};

	codec_ctx->gop_size = 0;

	codec_ctx->width = m_params.width;
	codec_ctx->height = m_params.height;

	codec_ctx->pix_fmt = AV_PIX_FMT_YUV420P;
```

```
	if (codec->id == AV_CODEC_ID_H264)
		av_opt_set(codec_ctx->priv_data, "preset",
Utilities::libvideo_h264preset_to_string(m_params.preset), 0);

	ret = avcodec_open2(codec_ctx, codec, nullptr);
	if (ret < 0) {
		LibvideoLog() << "Could not open codec: " <<
av_make_error_string(errBuf, sizeof(errBuf), ret) << "\n";
		return;
	}

	frame = av_frame_alloc();
	if (!frame) {
		LibvideoLog() << "Could not allocate video frame\n";
		return;
	}
	frame->format = codec_ctx->pix_fmt;
	frame->width = codec_ctx->width;
	frame->height = codec_ctx->height;
	frame->pts = 0;

	unfiltered_frame = frame;

	pkt = av_packet_alloc();
	if (!pkt) {
		LibvideoLog() << "Could not allocate packet\n";
		return;
	}

	ret = av_frame_get_buffer(frame, 0);
	if (ret < 0) {
		LibvideoLog() << "Could not allocate the video frame data:
" << av_make_error_string(errBuf, sizeof(errBuf), ret) << "\n";
		return;
```

```
	}

	ret = av_frame_make_writable(frame);
	if (ret < 0) {
		LibvideoLog() << "Could not make frame writable: " <<
av_make_error_string(errBuf, sizeof(errBuf), ret) << "\n";
		return;
	}
}

SWEncoder::~SWEncoder() {
	avcodec_free_context(&codec_ctx);
	av_frame_free(&frame);
	av_packet_free(&pkt);
}

void SWEncoder::encode_frame(std::vector<uint8_t> &rgba_data) {
	preprocessing(rgba_data);

	std::vector<uint8_t> encoded_data;

	frame->pts++;

	ret = avcodec_send_frame(codec_ctx, frame);
	if (ret < 0) {
		LibvideoLog() << "Error sending a frame for encoding: " <<
av_make_error_string(errBuf, sizeof(errBuf), ret) << "\n";
	}

	while (ret >= 0) {
		ret = avcodec_receive_packet(codec_ctx, pkt);
		if (ret == AVERROR(EAGAIN)) {
			ret = avcodec_send_frame(codec_ctx, frame);
			if (ret < 0) {
```

```
                    LibvideoLog() << "Error sending a frame for
encoding: " << av_make_error_string(errBuf, sizeof(errBuf), ret)
                                        << "\n";
                }
                continue;
            } else if (ret == AVERROR_EOF) {
                LibvideoLog() << "End of file: " <<
av_make_error_string(errBuf, sizeof(errBuf), ret) << "\n";
                break;
            } else if (ret < 0) {
                LibvideoLog() << "Error during encoding\n";
            }

            if (m_show_logs) {
                const int initial_size = m_filtered_width *
m_filtered_height * 4;
                show_encoder_info(initial_size, pkt->size, codec_ctx-
>frame_number);
            }

            encoded_data.insert(encoded_data.begin(), pkt->data, pkt-
>data + pkt->size);

            collect_stream_info();

            av_packet_unref(pkt);
            break;
        }

        for (auto &callback : m_out_data_callbacks)
            callback(encoded_data);
}

void SWEncoder::preprocessing(std::vector<uint8_t> &rgba_data) {
```

```
	for (const auto &filter : m_rgba_filters_list)
		rgba_data = filter->filtering_rgba(rgba_data);

	std::vector<uint8_t>	yuv_data	=	m_convertor-
>rgba_to_yuv(rgba_data);

	Utilities::raw_yuv_to_frame(unfiltered_frame, yuv_data);

	for (const auto &filter : m_yuv_filters_list)
		frame = filter->filtering_yuv(unfiltered_frame);
}

bool
SWEncoder::check_existed_filters(std::vector<std::shared_ptr<IFilter>>
&filters_list) {
	for (const auto &filter : filters_list) {
		if (dynamic_cast<CropFilter *>(filter.get())) {
			LibvideoLog() << "Crop filter already added\n";
			return true;
		}

		if (dynamic_cast<ScaleFilter *>(filter.get())) {
			LibvideoLog() << "Scale filter already added\n";
			return true;
		}
	}

	return false;
}

void   SWEncoder::add_crop_filter(std::vector<std::shared_ptr<IFilter>>
&filters_list, const FilterParams &filter_params) {
	float scale_coef = 1.f;
```

```
      ScaleFilter *scale_filter = nullptr;
      for (const auto &filter : m_rgba_filters_list) {
            scale_filter = dynamic_cast<ScaleFilter *>(filter.get());
      }

      for (const auto &filter : m_yuv_filters_list) {
            scale_filter = dynamic_cast<ScaleFilter *>(filter.get());
      }

      if (scale_filter)
            scale_coef = scale_filter->get_scale_coef();

      const int initial_width = (int)(scale_coef * m_params.width);


filters_list.emplace_back(std::make_shared<CropFilter>(initial_width,
filter_params.crop_params.cropped_width,

                filter_params.crop_params.cropped_height,

                                        filter_params.crop_params.x_start,
filter_params.crop_params.y_start));

      m_filtered_width = filter_params.crop_params.cropped_width;
      m_filtered_height = filter_params.crop_params.cropped_height;
}

void  SWEncoder::add_scale_filter(std::vector<std::shared_ptr<IFilter>>
&filters_list, const FilterParams &filter_params) {
      int initial_width = m_params.width;
      int initial_height = m_params.height;

      CropFilter *crop_filter = nullptr;
      for (const auto &filter : m_rgba_filters_list) {
```

```
            crop_filter = dynamic_cast<CropFilter *>(filter.get());
      }

      if (crop_filter) {
            initial_width = crop_filter->get_width();
            initial_height = crop_filter->get_height();
      }

      filters_list.emplace_back(
            std::make_shared<ScaleFilter>(initial_width,
initial_height, filter_params.scale_params.scale_coef));

      m_filtered_width = (int)(filter_params.scale_params.scale_coef *
initial_width);
      m_filtered_height = (int)(filter_params.scale_params.scale_coef *
initial_height);
}

void SWEncoder::add_filter(const FilterParams &filter_params) {

      if (check_existed_filters(m_rgba_filters_list))
            return;

      if (check_existed_filters(m_yuv_filters_list))
            return;

      if (filter_params.color_space == ColorSpace::RGBA) {
            if (filter_params.filter_type == FilterType::Crop)
                  add_crop_filter(m_rgba_filters_list, filter_params);

            if (filter_params.filter_type == FilterType::Scale)
                  add_scale_filter(m_rgba_filters_list, filter_params);
      }
```

```
	if (filter_params.color_space == ColorSpace::YUV) {
		if (filter_params.filter_type == FilterType::Crop)
			add_crop_filter(m_yuv_filters_list, filter_params);

		if (filter_params.filter_type == FilterType::Scale)
			add_scale_filter(m_yuv_filters_list, filter_params);
	}

	change_configuration(filter_params);
}

void SWEncoder::collect_stream_info() {
	const int initial_size = m_params.width * m_params.height * 4;

	m_current_frame_info.initial_size = initial_size;
	m_current_frame_info.pkt_size = pkt->size;
	m_current_frame_info.enc_pts = codec_ctx->frame_number;
}

void SWEncoder::change_configuration(const FilterParams
&filter_params) {
	avcodec_free_context(&codec_ctx);
	av_frame_free(&frame);

	codec = avcodec_find_encoder(AV_CODEC_ID_H264);
	if (!codec) {
		LibvideoLog() << "Codec not found\n";
		return;
	}

	codec_ctx = avcodec_alloc_context3(codec);
	if (!codec_ctx) {
		LibvideoLog() << "Could not allocate video codec context\
n";
```

```
            return;
        }

        codec_ctx->bit_rate = m_filtered_width * m_filtered_height * 4 *
8;
        codec_ctx->time_base = {1, m_params.framerate + 1};
        codec_ctx->framerate = {m_params.framerate + 1, 1};

        codec_ctx->width = m_filtered_width;
        codec_ctx->height = m_filtered_height;

        codec_ctx->pix_fmt = AV_PIX_FMT_YUV420P;

        if (codec->id == AV_CODEC_ID_H264)
            av_opt_set(codec_ctx->priv_data,                    "preset",
Utilities::libvideo_h264preset_to_string(m_params.preset), 0);

        ret = avcodec_open2(codec_ctx, codec, nullptr);
        if (ret < 0) {
            LibvideoLog()   <<   "Could   not   open   codec:   "   <<
av_make_error_string(errBuf, sizeof(errBuf), ret) << "\n";
            return;
        }

        frame = av_frame_alloc();
        if (!frame) {
            LibvideoLog() << "Could not allocate video frame\n";
            return;
        }
        frame->format = codec_ctx->pix_fmt;

        if (filter_params.color_space == ColorSpace::RGBA) {
            frame->width = m_filtered_width;
            frame->height = m_filtered_height;
```

```
	}

	if (filter_params.color_space == ColorSpace::YUV) {
		frame->width = m_params.width;
		frame->height = m_params.height;
	}
	frame->pts = 0;

	unfiltered_frame = frame;

	ret = av_frame_get_buffer(frame, 0);
	if (ret < 0) {
		LibvideoLog() << "Could not allocate the video frame data:
" << av_make_error_string(errBuf, sizeof(errBuf), ret) << "\n";
		return;
	}

	ret = av_frame_make_writable(frame);
	if (ret < 0) {
		LibvideoLog() << "Could not make frame writable: " <<
av_make_error_string(errBuf, sizeof(errBuf), ret) << "\n";
		return;
	}

	m_convertor = std::make_unique<RGBAtoYUVConvertor>(frame->width,
frame->height);
}

void SWEncoder::add_output(std::function<void(std::vector<uint8_t> &)>
&&out_callback) {
	m_out_data_callbacks.emplace_back(std::move(out_callback));
}
```

```
void SWEncoder::set_logs_active(bool enable_logs) { m_show_logs = enable_logs; }

void SWEncoder::show_encoder_info(int initial_size, int pkt_size, int enc_pts) {
     LibvideoLog() << "Encoder frame " << std::to_string(enc_pts) << " log: \n"
                                  << "frame size: " << std::to_string(initial_size) << "\n"
                          << "encoded size: " << std::to_string(pkt_size) << "\n\n";
}
```

Приложение Г

Листинг части библиотеки

Файл «SWDecoder.cpp»

```
SWDecoder::SWDecoder(DecoderParams params)
      :                                                    m_params(params),
m_convertor(std::make_unique<YUVtoRGBAConvertor>(m_params.width,
m_params.height)),
                                          m_filtered_width(m_params.width),
m_filtered_height(m_params.height) {

      parser_ctx = av_parser_init(AV_CODEC_ID_H264);
      if (!parser_ctx) {
            LibvideoLog() << "Parser not found\n";
            return;
      }

      pkt = av_packet_alloc();
      if (!pkt) {
            LibvideoLog() << "Could not allocate packet\n";
            return;
      }

      frame = av_frame_alloc();
      if (!frame) {
            LibvideoLog() << "Could not allocate video frame\n";
            return;
      }
      filtered_frame = frame;

      codec = avcodec_find_decoder(AV_CODEC_ID_H264);
      if (!codec) {
            LibvideoLog() << "Codec not found\n";
            return;
```

```
    }

    codec_ctx = avcodec_alloc_context3(codec);
    if (!codec_ctx) {
        LibvideoLog() << "Could not allocate video cdsaodec
context\n";
        return;
    }

#ifdef __ANDROID_API__
#if __ANDROID_API__ > 20
    if (m_params.extradata.size()) {
        av_freep(&codec_ctx->extradata);
        codec_ctx->extradata = (uint8_t
*)av_mallocz(m_params.extradata.size() +
AV_INPUT_BUFFER_PADDING_SIZE);
        memcpy(codec_ctx->extradata, m_params.extradata.data(),
m_params.extradata.size());
        codec_ctx->extradata_size = m_params.extradata.size();
    }
#endif
#endif

    ret = avcodec_open2(codec_ctx, codec, nullptr);
    if (ret < 0) {
        LibvideoLog() << "Could not open codec\n";
        return;
    }
}

SWDecoder::~SWDecoder() {
    av_parser_close(parser_ctx);
    avcodec_free_context(&codec_ctx);
    av_frame_free(&frame);
```

```
	av_packet_free(&pkt);
}

void SWDecoder::decode(std::vector<uint8_t> &encoded_data) {
	encoded_data.insert(encoded_data.end(),
AV_INPUT_BUFFER_PADDING_SIZE, 0);

	parse(encoded_data);

	for (auto &callback : m_out_data_callbacks)
		callback(m_decoded_data);

#ifdef WIN32
	m_wnd_manager.draw({&m_decoded_data,	&m_current_frame_info,
m_show_logs});
#endif
}

void SWDecoder::add_output(std::function<void(std::vector<uint8_t> &)>
&&out_callback) {
	m_out_data_callbacks.emplace_back(std::move(out_callback));
}

#ifdef WIN32
void SWDecoder::add_output(HWND hwnd) {
	RECT rect;
	GetClientRect(hwnd, &rect);
	const int window_width = rect.right - rect.left;
	const int window_height = rect.bottom - rect.top;

	if ((window_width != m_filtered_width) || (window_height !=
m_filtered_height)) {
		LibvideoLog() << "The window dimensions do not match the
image dimensions\n";
```

```
            return;
      }

      m_wnd_manager.initialize_window(hwnd);
}
#endif

bool
SWDecoder::check_existed_filters(std::vector<std::shared_ptr<IFilter>>
&filters_list) {
      for (const auto &filter : filters_list) {
            if (dynamic_cast<CropFilter *>(filter.get())) {
                  LibvideoLog() << "Crop filter already added\n";
                  return true;
            }

            if (dynamic_cast<ScaleFilter *>(filter.get())) {
                  LibvideoLog() << "Scale filter already added\n";
                  return true;
            }
      }

      return false;
}
void   SWDecoder::add_crop_filter(std::vector<std::shared_ptr<IFilter>>
&filters_list, const FilterParams &filter_params) {
      float scale_coef = 1.f;

      ScaleFilter *scale_filter = nullptr;
      for (const auto &filter : m_rgba_filters_list) {
            scale_filter = dynamic_cast<ScaleFilter *>(filter.get());
      }

      for (const auto &filter : m_yuv_filters_list) {
```

```
            scale_filter = dynamic_cast<ScaleFilter *>(filter.get());
      }

      if (scale_filter)
            scale_coef = scale_filter->get_scale_coef();

      const   int   initial_width   =   static_cast<int>(scale_coef   *
m_params.width);


filters_list.emplace_back(std::make_shared<CropFilter>(initial_width,
filter_params.crop_params.cropped_width,

               filter_params.crop_params.cropped_height,

                                      filter_params.crop_params.x_start,
filter_params.crop_params.y_start));

      m_filtered_width = filter_params.crop_params.cropped_width;
      m_filtered_height = filter_params.crop_params.cropped_height;
}
void  SWDecoder::add_scale_filter(std::vector<std::shared_ptr<IFilter>>
&filters_list, const FilterParams &filter_params) {
      int initial_width = m_params.width;
      int initial_height = m_params.height;

      CropFilter *crop_filter = nullptr;
      for (const auto &filter : m_rgba_filters_list) {
            crop_filter = dynamic_cast<CropFilter *>(filter.get());
      }

      if (crop_filter) {
            initial_width = crop_filter->get_width();
            initial_height = crop_filter->get_height();
```

```
	}

	filters_list.emplace_back(
		std::make_shared<ScaleFilter>(initial_width,
initial_height, filter_params.scale_params.scale_coef));

	m_filtered_width =
static_cast<int>(filter_params.scale_params.scale_coef *
initial_width);
	m_filtered_height =
static_cast<int>(filter_params.scale_params.scale_coef *
initial_height);
}

void SWDecoder::add_filter(const FilterParams &filter_params) {
	if (check_existed_filters(m_rgba_filters_list))
		return;

	if (check_existed_filters(m_yuv_filters_list))
		return;

	if (filter_params.color_space == ColorSpace::RGBA) {
		if (filter_params.filter_type == FilterType::Crop)
			add_crop_filter(m_rgba_filters_list, filter_params);

		if (filter_params.filter_type == FilterType::Scale)
			add_scale_filter(m_rgba_filters_list, filter_params);
	}

	if (filter_params.color_space == ColorSpace::YUV) {
		if (filter_params.filter_type == FilterType::Crop)
			add_crop_filter(m_yuv_filters_list, filter_params);

		if (filter_params.filter_type == FilterType::Scale)
```

```
                add_scale_filter(m_yuv_filters_list, filter_params);

          m_convertor                                                    =
std::make_unique<YUVtoRGBAConvertor>(m_filtered_width,
m_filtered_height);
     }
}

void  SWDecoder::set_logs_active(bool  enable_logs)  {  m_show_logs  =
enable_logs; }

void SWDecoder::parse(std::vector<uint8_t> &encoded_data) {
     int data_size = static_cast<int>(encoded_data.size());

     uint8_t *data = encoded_data.data();

     bool data_end = false;

     while (!data_end) {
          data_end = (data_size) ? false : true;

          ret  =  av_parser_parse2(parser_ctx,  codec_ctx,  &pkt->data,
&pkt->size, data, data_size, AV_NOPTS_VALUE, AV_NOPTS_VALUE, 0);
          if (ret < 0) {
               LibvideoLog() << "Error while parsing\n";
               return;
          }

          data += ret;
          data_size -= ret;

          if (pkt->size)
               decode_data();
     }
```

```
}

void SWDecoder::decode_data() {
	ret = avcodec_send_packet(codec_ctx, pkt);
	if (ret < 0) {
		LibvideoLog() << "Error sending a packet for decoding\n";
		return;
	}

	while (ret >= 0) {
		ret = avcodec_receive_frame(codec_ctx, frame);
		if (ret == AVERROR(EAGAIN)) {
			ret = avcodec_send_packet(codec_ctx, pkt);
			if (ret < 0) {
				LibvideoLog() << "Error sending a packet for decoding: " << av_make_error_string(errBuf, sizeof(errBuf), ret)
					<< "\n";
			}
			continue;
		} else if (ret == AVERROR_EOF) {
			LibvideoLog() << "End of file: " << av_make_error_string(errBuf, sizeof(errBuf), ret) << "\n";
		}

		if (m_show_logs) {
			collect_stream_info();
			show_decoder_info(m_filtered_width, m_filtered_height, codec_ctx->bit_rate, m_current_frame_info.framerate,
				codec_ctx->frame_number);
		}

		break;
	}
```

```
	postprocessing();
}

void SWDecoder::postprocessing() {
	if (m_params.width != frame->width || m_params.height != frame->height) {
		LibvideoLog() << "Decoder initialization sizes do not match decoded frame sizes\n";

		m_convertor = std::make_unique<YUVtoRGBAConvertor>(frame->width, frame->height);

		m_params.width = frame->width;
		m_params.height = frame->height;
	}

	for (const auto &filter : m_yuv_filters_list)
		filtered_frame = filter->filtering_yuv(frame);

	std::vector<uint8_t> temp_rgba = m_convertor->yuv_to_rgba(filtered_frame);

	for (const auto &filter : m_rgba_filters_list)
		temp_rgba = filter->filtering_rgba(temp_rgba);

	std::swap(m_decoded_data, temp_rgba);
}

void SWDecoder::show_decoder_info(int width, int height, int64_t bitrate, int framerate, int dec_pts) {
	LibvideoLog() << "Decoder frame " << std::to_string(dec_pts) << " log: \n"
			<< "resolution: " << std::to_string(width) << "x" << std::to_string(height) << "\n"
```

```
                        << "bitrate: " << std::to_string(bitrate) << "\
n"
                         << "framerate: " << std::to_string(framerate)
<< "\n\n";
}

void SWDecoder::collect_stream_info() {
      const int framerate = calc_framerate();

      m_current_frame_info.width = m_params.width;
      m_current_frame_info.height = m_params.height;
      m_current_frame_info.bitrate = codec_ctx->bit_rate;
      m_current_frame_info.framerate = framerate;
      m_current_frame_info.dec_pts = codec_ctx->frame_number;
}

int SWDecoder::calc_framerate() {
      second_time_point = std::chrono::steady_clock::now();

      std::chrono::duration<double> time_span =


std::chrono::duration_cast<std::chrono::duration<double>>(second_time_
point - first_time_point);

      first_time_point = second_time_point;

      return static_cast<int>(1 / time_span.count());
}
```

Приложение Д

Листинг сборки библиотеки

Файл «prebuild_thirdparty.py»

```
import sys, os
from sys import platform

build_type = "--build-release"
cflags_path = ""
enable_amf = "--disable-amf"
enable_clang_tidy = "--disable-tidy"
enable_android_build = False

for i in range(1, len(sys.argv)):
    if (sys.argv[i] == "--build-release"):
        build_type = "--build-release"

    if (sys.argv[i] == "--build-debug"):
        build_type = "--build-debug"

    if (sys.argv[i] == "-cflags"):
        cflags_path = sys.argv[i + 1]

    if (sys.argv[i] == "--enable-amf"):
        enable_amf = sys.argv[i]

    if (sys.argv[i] == "--enable-tidy"):
        enable_clang_tidy = sys.argv[i]

    if (sys.argv[i] == "--android"):
        enable_android_build = True

if (platform == "linux" or platform == "linux2"):
    if enable_android_build:
```

```
        archs = ["arm64-v8a", "armeabi-v7a", "x86_64", "x86"]

                                    profile_build    =    os.getcwd()    +
"/.ci/conan_profiles/conan_profile_linux_release"

        if (build_type == "--build-debug"):
                                        profile_build   =   os.getcwd()   +
"/.ci/conan_profiles/conan_profile_linux_debug"

        for arch in archs:
                                        profile_host   =   os.getcwd()   +
"/.ci/conan_profiles/Android/Release/host_android_"     +     arch     +
"_release"

            if (build_type == "--build-debug"):
                                          profile_host   =   os.getcwd()   +
"/.ci/conan_profiles/Android/Debug/host_android_" + arch + "_debug"

            optinons = "-o "
            if (enable_clang_tidy == "--enable-tidy"):
                optinons += "with_clangtidy=True"
            else:
                optinons += "with_clangtidy=False"

                                  libx264DeployCommand   =   "conan   create
.ci/libx264_conan_recipe  libx264/cci.20220602@libvideo/stable  -pr:h  "
+ profile_host + " -pr:b=" + profile_build + " --build=missing"
                                    ffmpegDeployCommand   =   "conan   create
.ci/ffmpeg_conan_recipe    ffmpeg/5.0@libvideo/stable    -pr:h    "    +
profile_host + " -pr:b=" + profile_build + " --build=missing"
                                libvideoDeployCommand   =   "conan   create   .
libvideo/1.0.0@ccor/libvideo  -tf  None  -pr:h  "  +  profile_host  +  "  -
pr:b=" + profile_build + " " + optinons + " --build=missing"
```

```
            os.system(libx264DeployCommand)
            os.system(ffmpegDeployCommand)
            os.system(libvideoDeployCommand)
    else:
                                    profile     =     os.getcwd()     +
"/.ci/conan_profiles/conan_profile_linux_release"

        if (build_type == "--build-debug"):
                                        profile    =    os.getcwd()    +
"/.ci/conan_profiles/conan_profile_linux_debug"

        optinons = "-o "
        if (enable_clang_tidy == "--enable-tidy"):
            optinons += "with_clangtidy=True"
        else:
            optinons += "with_clangtidy=False"

         libx264DeployCommand = "conan create .ci/libx264_conan_recipe
libx264/cci.20220602@libvideo/stable   -pr="   +   profile   +   "   --
build=missing"
           ffmpegDeployCommand = "conan create .ci/ffmpeg_conan_recipe
ffmpeg/5.0@libvideo/stable -pr=" + profile + " --build=missing"
                        libvideoDeployCommand   =   "conan   create   .
libvideo/1.0.0@ccor/libvideo   -tf   None   -pr="   +   profile   +   "   --
build=missing"
           libvideoInstallCommand = "conan install . --install-folder
build_ -pr=" + profile + " " + optinons + " --build=missing"

        os.system(libx264DeployCommand)
        os.system(ffmpegDeployCommand)
        os.system(libvideoDeployCommand)
        os.system(libvideoInstallCommand)

if (platform == "win32"):
```

```
            profile = os.getcwd() + "\.ci\conan_profiles\
conan_profile_release"

    if (build_type == "--build-debug"):
                profile = os.getcwd() + "\.ci\conan_profiles\
conan_profile_debug"

    if (cflags_path != ""):
        cflags_path = "-e " + "CFLAGS=" + "-I" + cflags_path

    optinons = "-o "
    if (enable_amf == "--enable-amf"):
        optinons += "with_amf=True "
        enable_amf = "-o with_amf=True"
    else:
        optinons += "with_amf=False "
        enable_amf = "-o with_amf=False"

    if (enable_clang_tidy == "--enable-tidy"):
        optinons += "-o with_clangtidy=True"
    else:
        optinons += "-o with_clangtidy=False"

       libx264DeployCommand = "conan create .ci/libx264_conan_recipe
libx264/cci.20220602@libvideo/stable -pr=" + profile + " --
build=missing"
        ffmpegDeployCommand = "conan create .ci/ffmpeg_conan_recipe
ffmpeg/5.0@libvideo/stable -pr=" + profile + " " + cflags_path + " " +
enable_amf + " --build=missing"
     libvideoInstallCommand = "conan install . --install-folder build_
-pr=" + profile + " " + cflags_path + " " + optinons + " --
build=missing"

    os.system(libx264DeployCommand)
```

```
os.system(ffmpegDeployCommand)
os.system(libvideoInstallCommand)
```