

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Аналіз та написання оптимізаційних методів при
створенні нелінійного відеоредактора для
підвищення продуктивності та поліпшення
користувацького досвіду»**

Виконав студент групи ІІЗм-23-1
ПОЛОНЕВИЧ Денис Вадимович

Керівник роботи:
БАКАЛЮК Тетяна Анатоліївна

Рецензент:
ЧИЖМОТРЯ Олена Геннадіївна

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«01» жовтня 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ПОЛОНЕВИЧ Денис Вадимович**

Керівник роботи: **ВАКАЛЮК Тетяна Анатоліївна**

Тема роботи: **«Аналіз та написання оптимізаційних методів при створенні нелінійного відеоредактора для підвищення продуктивності та поліпшення користувацького досвіду»,**

затверджена Наказом закладу вищої освіти від **«01» жовтня 2024 р. № 520/с**

Вихідні дані для роботи: об'єктом дослідження є нелінійний відеоредактор – програмний продукт, який використовується для монтажу та обробки відеофайлів, предметом дослідження є оптимізаційні методи, спрямовані на підвищення продуктивності та поліпшення користувацького досвіду в нелінійному відеоредакторі, а також аналіз їх впливу на якість та швидкодію редагування відео.

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Вакалюк Т.А.	01.10.2024р.	01.10.2024р.
2	Вакалюк Т.А.	02.10.2024р.	02.10.2024р.
3	Вакалюк Т.А.	03.10.2024р.	03.10.2024р.

РЕФЕРАТ

Випускна робота магістра включає в себе аналіз та написання оптимізаційних методів при створенні нелінійного відеоредактора для підвищення продуктивності та поліпшення користувацького досвіду. Робота складається з програмної реалізації нелінійного відеоредактора та пояснювальної записки.

Пояснювальна записка до випускної роботи містить 77 сторінок, 30 ілюстрацій, 7 таблиць, 3 додатки та 41 використане джерело.

В першому розділі були описані існуючі розробки нелінійних відеоредакторів, що призначені для підвищення продуктивності та поліпшення користувацького досвіду. Було обрано технології для розробки прототипу нелінійного відеоредактора. Наприкінці, великий акцент був приділений наявним оптимізаційним методам при створенні нелінійного відеоредактора.

В другому розділі було змодельовано архітектуру та поведінкову модель роботи відеоредактора. Крім цього, було реалізовано власні оптимізаційні методи при створенні нелінійного відеоредактора за допомогою обраних алгоритмів.

В третьому розділі було продемонстровано практичну реалізацію створеного нелінійного відеоредактора та його тестування.

У висновку було підведено підсумки про те, що вдалося дослідити та розробити в кваліфікаційній роботі.

КЛЮЧОВІ СЛОВА: НЕЛІНІЙНИЙ ВІДЕОРЕДАКТОР, ОПТИМІЗАЦІЙНІ МЕТОДИ, ПІРАМІДА ГАУСА, ЛАПЛАСОВА ПІРАМІДА, М'ЮТЕКС.

ABSTRACT

The master's thesis includes the analysis and writing of optimization methods when creating a nonlinear video editor to increase productivity and improve user experience. The work consists of a software implementation of a nonlinear video editor and an explanatory note.

The explanatory note to the thesis contains 77 pages, 30 illustrations, 7 tables, 3 appendices and 41 sources used.

The first section describes the existing developments of nonlinear video editors, which are designed to increase productivity and improve user experience. Technologies were selected for the development of a prototype of a nonlinear video editor. Finally, a great emphasis was placed on existing optimization methods when creating a nonlinear video editor.

In the second section, the architecture and behavioral model of the video editor were modeled. In addition, our own optimization methods were implemented when creating a nonlinear video editor using selected algorithms.

The third section demonstrated the practical implementation of the created nonlinear video editor and its testing.

The conclusion summarized what was researched and developed in the qualification work.

KEYWORDS: NONLINEAR VIDEO EDITOR, OPTIMIZATION METHODS, GAUSSIAN PYRAMID, LAPLASIAN PYRAMID, MUTEX.

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1. ТЕОРЕТИЧНИЙ АНАЛІЗ ПРОБЛЕМИ ДОСЛІДЖЕННЯ	10
1.1 ПОСТАНОВКА ЗАДАЧІ	10
1.2 ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ ВІДЕОРЕДАКТОРІВ	11
1.3 ВИБІР І ОБҐРУНТУВАННЯ СТЕКУ РОЗРОБКИ ПРОТОТИПУ ВІДЕОРЕДАКТОРА	17
1.4 ВИБІР І ОБҐРУНТУВАННЯ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ РОЗРОБКИ ПРОТОТИПУ ВІДЕОРЕДАКТОРА.....	22
1.5 АНАЛІЗ ФУНКЦІОНАЛЬНО-ОПТИМІЗАЦІЙНИХ МЕТОДІВ ПРИ РОБОТІ З ВІДЕО	23
Висновки до розділу 1.....	28
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ НЕЛІНІЙНОГО ВІДЕОРЕДАКТОРА	30
2.1 ВИЗНАЧЕННЯ ВАРІАНТІВ ВИКОРИСТАННЯ ВІДЕОРЕДАКТОРА ТА ОБ’ЄКТНО- ОРІЄНТОВАНОЇ СТРУКТУРИ	30
2.2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ АЛГОРИТМІВ РОБОТИ СИСТЕМИ	38
2.3 РЕАЛІЗАЦІЯ ОПТИМІЗАЦІЙНИХ МЕТОДІВ ПРИ СТВОРЕННІ НЕЛІНІЙНОГО ВІДЕОРЕДАКТОРА.....	42
Висновки до розділу 2.....	48
РОЗДІЛ 3. ПОРЯДОК РОБОТИ ТА ТЕСТУВАННЯ НЕЛІНІЙНОГО ВІДЕОРЕДАКТОРА	49
3.1 ПОРЯДОК РОБОТИ НЕЛІНІЙНОГО ВІДЕОРЕДАКТОРА	49
3.2 ТЕСТУВАННЯ НЕЛІНІЙНОГО ВІДЕОРЕДАКТОРА	58
Висновки до розділу 3.....	60
ВИСНОВКИ.....	62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	64

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

GP – (англ. Gaussian pyramid) – піраміда Гауса

LP – (англ. Laplacian pyramid) – піраміда Лапласа

MUTEX – (англ. MUTual EXclusion) - взаємне виключення, м'ютекс

FFMPEG – (англ. Fast Forward Moving Picture Experts Group) – комплекс програм та бібліотек для обробки відео

ВСТУП

Актуальність теми. Аналіз та написання оптимізаційних методів при створенні нелінійного відеоредактора є надзвичайно актуальною темою в сучасному контексті розвитку інформаційних технологій. Сьогодні, коли відеоконтент стає одним із основних способів комунікації та передачі інформації, ефективність та зручність інструментів для його редагування набувають особливого значення [4].

Однією з головних проблем є оптимізація швидкості обробки відео, зокрема, забезпечення швидкої та плавної роботи при редагуванні високоякісного відео. Робота з відеоефектами, трансформувannya зображення, колажування та інші графічні прийоми, що напряду працюють з відео потоком, є доволі вимогливими функціональними можливостями у ресурсах для машини кожного користувача [5]. Сучасні відеоредактори часто стикаються з проблемами затримок і зависань, що негативно впливає на робочий процес.

Ще одним важливим аспектом є забезпечення інтуїтивного та зручного інтерфейсу користувача. Це включає в себе не лише зручність навігації, але й доступність та простоту використання складних функцій для початківців і професіоналів [3]. У сфері кінематографії та відеопродукції, оптимізація процесу редагування може значно знизити витрати часу та ресурсів, що особливо важливо при створенні високоякісного контенту.

Відеоредактори широко використовуються в освітніх програмах для створення навчальних матеріалів. Покращення продуктивності та зручності інструментів дозволить створювати більш ефективні та привабливі навчальні відео, що сприятиме кращому засвоєнню матеріалу учнями [6].

Метою дослідження є написання та аналіз оптимізаційних методів для створення нелінійного відеоредактора. Основною задачею є покращення ефективності та продуктивності процесу редагування відеоматеріалів шляхом оптимізації внутрішніх механізмів редактора.

Встановлена мета обумовлює наступні **завдання**:

- проаналізувати існуючі розробки нелінійних відеоредакторів;
- проаналізувати існуючі оптимізаційні функціональні методи при створенні відеоредактора;
- обрати технології розробки редактора;
- спроектувати архітектуру програмного застосунку;
- змодельовати поведінку роботи нелінійного відеоредактора;
- за допомогою спроектованої архітектури системи реалізувати власні оптимізаційні методи по роботі з функціоналом редактора та відповідно розробити сам програмний застосунок у вигляді нелінійного відеоредактора;

Об’єктом дослідження є нелінійний відеоредактор – програмний продукт, який використовується для монтажу та обробки відеофайлів.

Предметом дослідження є функціонально-оптимізаційні методи, спрямовані на підвищення продуктивності та поліпшення користувацького досвіду в нелінійному відеоредакторі, а також аналіз їх впливу на якість та швидкодію редагування відео.

Наукова новизна отриманих результатів полягає у тому, що:

- спроектовано власний підхід створення нелінійного відеоредактора;
- реалізовано власні оптимізаційно-функціональні методи відеоредактора.

Практичне значення полягає у тому, що:

- створено відеоредактор, що надає можливість до обробки відео базового та середніх рівнів – від різних трансформацій картинки до додавання водяних знаків та об’єднання;
- розроблені оптимізаційні методи надають можливість відеоредактору працювати з відео різних розмірів та якості, при цьому зберігаючи швидкість в процесі редагування так і в процесі рендеренгу вихідного відео.

Апробація результатів роботи. За матеріалами дослідження взято участь в 3 конференціях та опубліковано праці [1,2,3].

РОЗДІЛ 1. ТЕОРЕТИЧНИЙ АНАЛІЗ ПРОБЛЕМИ ДОСЛІДЖЕННЯ

1.1 Постановка задачі

Створення нелінійного відеоредактора нині є стратегічно важливим завданням, оскільки відеоконтент не тільки розважає, але й використовується у багатьох сферах: від маркетингу та освіти до наукових досліджень. Відеоредактор, який дозволяє швидко та ефективно створювати привабливий та високоякісний вміст, стає ключовим інструментом для вираження та розвитку креативності.

Оптимізація відеоредакторів відіграє важливу роль у покращенні їхньої продуктивності та економії ресурсів. Це є ключовим завданням для кожного програміста, який прагне опанувати розробку такого програмного забезпечення. Існують два підходи: перший - аналіз та розуміння існуючих рішень для реалізації функціональної частини відеоредактора, починаючи від створення вікон для трансляції відео до його редагування та збереження на персональний комп'ютер; другий - розробка власних рішень, що сприятиме глибшому розумінню проблематики та важливості написання чистого та ефективного коду [8].

Базуючись на інформації, що згадана вище, було вирішено об'єднати два підходи та розробити нелінійний відеоредактор, що виконуватиме поставлені задачі. Тепер користувач матиме можливість до швидкого та легкого у використанні відеоредактора.

Програмний продукт повинен мати наступні функціональні можливості:

- імпортування відео різних форматів;
- зрозумілий користувацький інтерфейс, починаючи від слайдерів для відтворення відео до функціональних кнопок “Старт” / “Імпорт”, “Пауза” та інше;
- базове редагування відео: зміна контрасту, яскравості, трансформування по позиції;

- можливість зміни кольорокорекції;
- зміна розміру вихідного відео за допомогою методу Resize або більш технічна імплементація за рахунок піраміди Лапласа та Гауса;
- обрізання різних частин відео, так звані Zoom-операції;
- додавання фотографій поверх відео з можливостями трансформування першого об'єкту;
- накладання відео на відео, так звані Blend та Merge-операції;
- робота з аудіо та можливість розділення аудіо та відео-доріжок.

Перед тим як розпочати розробку нелінійного відеоредактора, необхідно провести детальний аналіз існуючих відеоредакторів та їх оптимізаційних функціональних методів, що забезпечують програмні можливості. Це дозволить зрозуміти, які рішення вже застосовуються в інших відеоредакторах, які їхні сильні та слабкі сторони, а також які технології можна використовувати для досягнення найкращих результатів.

Аналіз включатиме вивчення функціональних реалізацій, що пропонують імпортування відео різних форматів, базові функції редагування, зміни кольорокорекції, розмірів та обрізання відео, додавання фотографій і накладання одного відео на інше, а також роботу з аудіо-доріжками. Це стане фундаментом для створення конкурентоспроможного та ефективного програмного продукту.

1.2 Огляд та аналіз існуючих відеоредакторів

В нашій час людство має доступ до багатьох рішень і ідея створення чогось абсолютно нового стає важким викликом. У процесі проєктування програмних продуктів важливо знати про готові рішення та вивчати їх, щоб краще розуміти вимоги до програмного забезпечення з досвіду користувачів, щоб сформулювати унікальну ціннісну пропозицію для продукту, що створюватиметься.

Для порівняльного аналізу були обрані наступні нелінійні відеоредактори: Kdenlive та Vegas Pro.

Kdenlive (англ. KDE Non-Linear Video Editor) – це потужний нелінійний відеоредактор з відкритим вихідним кодом, є частиною проєкту KDE. Вперше випущений у 2002 році, з того часу Kdenlive постійно вдосконалюється завдяки активній спільноті розробників. Програма написана на C++ з використанням бібліотеки Qt для створення графічного інтерфейсу та бібліотеки MLT для обробки мультимедіа [9].

Цей редактор використовують як аматори, так і професійні відеомонтажери для реалізації різноманітних відеопроєктів – від домашніх відео до незалежних фільмів та онлайн-контенту. Завдяки підтримці широкого спектра відеоформатів і сумісності з різними операційними системами, такими як Linux, Windows і MacOS, Kdenlive залишається актуальним і популярним інструментом у світі відеомонтажу. Головними перевагами є зручний інтерфейс та потужні функціональні можливості, що роблять редактор привабливим вибором для тих, хто шукає безкоштовний, але ефективний відеоредактор [10].

На рис. 1.1 зображено інтерфейс відеоредактора.

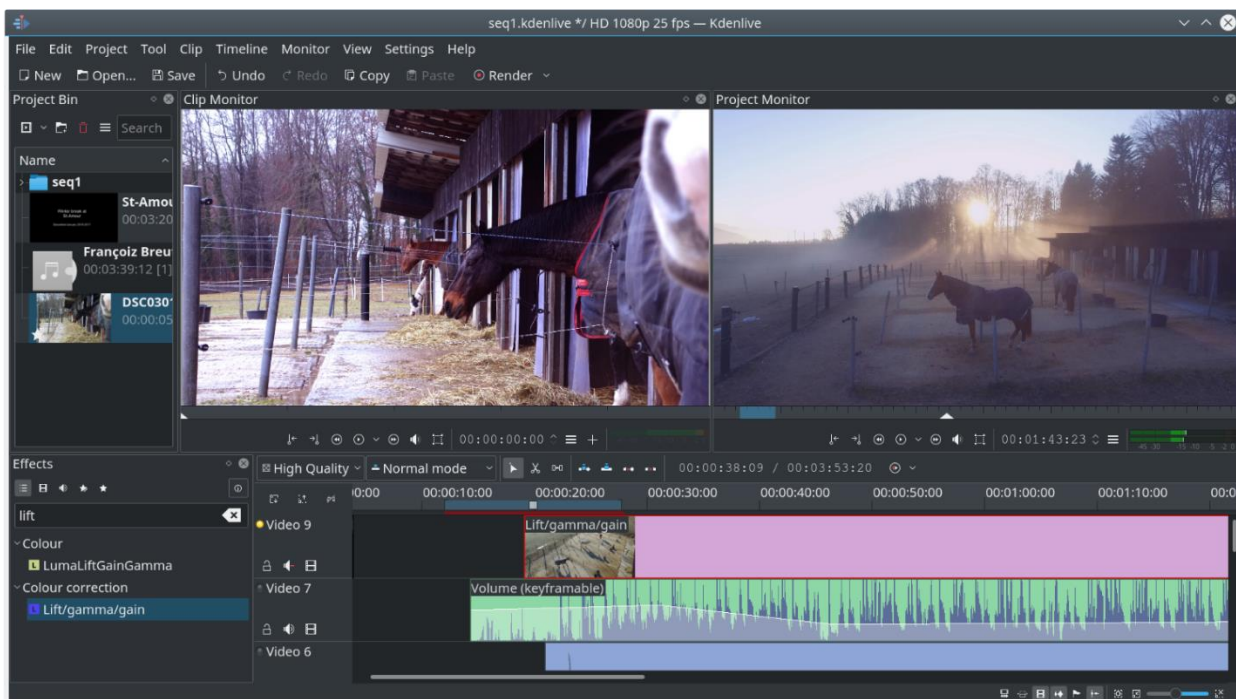


Рис. 1.1 – Kdenlive (Інтерфейс відеоредактора)

Цей програмний продукт має кілька важливих функціональних переваг. По-перше, Kdenlive є безкоштовним і має відкритий код, що дозволяє користувачам безкоштовно користуватися програмою та отримувати доступ до вихідного коду. По-друге, він пропонує широкий спектр функцій, включаючи обрізання та склеювання відео, додавання тексту, застосування фільтрів і переходів, а також роботу з аудіо. По-третє, програма підтримує різні формати відео та аудіо, що робить її дуже гнучкою. Крім того, Kdenlive є мультиплатформенним продуктом, що означає його доступність на різних операційних системах. Нарешті, інтерфейс користувача Kdenlive є зручним і інтуїтивно зрозумілим, що полегшує роботу з програмою навіть для новачків.

Серед архітектурних переваг Kdenlive потрібно відзначити модульність, що дозволяє легко розширювати програму та додавати новий функціонал. Модульність гарантується використанням MLT, і також використання Qt сприяє створенню ефективного та зручного графічного інтерфейсу, що підвищує загальну зручність використання програми.

Наступним аналогом став Vegas Pro — це професійний NLE-відеоредактор, розроблений компанією MAGIX Software. Спочатку випущений у 1999 році компанією Sonic Foundry, а пізніше придбаний Sony Creative Software, Vegas Pro завоював популярність серед професіоналів у сфері відеомонтажу та створення контенту. Програма написана на C++ і використовує власний рушій для обробки мультимедіа, що забезпечує високу продуктивність і гнучкість [11], [12].

Відеоредактор пропонує широкий набір інструментів для редагування відео, включаючи багатоканальну аудіообробку, підтримку 4K-відео, вдосконалені ефекти і переходи, а також можливості для кольорокорекції та стабілізації зображення. Цей відеоредактор широко використовується у професійному середовищі для створення фільмів, телепередач, музичних відео та інших видів мультимедійного контенту. Vegas Pro відомий своєю інтуїтивно зрозумілою часовою шкалою, що дозволяє користувачам швидко і легко редагувати відео. Завдяки регулярним оновленням і підтримці нових

технологій, Vegas Pro залишається актуальним інструментом для відеомонтажу, який обирають як професіонали, так і ентузіасти.

На рис. 1.2 зображено інтерфейс відеоредактора.

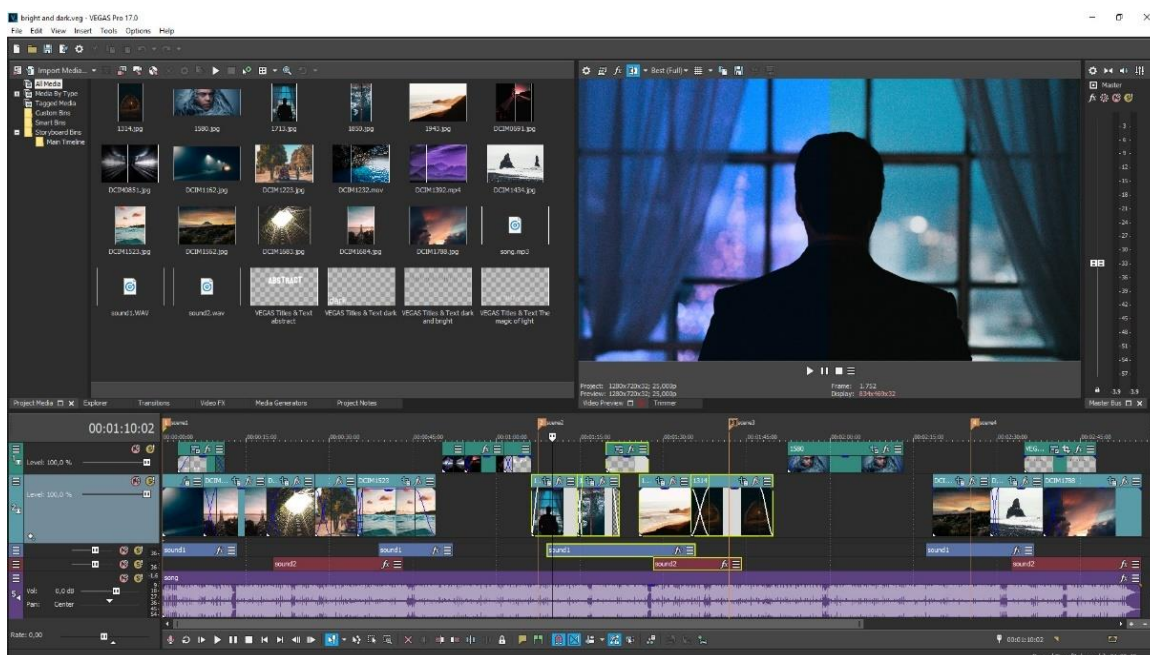


Рис. 1.2 – Vegas Pro (Інтерфейс відеоредактора)

Даний відеоредактор вирізняється широким функціональним арсеналом, що задовольнить навіть найвибагливіших користувачів. Він пропонує потужні можливості для колірної корекції, редагування аудіо, мультитрекового монтажу (створення водночас десятки і більше аудіо, відеодоріжок) та створення спеціальних графічних ефектів. Завдяки мультиплатформенній підтримці, цей редактор доступний на різних операційних системах, що робить його зручним для багатьох користувачів.

Інтерфейс користувача розроблений так, щоб бути максимально зручним та інтуїтивно зрозумілим, що дозволяє швидко освоїти програму навіть новачкам. Підтримка великої кількості відео та аудіо форматів робить цей інструмент універсальним для будь-яких потреб у відеомонтажі. Однією з головних переваг є професійні інструменти для колірної корекції та градієнтів, які за своїми можливостями не поступаються відомому DaVinci Resolve.

Програма також має велику бібліотеку ефектів, і її можливості можна значно розширити за допомогою сторонніх плагінів, таких як Sapphire, BCC, MBL, RSMB, Mocha та інших. Висока швидкість і стабільність роботи

забезпечуються завдяки ефективній оптимізації ресурсів, що робить цей відеоредактор надійним вибором для професійного використання.

Серед архітектурних переваг варто відзначити підтримку GPU-прискорення, що значно підвищує швидкість обробки відео та графіки завдяки використанню потужності графічного процесора. Модульна структура програми дозволяє легко додавати нові функції та розширювати можливості редактора.

На основі отриманих даних побудуємо порівняльну таблицю.

Таблиця 1.1 Порівняльна таблиця відеоредакторів Vegas Pro та Kdenlive

Характеристика	Vegas Pro	Kdenlive
<i>Операційні системи</i>	Windows	Windows, Linux, macOS
<i>Функціональність</i>	Професійний рівень, широкий функціонал	Призначений для широкого спектру користувачів, базовий та деякі продвинуті функції
<i>Інтерфейс користувача</i>	Інтуїтивно зрозумілий, професійний	Легкий у використанні, інтуїтивний, призначений для різних рівнів користувачів
<i>Підтримка форматів</i>	Різноманітна, велика кількість форматів відео та аудіо	Значна кількість підтримуваних форматів, але менше ніж у Vegas Pro
<i>Ціна та ліцензування</i>	Комерційний продукт, платна ліцензія	Вільне програмне забезпечення, безкоштовний доступ та відкритий код
<i>Спрямованість</i>	Професіональний відеомонтаж, більше підходить для великих та складних проектів	Редагування відео для різних користувачів, починаючи від новачків до більш досвідчених редакторів

Продовження таблиці 1.1

Характеристика	Vegas Pro	Kdenlive
<i>Інтеграція з іншими програмами</i>	Є інтеграція з іншими продуктами Sony та стандартами відеопромисловості	Зручна інтеграція з іншими програмами та бібліотеками, але менше партнерств та підтримки стандартів
<i>Підтримка GPU-прискорення</i>	Підтримка GPU для прискорення обчислень та оптимізації	Здатність використовувати потужність графічного процесора для високошвидкісної обробки відео

Створення порівняльної таблиці після опису таких відеоредакторів, як Kdenlive і Vegas Pro, є важливим кроком для надання чіткого і зручного порівняння їхніх функціональних можливостей. Така таблиця допомагає користувачам швидко побачити, які особливості пропонує кожен відеоредактор, що дозволяє визначити, який з них найкраще відповідає їхнім потребам. Вона надає структуровану інформацію про різні аспекти програм, такі як підтримка форматів, можливості редагування, сумісність з операційними системами, та інші важливі характеристики.

Порівняльна таблиця також допомагає підкреслити унікальні переваги та можливі недоліки кожного редактора, що є корисним для прийняття обґрунтованого рішення. Це особливо актуально для користувачів, які можуть не мати часу або технічних знань для детального аналізу текстових описів. Таким чином, порівняльна таблиця є ефективним інструментом для спрощення процесу вибору між різними програмними продуктами, забезпечуючи наочність і легкість сприйняття інформації.

1.3 Вибір і обґрунтування стеку розробки прототипу відеоредактора

Вибір стеку для написання відеоредактора є критично важливим, особливо враховуючи, що оптимізація продуктивності та ресурсозбереження є однією з ключових проблем, як згадувалося раніше у розділі. Для досягнення високої ефективності та гнучкості, було вирішено зупинитися на використанні мови програмування C++, яка забезпечує високий рівень контролю над апаратними ресурсами та ефективну роботу з мультимедіа-даними.

Інтеграція бібліотеки OpenCV надає потужні інструменти для обробки зображень та відео, що є необхідним для реалізації складних відеоефектів та маніпуляцій.

Використання FFmpeg як основи для кодування, декодування та трансляції мультимедіа гарантує підтримку широкого спектра форматів і високу продуктивність.

У висновку, стек, що використовуватиметься для побудови прототипу відеоредактора включає у себе C++, OpenCV, FFmpeg [13], [14], [15].

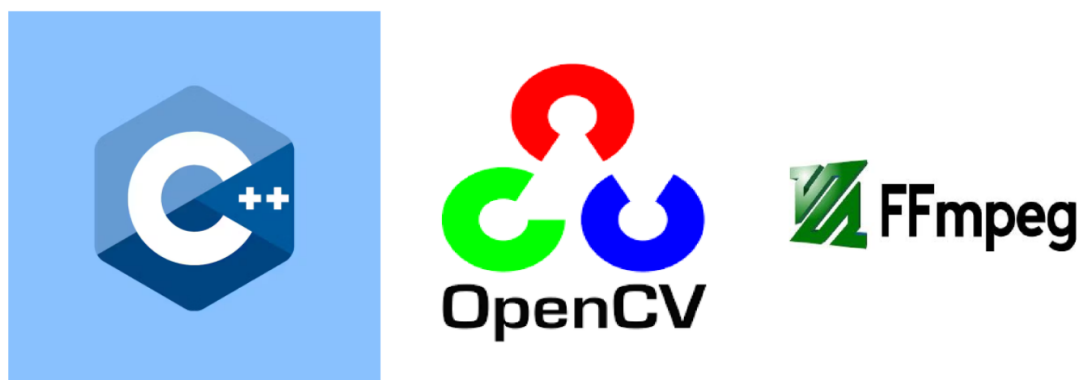


Рис. 1.3 – Вибір стеку для розробки прототипу ПЗ

Перейдемо до обґрунтування того, чому це поєднання є гарним рішенням для створення відеоредактора. По порядку, мова програмування C++:

1. Використання мови програмування C++ у створенні відеоредактора

Вибір мови C++ для розробки відеоредактора обумовлений кількома важливими аспектами, зокрема потребою в максимальній продуктивності та

контролі над апаратними ресурсами. C++ є мовою низького рівня, що дозволяє ефективно працювати з пам'яттю та оптимізувати роботу програм, що є критично важливим для складних операцій обробки відео та аудіо, які потребують великих обчислювальних ресурсів.

Завдяки можливості ручного керування пам'яттю, C++ дає змогу зменшити навантаження на систему та покращити швидкість обробки великих відеофайлів. Крім того, бібліотеки, такі як OpenCV та FFmpeg, були обрані через свою сумісність і високу ефективність при роботі з C++.

2. Використання бібліотеки OpenCV у створенні відеоредактора

OpenCV (англ. Open Source Computer Vision Library) — це потужна бібліотека з відкритим кодом для комп'ютерного зору та обробки зображень і відео. Вона надає набір інструментів для вирішення різноманітних задач у цих сферах і активно використовується в багатьох галузях, від науки та медицини до розваг і робототехніки. Важливою особливістю OpenCV є його багаторівнева архітектура, яка дозволяє розробникам використовувати різні модулі бібліотеки для досягнення бажаних результатів, забезпечуючи високу продуктивність та гнучкість у роботі. Розберемо модульну структуру бібліотеки на рис. 1.4.

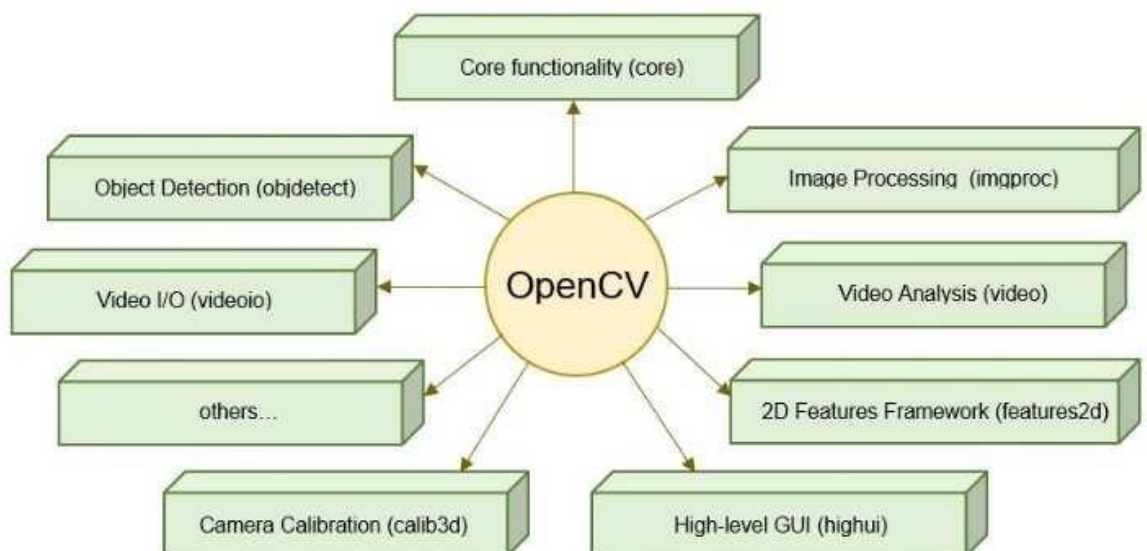


Рис. 1.4 – Модульна структура бібліотеки OpenCV

Архітектура OpenCV побудована таким чином, щоб забезпечити зручність у використанні та можливість масштабування. Вона включає декілька основних модулів, кожен з яких відповідає за конкретну задачу.

Наприклад, модуль core містить базові функції для роботи з матрицями (які є основною структурою даних в OpenCV), обробки зображень і виконання математичних операцій. Це основа бібліотеки, на якій базуються всі інші модулі [16].

Модуль imgproc відповідає за обробку зображень: фільтрацію, зміну розміру, контрастність, обертання, вирізання, корекцію кольору та багато іншого. Ці функції є надзвичайно корисними для створення складних відео та графічних ефектів [17].

Ще одним важливим модулем є video, який забезпечує можливість роботи з відеофайлами та виконання операцій у реальному часі. Завдяки цьому модулю можна реалізувати такі функції, як детекція руху, стабілізація відео та відстеження об'єктів у відеопотоці. Це відкриває можливості для розробки відеоредакторів, здатних обробляти та аналізувати відео в реальному часі, що є критично важливим для створення інтерактивних програм і систем безпеки [18].

Модуль highgui відповідає за графічний інтерфейс, забезпечуючи виведення зображень та відео на екран, а також взаємодію з користувачем. Він дозволяє створювати прості вікна для демонстрації результатів обробки, а також інтегрувати інтерфейси для управління програмами. Це дає змогу швидко розробляти інструменти для візуалізації оброблених даних [19].

Останні три модулі по значимості це features2d, спеціалізується на виявленні та порівнянні ключових точок на зображеннях. Це надзвичайно корисно для завдань, таких як розпізнавання облич, пошук об'єктів на зображеннях або відео, а також для реалізації функцій стабілізації зображення. Модулі calib3d і stitching використовуються для калібрування камер і створення панорам із кількох зображень [20 – 22].

Варіативність модульної структури по-справжньому вражає, у наступних розділах буде більше детальної інформації, що стосується аналізу функціональних можливостей модулів у реальних практичних задачах.

3. Використання набору інструментів FFmpeg у створенні відеоредактора

FFmpeg — це багатofункціональна бібліотека для обробки мультимедійних даних, яка включає в себе різноманітні компоненти для роботи з відео, аудіо та іншими мультимедійними потоками. На рис. 1.5 наведено інформацію про основні компоненти архітектури FFmpeg:

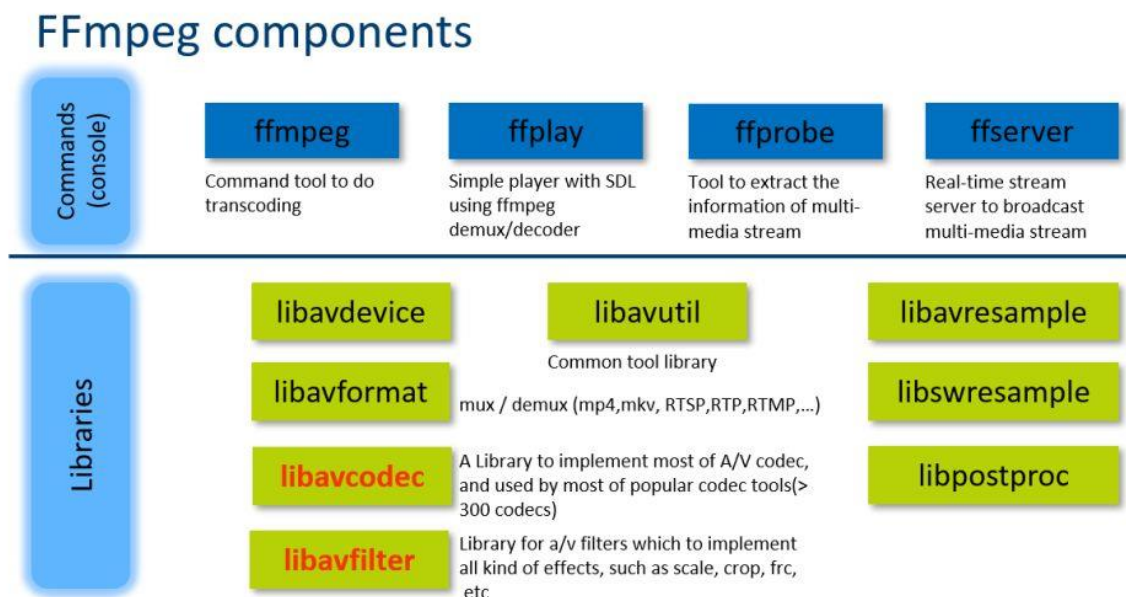


Рис. 1.5 - FFmpeg-компоненти

Компонентна архітектура є ключовою для створення відеоредактора завдяки своїй гнучкості та спеціалізації в окремих функціях. Кожен компонент виконує свою специфічну задачу, що дозволяє розробникам ефективніше працювати з різними аспектами обробки відео та аудіо. Перейдемо до огляду компонентної частини FFmpeg.

libavformat є одним із основних компонентів, який забезпечує можливість читання та запису даних у різних форматах. Це робить відеоредактор більш універсальним і сумісним із різними мультимедійними файлами, що важливо для роботи з різноманітними джерелами контенту [23].

libavcodec відповідає за кодування та декодування відео і аудіо. Його основною перевагою є підтримка популярних стандартів, таких як H.264 та AAC, що є критично важливим для обробки сучасного відеоконтенту. Це дозволяє забезпечити високу якість відтворення та збереження медіа [24].

libavfilter надає можливості для фільтрації та обробки відео і аудіо. Це важливо для реалізації різних ефектів, корекції кольорів і розмиття, а також для швидкої обробки даних, що підвищує загальну продуктивність редактора [25].

libavutil містить допоміжні функції, які спрощують взаємодію між іншими компонентами. Використання цього модуля дозволяє організувати код проекту, що полегшує його підтримку та розвиток [26].

libswscale та libswresample є важливими для обробки різних форматів і роздільностей відео та аудіо. libswscale використовується для масштабування та конвертації кольорів відеофреймів, тоді як libswresample відповідає за перетворення та масштабування аудіосемплів (англ. Audio sample – це відрізок аудіоінформації, вирізаний або записаний з якого-небудь наявного джерела) . Це забезпечує редактору можливість ефективно працювати з різноманітними джерелами мультимедіа [27 – 28].

libavdevice надає можливості для взаємодії з різними пристроями введення і виведення, такими як веб-камери або аудіоінтерфейси. Це розширює функціонал відеоредактора, дозволяючи інтегрувати його з апаратним забезпеченням [29].

Загалом, ці компоненти формують потужний стек технологій, який є функціонально підходящим для створення власного прототипу відеоредактора. Розробники отримують важливі знання про принципи роботи з відеопроцесінгом і можуть реалізувати свої ідеї в практичному проекті.

Аналіз можливостей модулів OpenCV та FFmpeg, а якщо детальніше, функціональні рішення, є однією з головних завдань обраної проблематики кваліфікаційної роботи, що буде описано у розділі 1.5.

1.4 Вибір і обґрунтування інструментальних засобів розробки прототипу відеоредактора

Для вибору інструментальних засобів розробки прототипу відеоредактора було проведено аналіз доступних рішень на ринку, зважаючи на специфічні вимоги проекту. Основними критеріями вибору стали функціональність, продуктивність, зручність використання та підтримка сучасних технологій.

Для розробки користувацького інтерфейсу обрано Qt Creator, оскільки ця платформа пропонує потужний набір інструментів для створення кросплатформених додатків з багатими інтерфейсами.

Qt Creator забезпечує швидку розробку завдяки інтегрованим візуальним редакторам та великій кількості готових компонентів. Крім того, Qt підтримує широкий спектр платформ, що робить його ідеальним для створення відеоредактора, який може працювати на різних операційних системах [30].

Основним середовищем розробки (IDE) було обрано Microsoft Visual Studio. Це рішення базується на його високій продуктивності, багатому наборі інструментів для налагодження та тестування, а також інтеграції з системами контролю версій, такими як Git. Visual Studio надає розширені можливості для розробки на різних мовах програмування, зокрема C++, що є важливим для розробки високопродуктивного програмного забезпечення, такого як відеоредактор [31].

Використання Visual Studio в поєднанні з Qt Creator дозволяє досягти оптимального балансу між швидкістю розробки, якістю коду та функціональністю кінцевого продукту.



Рис. 1.6 – Інструментальні засоби для написання та моделювання користувацького інтерфейсу прототипу відеоредактора

1.5 Аналіз функціонально-оптимізаційних методів при роботі з відео

Цей розділ висвітлює одну з ключових проблем теми даної кваліфікаційної роботи. Зважаючи на обраний стек технологій та функціональні можливості розроблюваного прототипу, методи, що використовуються, охоплюють більшість аспектів відеопроцесінгу. Наведений нижче аналіз охоплює широкий спектр функцій відеоредактора, від обробки зображень до пояснення принципів потокового зчитування відео. Багато методів базуються на бібліотеці OpenCV, яка є частиною стеку розробки прототипу відеоредактора.

Функція "Зміна розміру вихідного файлу" у відеоредакторі дозволяє користувачам змінювати розміри відео під час редагування. Ця функція є незамінною для адаптації відео до різних платформ та пристроїв. Користувачі можуть легко налаштовувати ширину та висоту відео, зберігаючи або змінюючи пропорції за потреби. Швидкодія цієї функції є критично важливою для створення якісного прототипу, тому далі розглянемо готове рішення.

Використання функцій OpenCV *pyrUp()* і *pyrDown()* дозволяє зменшити або збільшити роздільну здатність певного зображення. Зазвичай нам потрібно перетворити зображення на розмір, відмінний від оригінального. Для цього можливі два варіанти: збільшити розмір зображення або зменшити його. Хоча в OpenCV існує функція геометричної трансформації, яка буквально змінює розмір зображення, у цьому розділі ми спочатку проаналізуємо використання пірамід зображень, які широко застосовуються у величезному діапазоні програм по роботі з відео. Що таке піраміда зображень?

Піраміда зображень — це набір зображень, які утворюються з одного оригінального зображення, а дискретизація послідовно знижується, доки не буде досягнуто бажаної точки зупинки. Існує два поширених типи пірамід зображень: піраміда Гауса, що використовується для зменшення дискретизації зображень, та піраміда Лапласа, яка використовується для реконструкції зображення з підвищеною дискретизацією із зображення, розташованого нижче

в піраміді (з меншою роздільною здатністю). Перейдемо до теоретичного підґрунтя реалізації.

Уявляємо піраміду як набір шарів, у яких чим вищий шар, тим менший розмір. Графічне представлення на рис. 1.7.

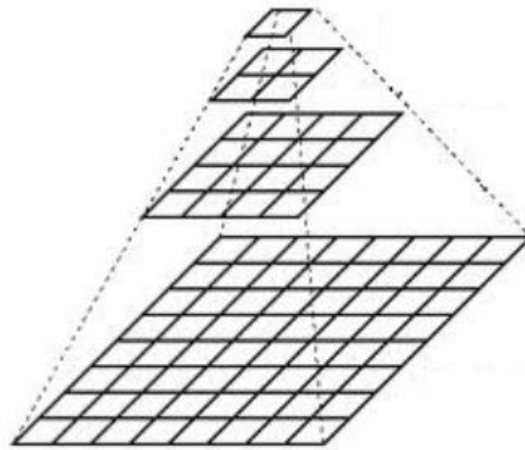


Рис. 1.7 – Представлення піраміди як набір шарів

Кожен шар нумерується знизу вгору, тому шар $(i + 1)$ (позначається як G_{i+1}) менший за шар $I(G_i)$. Щоб створити шар $(i + 1)$ у піраміді Гауса, ми робимо згортання з ядром Гауса:

$$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Рис. 1.8 – Згортання з ядром Гауса

Далі видаляємо усі парні рядки та стовпці.

Легко помітити, що отримане зображення буде рівно на чверть площі свого попередника. Ітерація цього процесу на вхідному зображенні (оригінальне зображення) створює всю піраміду. Наведена вище процедура була корисною для зменшення дискретизації зображення.

Що, якщо ми хочемо зробити його більшим? Стовпці, заповнені нулями (0). По-перше, збільшуємо розмір зображення у пропорції двічі до оригіналу в

кожному вимірі, додавши нові парні рядки а далі виконуємо згортку з тим самим ядром, що показано вище (помноженим на 4), щоб наблизити значення “відсутніх пікселів”. Прикладне демонстрування виконання ефекту на рис. 1.9.



Рис. 1.9 – Виконання методу *pyrDown()*

Ці дві процедури (зменшення та збільшення, як пояснено вище) реалізуються функціями OpenCV *pyrUp()* і *pyrDown()*, що буде продемонстровано у реальному прототипі відеоредактора [31].

Наступний метод, що активно використовуватиметься у аналізі та розробці це лінійне об’єднання зображень. Чому воно важливе при роботі з відео та як це реалізувати на практиці за допомогою методу OpenCV *addWeighted()*?

Метод *addWeighted()* в OpenCV використовується для змішування двох зображень або відео з заданими ваговими коефіцієнтами. Це основний інструмент для створення ефектів плавного переходу (англ. cross dissolve), де одне зображення поступово зникає, а інше з’являється. Основна формула для методу *addWeighted()* виглядає наступним чином:

$$g(x) = \alpha \cdot f_0(x) + \beta \cdot f_1(x) + \gamma, \text{ де:}$$

- $g(x)$ — результат зображення,
- $f_0(x)$ і $f_1(x)$ — два вхідних зображення,
- α і β — вагові коефіцієнти для кожного зображення,
- γ — додаткова константа, яка додається до результату (опціонально).

Цей метод дозволяє комбінувати два зображення за допомогою вагових коефіцієнтів α і β , які визначають вплив кожного зображення на кінцевий результат. Змінюючи значення α і β , можна керувати прозорістю та інтенсивністю кожного зображення у кінцевому результаті.

У практиці метод *addWeighted()* часто використовується для створення ефектів змішування (рис. 1.10), таких як плавний перехід між двома кадрами відео або зображеннями в слайд-шоу. Це досягається шляхом поступової зміни вагових коефіцієнтів α і β з 0 до 1 (або навпаки), як було сказано раніше, що призводить до поступового зникнення одного зображення та появи іншого.



Рис. 1.10 – Виконання методу *addWeighted()*

Наступний метод, який ми розглянемо, стосується корекції кольору зображень. Процес переведення зображення з інших кольірних форматів, таких як RGB, CMYK або HSV, у відтінки сірого називається градацією сірого (англ. Grayscale). Це перетворення дозволяє зображенню варіюватися від чисто чорного до чисто білого.

Відтінки сірого мають кілька важливих переваг. По-перше, вони зменшують обсяг даних, оскільки в зображеннях RGB є три кольорові канали, а в градаціях сірого — лише один. Це спрощує обробку інформації. По-друге, зменшення складності моделі дозволяє нейронним мережам легше працювати із

зображеннями. Наприклад, для зображень RGB розміром 10 на 10 пікселів потрібно 300 вхідних вузлів, тоді як для зображень у відтінках сірого - лише 100.

Функціональність цього методу можна продемонструвати за допомогою функції `cv2.cvtColor()` (рис. 1.11), яка буде описана детальніше у практичній частині нашої роботи. Ще існують інші інтерпретації цієї функціональності, але у прототипі відеоредактора використовувалося саме це, тому зупинимося на ньому.



Рис. 1.11 - Виконання методу `cv2.cvtColor()`

Наступний метод буде присвячений функціоналу “під коробкою”, а саме можливість перемотування відео до зазначеного кадру. Доволі звична дія при роботі з редактором, але саме у таких речах і відчутна оптимізація та швидкодія системи, особливо якщо вихідний файл значної довжини та якості.

Метод `Seek()` служить для швидкого переходу до конкретного місця (кадру) у відео. Цей інструмент є важливим для випадкового доступу, редагування та навігації в медіафайлах. Принцип дії методу полягає в установці поточної позиції для читання або відтворення у відеофайлі. Це дозволяє миттєво перейти до необхідного кадру без потреби переглядати всі попередні кадри послідовно.

У OpenCV для роботи з відео використовується клас `VideoCapture`, який містить метод `set()`. Цей метод дозволяє змінювати різні властивості відео, включаючи позицію кадру. Метод `set()` приймає два параметри: `propId` та `value`. `PropId` є ідентифікатором властивості, яку потрібно змінити; для переходу до конкретного кадру використовується `cv2.CAP_PROP_POS_FRAMES`. `Value`, в

свою чергу, є новим значенням для цієї властивості, а для `cv2.CAP_PROP_POS_FRAMES` це номер кадру, до якого потрібно переміститися.

Практична реалізація функціоналу буде представлена у наступному розділі на прототипі, що розроблятиметься.

Останній метод, який ми розглянемо, стосується обрізання відеокадрів — поширеної задачі для користувачів. Наприклад, є фрагмент відео, в якому водяна мітка заважає сприйняттю матеріалу і її потрібно або обрізати або замалювати. Для вирішення цієї проблеми був розроблений метод *clip()*, що виконає перший функціонал.

Метод *clip()* використовується для обрізання частини зображення або відеокадру. Це дуже корисний інструмент для виділення конкретних областей, які можуть бути важливими для подальшої обробки або аналізу. Принцип його роботи полягає в тому, що ви визначаєте прямокутну область у зображенні або відеокадрі і зберігаєте лише цю частину, видаляючи все, що знаходиться за межами. Це особливо корисно для фокусування на певних об'єктах чи ділянках зображення, а також для зменшення розміру зображення перед обробкою або створення спеціальних ефектів.

У OpenCV метод *clip()* реалізується через зрізи масиву (англ. array slicing), що дозволяє легко виділити потрібну область зображення або кадру. Наприклад, можна вказати координати верхнього лівого і нижнього правого кута прямокутника, щоб отримати бажану частину відео. Це робить процес обрізання простим і швидким, що є важливим аспектом при редагуванні відео та підготовці матеріалів для подальшого використання. Практична реалізація цього функціоналу буде у розділі 2.

Висновки до розділу 1

У цьому розділі спочатку було сформульовано задачу, окреслено основні проблеми, що розглядаються в темі кваліфікаційної роботи, а також опи-

сано функціональні можливості прототипу відеоредактора, який буде розроблятися. Далі проведено аналіз та детальний розгляд існуючих аналогів програмного забезпечення, зокрема відеоредакторів Kdenlive та Sony Vegas Pro.

Окремо було розглянуто стек технологій для розробки прототипу програмного забезпечення, а також інструменти для написання коду, включаючи програмне забезпечення для створення графічного інтерфейсу.

Особливу увагу було приділено одній з основних задач кваліфікаційної роботи — аналізу методів оптимізації при створенні відеоредактора. Розглянуто функціональні методи для роботи з відео, зокрема зміна розміру, обрізання частини відео, корекція кольору, об'єднання відеофрагментів, реалізація слайдера для прокручування відеокадрів.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ НЕЛІНІЙНОГО ВІДЕОРЕДАКТОРА

2.1 Визначення варіантів використання відеоредактора та об'єктно-орієнтованої структури

У цьому розділі проведено детальний аналіз вимог до програмного забезпечення з особливою увагою на виявлення функціональних та нефункціональних вимог, важливих для покращення користувацького досвіду при роботі з нелінійним відеоредактором. Зокрема, розглянуто вимоги до часових характеристик реакції додатку на дії користувача та описано варіанти використання програми. Для забезпечення ясності структури було додано узагальнену діаграму елементів системи (сутностей), їх структуру та методи з детальним описом.

Аналіз вимог включає вивчення потреб та очікувань користувачів щодо функціональності та характеристик відеоредактора. Основна мета аналізу полягає у виявленні необхідних функцій, які мають бути реалізовані в програмному забезпеченні для задоволення потреб користувачів. Шляхом інтерв'ю з користувачами, опитувань та зворотного зв'язку були визначені функціональні вимоги, що окреслюють конкретні особливості та можливості, які має мати програмний продукт. Крім того, були зібрані нефункціональні вимоги, що стосуються таких аспектів, як продуктивність, стабільність роботи та зручність використання [33].

Функціональні вимоги відеоредактора, що розроблюється:

- імпортування відео різних форматів;
- зрозумілий користувацький інтерфейс, починаючи від слайдерів для відтворення відео до функціональних кнопок “Старт” / “Імпорт”, “Пауза” та інше.
- базове редагування відео: зміна контрасту, яскравості, трансформування по позиції;
- можливість зміни кольорокорекції;

- зміна розміру вихідного відео за допомогою методу Resize або більш технічна імплементація за рахунок піраміди Лапласа та Гауса;
- обрізання різних частин відео, так звані Zoom-операції;
- додавання фотографій поверх відео з можливостями трансформування першого об'єкту;
- накладання відео на відео, так звані Blend та Merge-операції;
- робота з аудіо та можливість розділення аудіо та відео-доріжок.

Нефункціональні вимоги:

- система повинна швидко реагувати на дії користувача, забезпечуючи безперебійну роботу;
- відеоредактор повинен ефективно використовувати пам'ять та процесор, щоб забезпечити швидку та безперебійну роботу, навіть з великими відео;
- підтримка різних форматів медіафайлів;
- гарантування безпеки та конфіденційності збережених відеофайлів;
- відеоредактор повинен мати зрозумілий та легкий у використанні графічний інтерфейс.

Часові характеристики реакції системи:

- швидкий час відгуку забезпечує миттєву відповідь на дії користувача, що поліпшує загальний досвід роботи з редактором
- редактор повинен ефективно завантажувати великі файли відео та зберігати їх після редагування; оптимізація цих процесів допомагає забезпечити швидку обробку та збереження відеофайлів;
- деякі операції, такі як фільтри, ефекти або робота з зображеннями поверх відео, можуть займати багато часу, особливо при роботі з

великими відеофайлами, тому швидкість виконання цих операцій впливає на плавність та продуктивність роботи з редактором;

- швидке відкриття програми дозволяє користувачеві швидко почати працювати, а швидке закриття забезпечує збереження часу та енергії користувача.

Важливо враховувати, що часові характеристики реакції відеоредактора можуть залежати від різних факторів, таких як обладнання комп'ютера, розмір та якість відеофайлу, а також оптимізація самої програми.

За допомогою тестування продуктивності визначаються прийнятні часові межі для різних дій та процесів у системі. Ці часові характеристики є важливими для задоволення очікувань користувачів та уникнення розчарування, спричиненого затримками або відсутністю реакції.

Перейдемо до варіантів використання системи. Для задоволення різноманітних потреб користувачів система пропонує кілька варіантів використання. Завдяки цій гнучкості, користувачі можуть налаштовувати програмний продукт відповідно до своїх конкретних вимог і робочих процесів, що підвищує загальний досвід і продуктивність.

Діаграма варіантів використання нелінійного відеоредактора демонструє основні функціональні можливості редактора та взаємодію користувача з системою. Головним актором на діаграмі є користувач редактора, таким як артист, дизайнер чи фотограф, який взаємодіє з системою через різні сценарії використання [34].

Створена діаграма варіантів використання (рис. 2.1) – це візуальне представлення взаємодії між користувачем системи (актором) і самою системою та специфікацією прецедентів.

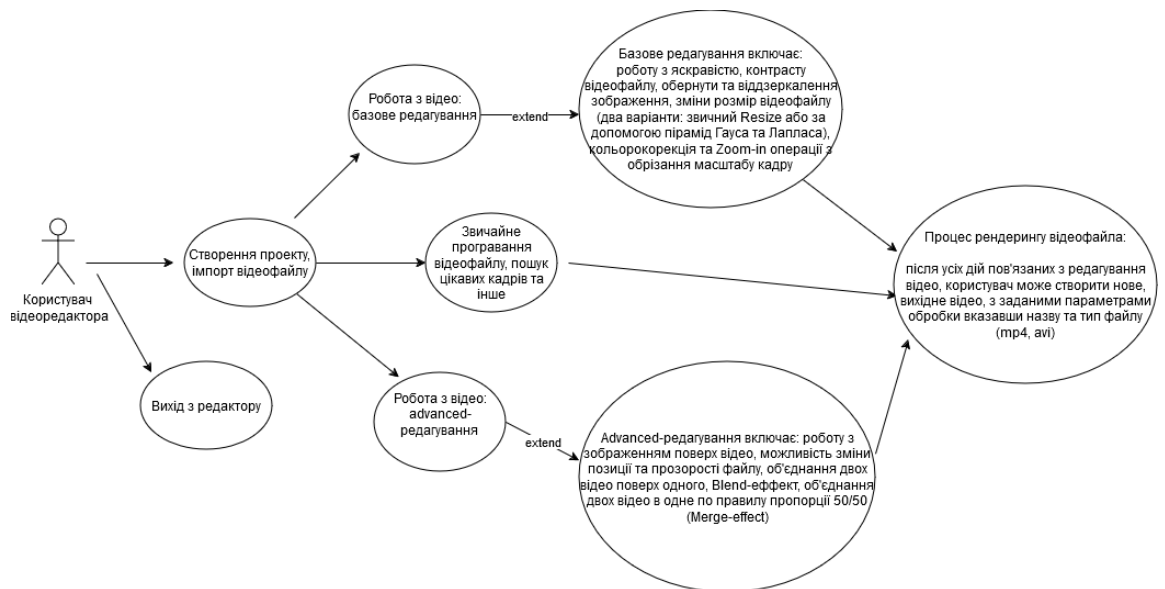


Рис. 2.1 – Діаграма варіантів використання

Визначені функціональні можливості для користувача утворюють комплексне рішення, яке задовольняє різноманітні графічні потреби.

Загалом, діаграма варіантів використання надає повне та наочне представлення функціональних можливостей. Визначаючи та документуючи варіанти використання, діаграма стає цінним інструментом для аналізу системи, збору вимог і проектування.

Наступною складовою цього розділу є опис об'єктно-орієнтованої структури програмного забезпечення. Структура забезпечує модульний дизайн системи, де різні компоненти виступають як об'єкти з визначеними атрибутами та методами. Інкапсуляція функціональності в об'єкти сприяє повторному використанню, модульності та розширюваності системи [34].

Наведено узагальнену діаграму об'єктно-орієнтованої структури системи, що показує взаємозв'язки та взаємодію між різними об'єктами (рис. 2.2). Кожна сутність представляє конкретний компонент або модуль програмного продукту зі своїми атрибутами та методами. Діаграма надає візуальне уявлення про архітектуру системи, полегшуючи розуміння її загальної структури.

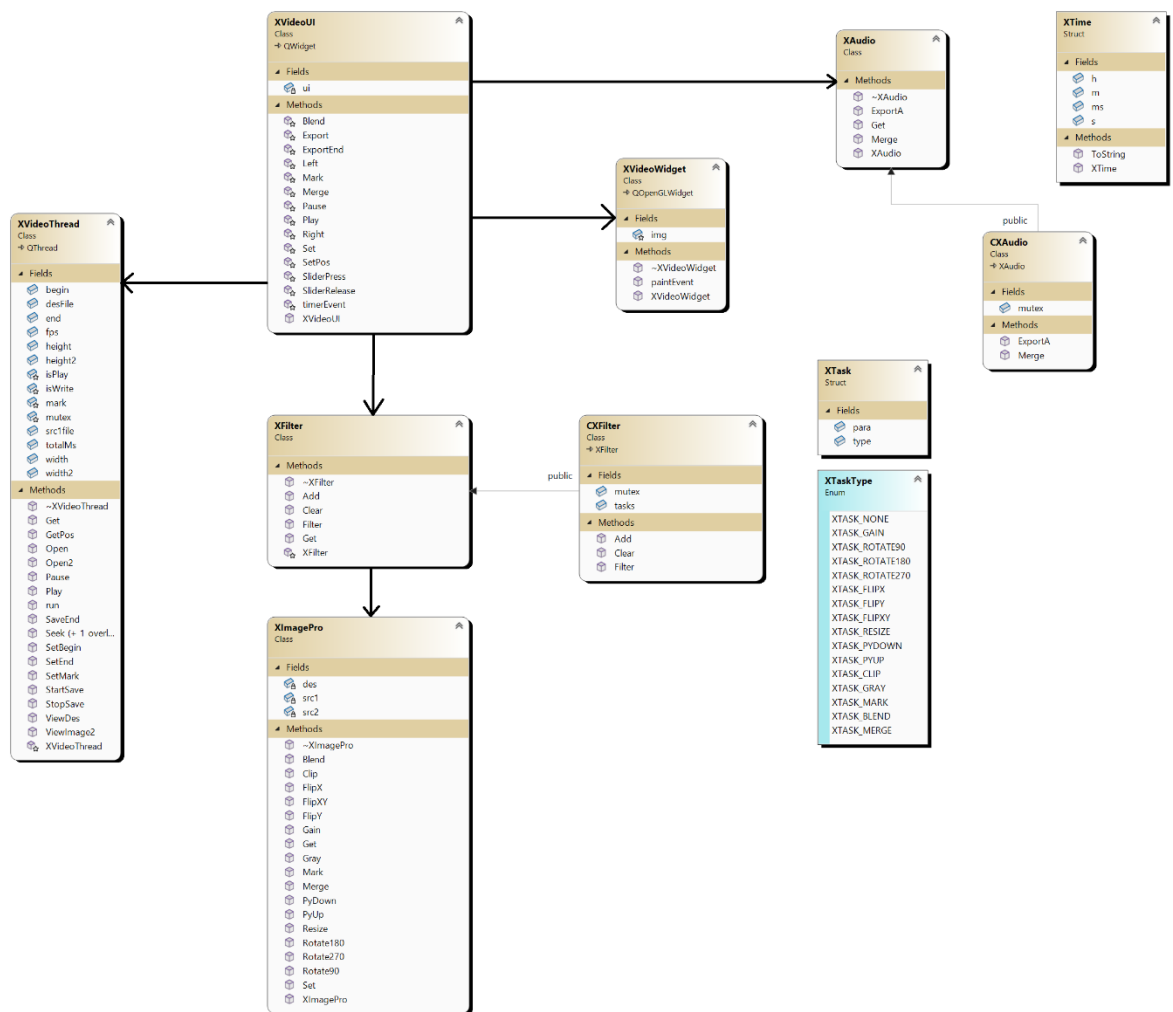


Рис. 2.2 – Діаграма класів прототипу відеоредактора

У діаграмі варто виділити класи, які було спроектовано, використовуючи шаблон проектування “Фабрика”.

Фабричний метод – це породжуючий шаблон проектування, що дозволяє підкласам створювати екземпляри певного класу. Під час створення підкласи можуть визначити, який саме клас буде створено. Іншими словами, цей шаблон передає право створення об’єктів базовому класу. Це дозволяє використовувати в коді не конкретні класи, а маніпулювати абстрактними об’єктами на більш високому рівні.

Огляд сутностей системи. Породжуючим класом є *XVideoUI*, що відповідає за графічний інтерфейс користувача для роботи з відео. Він забезпечує відображення відео, управління відтворенням, паузою, експортом та обробкою відео. Клас використовує бібліотеку Qt для створення діалогових

вікон, повідомлень і таймерів, а також OpenCV для обробки зображень та відео.

Основні функції класу *XVideoUI* включають підключення сигналів та слотів для відображення зображень, обробку подій інтерфейсу, налаштування фільтрів та параметрів відео, а також забезпечення експортованого відео. Клас також дозволяє виконувати операції з відео, такі як обрізка, змішування, маркування та об'єднання відео.

Клас *XVideoThread* відповідає за обробку відео в окремому потоці, використовуючи бібліотеку OpenCV. Він дозволяє відкривати два відеофайли, синхронізувати їх відтворення, застосовувати фільтри та зберігати оброблені відео у новий файл.

Основні методи класу включають *StartSave()*, який ініціалізує збереження відео у файл, встановлюючи розміри відео, кольорову палітру та частоту кадрів (англ. FPS – Frames per second); *StopSave()*, що зупиняє процес збереження; *SetBegin()* та *SetEnd()*, які визначають початкові та кінцеві кадри для відтворення; *Seek()*, що дозволяє перемотувати відео до вказаного кадру; а також *Open()* і *Open2()*, які відкривають перше та друге відео відповідно. Метод *run()* є основним циклом потоку, відповідальним за обробку відео: він зчитує кадри, застосовує фільтри та записує результати у файл.

Крім того, клас містить механізм синхронізації доступу до спільних ресурсів через м'ютекси, що допомагає уникнути проблем із конкурентним доступом під час обробки відео.

Клас *CXFilter* є похідним від абстрактного класу *XFilter* і реалізує методи для обробки зображень. Цей клас зберігає список завдань, які потрібно виконати над зображенням, і забезпечує синхронізацію доступу до цих завдань за допомогою м'ютекса.

Основні методи класу є метод *Filter()* — для обробки зображень, який використовує список завдань та обробляє два зображення за допомогою об'єкта *XImagePro*. Завдання можуть включати різні операції, такі як корекція яскравості та контрасту, обертання, відбиття, зміна розміру, перетворення в

чорно-біле зображення тощо; Декілька допоміжних методів, такі як *Add()* — додає нове завдання до списку завдань; *Clear()* — очищає список завдань.

Клас також реалізує патерн проектування Singleton через метод *Get()*, який повертає єдиний екземпляр класу *CXFilter*. Всі операції з обробки зображень виконуються в межах цього екземпляра, що дозволяє централізовано управляти завданнями та їх виконанням.

М'ютекс використовується для синхронізації доступу до списку завдань і забезпечує безпечну роботу з даними у багатопоточному середовищі.

Клас ***XImagePro*** забезпечує набір методів для обробки зображень за допомогою бібліотеки OpenCV. Основна мета класу — це виконання різноманітних операцій над зображеннями, таких як обертання, відбиття, змішування зображень, а також застосування корекцій яскравості та контрасту.

Основними методами класу є *Set()* — метод для ініціалізації двох зображень, які будуть оброблятися. Перше зображення копіюється у змінну *des*, яка використовується для збереження результату обробки; *Rotate90()*, *Rotate180()*, *Rotate270()* — методи для обертання зображення на 90, 180 та 270 градусів відповідно; *FlipX()*, *FlipY()*, *FlipXY()* — методи для відбиття зображення по осях X, Y та обох осях одночасно; *Merge()* — об'єднує два зображення в одне, розташовуючи їх поруч (side-by-side); *Blend()* — змішує два зображення з заданим коефіцієнтом прозорості *a*; *Mark()* — додає мітку на зображення, накладаючи інше зображення з певною прозорістю; *Resize()* — змінює розмір зображення; *PyDown()*, *PyUp()* — методи для зменшення та збільшення розміру зображення за допомогою пірамідальних методів; *Gray()* — перетворює зображення в чорно-біле (градації сірого); *Clip()* — обрізає зображення на зазначеній області; *Gain()* — регулює яскравість та контрастність зображення;

Конструктор класу ініціалізує об'єкти, а деструктор очищає ресурси, у свою чергу клас використовує OpenCV для реалізації усіх основних функцій,

таких як обертання, змішування, корекція кольору тощо, і спрямований на виконання цих операцій безпосередньо над зображеннями.

Клас **XVideoWidget** реалізує віджет для відображення відео або зображень у Qt, використовуючи OpenCV для обробки зображень. Основні функції цього класу включають відображення зображень на віджеті та масштабування зображень для підгонки до розміру віджету.

Метод *SetImage(cv::Mat mat)* — цей метод дозволяє встановлювати зображення, передане як об'єкт *cv::Mat* (формат OpenCV). Зображення масштабується під розмір віджету, і якщо зображення кольорове, воно конвертується з формату BGR у RGB. Якщо зображення чорно-біле, його формат змінюється на *QImage::Format_Grayscale8*. Потім дані зображення копіюються у буфер для відображення; *paintEvent(QPaintEvent* e)* — цей метод викликається при малюванні на віджеті. Він створює об'єкт *QPainter*, ініціалізує його для поточного віджету, а потім малює зображення на віджеті, використовуючи метод *drawImage*;

Клас **XAudio** реалізує функції для роботи з аудіо, використовуючи бібліотеку *ffmpeg* для витягування звуку з відеофайлів та об'єднання аудіо з відео. Клас *SXAudio* є нащадком абстрактного класу *XAudio* і реалізує його методи. Основними складовими цього класу є структура *XTime* та сам клас *SXAudio*. Структура *XTime* призначена для перетворення часу з мілісекунд у формат години:хвилини:секунди.мілісекунди. Конструктор цієї структури приймає час у мілісекундах і розбиває його на години, хвилини, секунди та мілісекунди, а метод *ToString* форматує цей час у рядок у вигляді "h:m:s.ms".

Клас *SXAudio* містить кілька важливих елементів, зокрема м'ютекс (*QMutex*), який забезпечує синхронізацію доступу до команд, що виконуються через *ffmpeg*, запобігаючи одночасному виконанню кількох команд, що може викликати помилки. Метод *ExportA* відповідає за експорт аудіо з відеофайлу в окремий аудіофайл, формуючи команду для *ffmpeg*, яка витягує аудіо з відео, де параметри *beginMs* та *outMs* визначають час початку та тривалість експорту в мілісекундах. Ця команда виконується через системний виклик

system(cmd.c_str()) з блокуванням м'ютекса для забезпечення коректності виконання.

Ще один важливий метод *Merge* об'єднує аудіо та відео в один файл, формуючи команду для *ffmpeg*, щоб без перекодування об'єднати відеофайл і аудіофайл, використовуючи опцію *-c copy*. Цю команду також виконує системний виклик *system(cmd.c_str())* з блокуванням м'ютекса. Метод *Get* реалізує патерн *Singleton*, повертаючи єдиний екземпляр класу *CXAudio*. Конструктор та деструктор класу *XAudio* не містять додаткової логіки, оскільки основні операції виконуються в методах класу *CXAudio*.

Загальна логіка класу *CXAudio* дозволяє ефективно виконувати операції з аудіо та відео через командний інтерфейс *FFmpeg*. Всі команди виконуються по черзі з використанням м'ютекса для забезпечення синхронізації, що допомагає уникнути проблем із паралельним доступом до команд або ресурсів під час обробки аудіо та відео.

2.2 Проектування та реалізація алгоритмів роботи системи

Розробка та реалізація системних алгоритмів відповідають за виконання різних обчислювальних завдань, обробку даних і прийняття рішень в системі. Було створено декілька діаграм активності для візуалізації та представлення потоку діяльності в системі. Вони пропонують чіткий спосіб зображення послідовних і паралельних дій, які відбуваються під час виконання системних процесів. Створюючи діаграми активності, було за мету забезпечити всебічне розуміння алгоритмічного дизайну системи та її реалізації [36].

Діаграма активності для *створення нового проекту*: ця діаграма показує послідовність дій користувача, таких як імпорт відеофайлу до редактора, що в подальшому перейде у створення нового об'єкту системи та його ініціалізацію.



Рис. 2.3 – Діаграма активності для створення нового проекту

На основі діаграми активності для створення нового проекту було створено наступну таблицю специфікації [36]:

Таблиця 2.1

Ім'я варіанту використання: створення нового проекту		Рівень: високий
Основний актор: користувач		
Короткий опис:		Створення робочого полотна для редагування відео
Передумови:		Запустити відеоредактор
Відносини:		
	Включення:	Ініціалізація
Основний потік:		Користувач може створити нове робоче полотно у поточному вікні успішно імпортуючи відеофайл
Альтернативні/виключні потоки:		Повідомлення помилки

Діаграма активності для *редагування відео* (рис. 2.4): Ця діаграма може показати послідовність дій користувача для редагування відеофайлу.

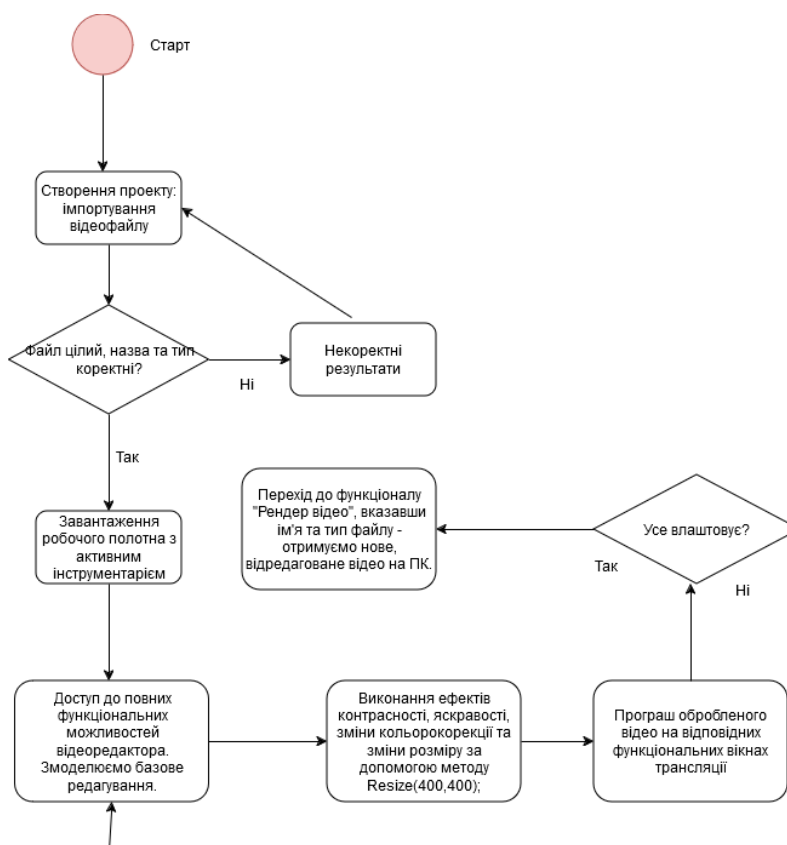


Рис. 2.4 – Діаграма активності для процесу редагування відеофайлу

На основі діаграми активності для редагування відеофайлу було створено наступну таблицю специфікації:

Таблиця 2.2

Ім'я варіанту використання: редагування відеофайлу		Рівень: високий
Основний актор: користувач		
Короткий опис:		Процес редагування відео матеріалу
Передумови:		Запустити відеоредактор, створити робочий проект
Відносини:		
	Включення:	Ініціалізація
Основний потік:		Користувач має створити нове робоче полотно у поточному вікні, імпортувавши відеофайл. Подальший процес редагування з використанням інструментарію редактора.
Альтернативні/виключні потоки:		Повідомлення помилки

Наступна діаграма активності буде представлена для можливості відеоредактора *розділяти відео та аудіо-файли*, якщо аудіо присутнє у початковому відео, що імпортувалося до додатку.

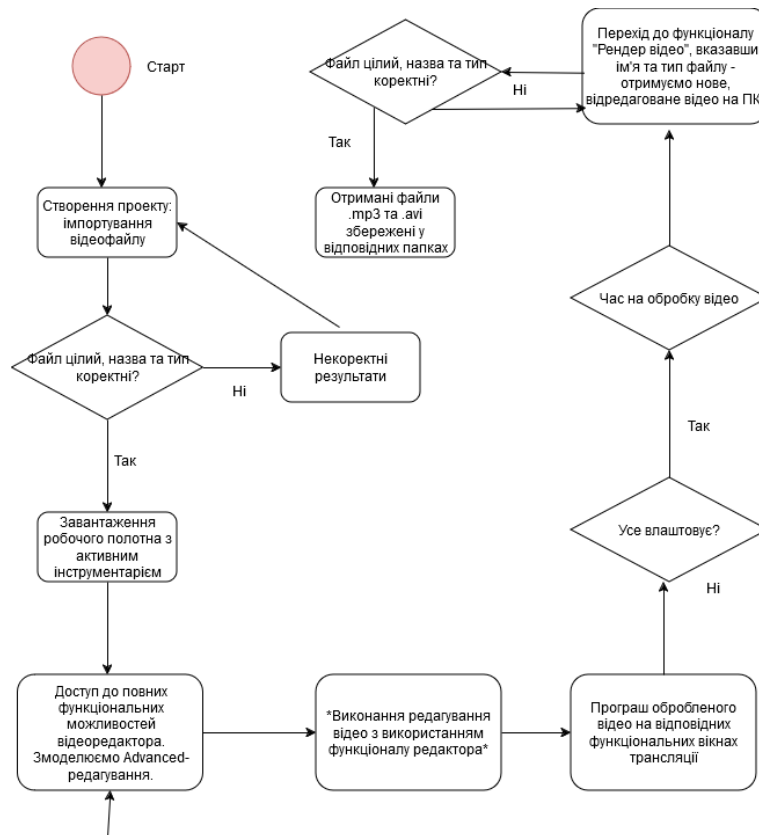


Рис. 2.5 - Діаграма активності для процесу редагування відеофайлу з аудіо

На основі діаграми активності для редагування відеофайлу з аудіо було створено наступну таблицю специфікації:

Таблиця 2.3

Ім'я варіанту використання: редагування відеофайлу		Рівень: високий
Основний актор: користувач		
Короткий опис:		Процес редагування відео матеріалу з вихідним аудіо
Передумови:		Запустити відеоредактор, створити робочий проект
Відносини:		
	Включення:	Ініціалізація
Основний потік:		Користувач має створити нове робоче полотно у поточному вікні, імпортувавши відеофайл з аудіо. Подальший процес редагування з використанням інструментарію редактора.
Альтернативні/виключні потоки:		Повідомлення помилки

Ці діаграми активності та таблиці специфікації допоможуть уявити та візуалізувати послідовність дій та взаємодію користувача з відеоредактором.

2.3 Реалізація оптимізаційних методів при створенні нелінійного відеоредактора

У цьому розділі буде обговорено етапи процесу впровадження оптимізаційних методів та висвітлено ключові аспекти розробки нелінійного відеоредактора, що було згадано раніше, у розділі 1.5.

При виборі стеку розробки було зазначено, що пріоритетною причиною вибору C++ - це можливість оптимізації графічних перетворень та доступ до пам'яті [37].

Використання функцій OpenCV *pyrUp()* і *pyrDown()*, щоб зменшити або збільшити роздільну здатність певного зображення, аналіз цих методів був проведений у минулому розділі, зараз практична реалізація та пояснення кодової частини.

```
void XImagePro::PyDown(int count){
    if (des.empty()) return;
    for (int i = 0; i < count; i++)
        pyrDown(des, des);
}
```

У коді відбувається зменшення розміру зображення за допомогою пірамідального зменшення у класі *XImagePro*. Функція *PyDown* приймає один параметр *count*, який вказує, скільки разів потрібно зменшити розмір зображення. Спочатку вона перевіряє, чи є зображення (змінна *des*) порожнім, і якщо так, функція завершується. Далі, у циклі, що повторюється *count* разів, викликається функція *pyrDown*, яка зменшує розмір зображення на один рівень піраміди. Функція *pyrDown* приймає вхідне та вихідне зображення як параметри, у цьому випадку — це одна й та ж змінна *des*, тобто вихідне зображення замінює вхідне. Кожен виклик *pyrDown* зменшує розмір зображення вдвічі за шириною та висотою. Таким чином, якщо параметр *count* дорівнює 3, зображення буде зменшене вдвічі тричі, тобто в 8 разів від початкового розміру.

```
void XImagePro::PyUp(int count){
    if (des.empty()) return;
    for (int i = 0; i < count; i++)
        pyrUp(des, des);
}
```

Цей код виконує збільшення розміру зображення за допомогою пірамідального збільшення у класі *XImagePro*. Функція *PyUp* приймає параметр *count*, що вказує, скільки разів потрібно збільшити розмір зображення. Якщо зображення (змінна *des*) порожнє, функція завершується. У циклі, який повторюється *count* разів, викликається функція *pyrUp*, яка збільшує розмір зображення на один рівень піраміди. Кожен виклик *pyrUp* збільшує розмір зображення вдвічі за шириною та висотою.

Наступний метод, що активно використовуватиметься у аналізі та розробці це лінійне об'єднання зображень. Чому воно важливе при роботі з відео та як це реалізувати на практиці за допомогою методу OpenCV *addWeighted()*?

```
void XImagePro::Merge(){
    if (des.empty()) return;
    if (src2.empty()) return;
    if (src2.size() != des.size())
    {
        int w = src2.cols * ((double)src2.rows / (double)des.rows);
        resize(src2, src2, Size(w, des.rows));
    }

    int dw = des.cols + src2.cols;
    int dh = des.rows;
    des = Mat(Size(dw, dh), src1.type());
    Mat r1 = des(Rect(0, 0, src1.cols, dh));
    Mat r2 = des(Rect(src1.cols, 0, src2.cols, dh));

    src1.copyTo(r1);
    src2.copyTo(r2);
}
```

Цей код виконує об'єднання двох зображень (side-by-side) у класі *XImagePro*. Функція *Merge* спочатку перевіряє, чи є перше (*des*) або друге (*src2*) зображення порожнім, і завершується, якщо так. Далі перевіряється, чи мають зображення однакові розміри. Якщо розміри не співпадають, змінюється розмір другого зображення (*src2*), щоб його висота відповідала висоті

першого зображення (*des*). Потім обчислюється ширина (*dw*) та висота (*dh*) об'єднаного зображення. Створюється нове зображення *des* з відповідними розмірами. Після цього визначаються регіони для першого (*r1*) та другого (*r2*) зображень у новому об'єднаному зображенні. Нарешті, обидва зображення копіюються у відповідні регіони нового зображення, створюючи єдине зображення, де два вихідні зображення розміщені поруч.

Наступний метод виконує схожий функціонал, а саме змішування двох зображень по пропорції.

```
void XImagePro::Blend(double a){
    if (des.empty()) return;
    if (src2.empty()) return;
    if (src2.size() != des.size())
    {
        resize(src2, src2, des.size());
    }
    addWeighted(src2, a, des, 1 - a, 0, des);
}
```

Цей код виконує змішування двох зображень за допомогою коефіцієнта *alpha* у класі *XImagePro*. Функція *Blend* спочатку перевіряє, чи є перше (*des*) або друге (*src2*) зображення порожнім, і завершується, якщо так. Далі перевіряється, чи мають зображення однакові розміри. Якщо розміри не співпадають, змінюється розмір другого зображення (*src2*), щоб воно відповідало розміру першого зображення (*des*). Потім використовується функція *addWeighted*, яка змішує два зображення за допомогою коефіцієнта *a* (*alpha*). Результат змішування зберігається у першому зображенні (*des*). Коефіцієнт *a* визначає вагу другого зображення (*src2*), а *1 - a* визначає вагу першого зображення (*des*).

Наступний метод буде пов'язаний з кольорокорекцією зображення.

```
void XImagePro::Gray(){
    if (des.empty()) return;
    cvtColor(des, des, COLOR_BGR2GRAY);
}
```

Цей код виконує перетворення зображення в чорно-біле (градації сірого) у класі *XImagePro*. Функція *Gray* спочатку перевіряє, чи є зображення (*des*) порожнім, і якщо так, завершується. Якщо зображення не порожнє, викликається функція *cvtColor*, яка перетворює кольорове зображення у відтінки

сірого, використовуючи параметр *COLOR_BGR2GRAY*. Результат зберігається у тій же змінній *des*, замінюючи початкове кольорове зображення на чорно-біле.

Наступний метод буде присвячений функціоналу “під коробкою”, а саме можливість перемотування відео до зазначеного кадру. Доволі звична дія при роботі з редактором, але саме у таких речах і відчутна оптимізація та швидкодія системи, особливо якщо вихідний файл значної довжини та якості.

```
bool XVideoThread::Seek(double pos){
    double count = cap1.get(CAP_PROP_FRAME_COUNT);
    int frame = pos * count;
    return Seek(frame);
}
```

Метод *Seek(double pos)* приймає позицію *pos* у відсотках від загальної довжини відео (значення від 0 до 1).

Спочатку він отримує загальну кількість кадрів у відео (*count*) за допомогою *cap1.get(CAP_PROP_FRAME_COUNT)*, потім обчислює номер кадру, відповідний зазначеній позиції (*frame = pos * count*), і викликає метод *Seek(int frame)* з цим номером кадру.

```
bool XVideoThread::Seek(int frame){
    mutex.lock();
    if (!cap1.isOpened())
    {
        mutex.unlock();
        return false;
    }

    int re = cap1.set(CAP_PROP_POS_FRAMES, frame);
    if (cap2.isOpened())
        cap2.set(CAP_PROP_POS_FRAMES, frame);
    mutex.unlock();
    return re;
}
```

Цей код включає два методи класу *XVideoThread* для пошуку та перемотування відео. Метод *Seek(int frame)* приймає номер кадру *frame*, захоплює блокування *mutex* і перевіряє, чи відкрите відео (*cap1.isOpened()*). Якщо відео не відкрите, метод повертає *false*. Інакше він встановлює поточну позицію кадру для першого відео (*cap1.set(CAP_PROP_POS_FRAMES, frame)*) і, якщо

друге відео також відкрите, для другого відео (`cap2.set(CAP_PROP_POS_FRAMES, frame)`). Після цього блокування знімається, і метод повертає результат операції `re`.

Наступний метод буде пов'язаний з обрізанням відеокадру, типова ситуація для користувача. Є фрагмент відео, водяна мітка, що заважає очам сприймати матеріал, для цього був вигаданий метод `clip()`.

```
void XImagePro::Clip(int x, int y, int w, int h){
    if (des.empty()) return;
    if (x < 0 || y < 0 || w <= 0 || h <= 0) return;
    if (x > des.cols || y > des.rows) return;
    des = des(Rect(x, y, w, h));
}
```

Цей код виконує обрізку зображення на зазначеній області у класі `XImagePro`. Функція `Clip` приймає координати верхнього лівого кута області (`x, y`) та її ширину (`w`) і висоту (`h`). Спочатку перевіряється, чи є зображення (`des`) порожнім, і якщо так, функція завершується. Далі перевіряється валідність координат: вони мають бути не меншими за нуль, а ширина та висота — більше нуля. Також перевіряється, щоб координати не виходили за межі зображення. Якщо всі перевірки пройдені, виконується обрізка зображення за допомогою функції `Rect`, яка визначає прямокутну область для обрізки. Результат зберігається у змінній `des`, замінюючи початкове зображення обрізаним.

Наступним розберемо метод для початку збереження відео в файл.

```
bool XVideoThread::StartSave(const std::string filename, int width,
int height, bool isColor){
    cout << "Start export" << endl;
    Seek(begin);
    mutex.lock();

    if (!cap1.isOpened())
    {
        mutex.unlock();
        return false;
    }

    if (width <= 0)
        width = cap1.get(CAP_PROP_FRAME_WIDTH);
    if (height <= 0)
        height = cap1.get(CAP_PROP_FRAME_HEIGHT);
```

```

        vw.open(filename,
            VideoWriter::fourcc('X', '2', '6', '4'),
            this->fps,
            Size(width, height), isColor
        );

        if (!vw.isOpened())
        {
            mutex.unlock();
            cout << "start save failed!" << endl;
            return false;
        }

        this->isWrite = true;
        desFile = filename;
        mutex.unlock();
        return true;
    }
}

```

Цей код реалізує метод для початку збереження відео в файл у класі *XVideoThread*. Функція *StartSave* приймає параметри для імені файлу (*filename*), ширини та висоти відео (*width, height*), а також прапорець *isColor*, що вказує, чи має відео бути кольоровим. Спочатку метод переходить до початкового кадру відео за допомогою функції *Seek(begin)*, потім захоплює блокування *mutex* для синхронізації доступу до спільних ресурсів.

Далі перевіряється, чи відкрито відео джерело (*cap1*). Якщо ні, функція завершується з *false*. Якщо ширина або висота не задані, вони автоматично встановлюються за розмірами першого відео. Потім ініціалізується об'єкт *VideoWriter* для запису відео у файл, використовуючи кодек X264 та задану частоту кадрів (*fps*).

Якщо ініціалізація не вдалася, метод виводить повідомлення про помилку і повертає *false*. Якщо все успішно, встановлюється прапорець *isWrite*, запам'ятовується ім'я файлу, знімається блокування *mutex* і метод повертає *true*, що вказує на успішний початок збереження відео.

Останнім розберемо практичний метод встановлення початкового кадру для відтворення на нашому UI-слайдеру.

```
void XVideoThread::SetBegin(double p){  
    mutex.lock();  
    double count = cap1.get(CAP_PROP_FRAME_COUNT);  
    int frame = p * count;  
    begin = frame;  
    mutex.unlock();  
}
```

Цей код реалізує метод для встановлення початкового кадру для відтворення відео в класі *XVideoThread*. Функція *SetBegin* приймає параметр *p*, що вказує на відносну позицію початкового кадру у відео (значення від 0 до 1). Спочатку захоплюється блокування *mutex* для синхронізації доступу до спільних ресурсів. Потім обчислюється загальна кількість кадрів у відео за допомогою *cap1.get(CAP_PROP_FRAME_COUNT)* і визначається номер кадру, що відповідає відносній позиції *p*, множачи його на загальну кількість кадрів. Цей номер кадру зберігається в змінній *begin*. Після цього знімається блокування *mutex*, що дозволяє іншим потокам взаємодіяти з ресурсами.

Висновки до розділу 2

У цьому розділі ми заглибилися у розробку та реалізацію нелінійного відеоредактора. Першим ділом, було визначено варіанти використання та створено об'єктно-орієнтовану структуру відеоредактора. Шляхом ретельного аналізу було визначено ключові функціональні можливості, необхідні для того, щоб додаток задовольняв потреби різних користувачів.

Потім ми приступили до розробки та впровадження алгоритмів системи. Це передбачало розробку ефективних та дієвих алгоритмів для оптимізації графічних перетворень. Використовуючи відповідні алгоритми та найкращі практики, ми прагнули підвищити продуктивність системи та забезпечити надійну роботу.

РОЗДІЛ 3. ПОРЯДОК РОБОТИ ТА ТЕСТУВАННЯ НЕЛІНІЙНОГО ВІДЕОРЕДАКТОРА

3.1 Порядок роботи нелінійного відеоредактора

Перейдемо до огляду інтерфейсу та робочого процесу прототипу відеоредактора. У цьому розділі розглядатимуться різні знімки екрану та надаватимуться описи до кожного з них, підкреслюючи ключові особливості та функціональні можливості програмного продукту. Ці скріншоти демонструють елементи користувацького інтерфейсу, з якими користувачі взаємодіятимуть під час роботи.

Починаємо з відкриття відеоредактора на своєму персональному Windows-комп'ютері (рис. 3.1). При відкритті відеоредактора користувач зазвичай бачить інтерфейс програми, який дозволяє йому взаємодіяти з редактором і працювати з відеофайлами [39].

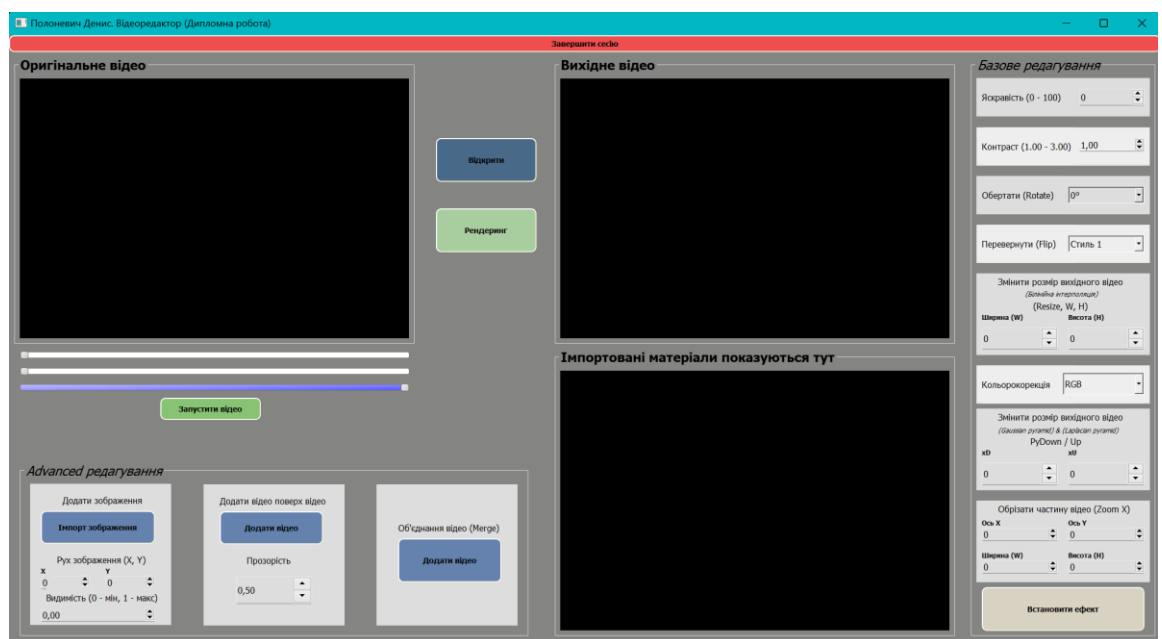


Рис. 3.1 – Інтерфейс прототипу відеоредактора

Основні елементи, які можуть бути присутніми на екрані включають у себе діалогові секції “Базове” та “Advanced” редагування, робочі кнопки “Відкрити відео”, “Старт / Пауза” та “Рендер”, функціональні слайдери для програвання відео та його зупинки з переходом на будь-який кадр, і на останок 3 OpenGL-віджети для відтворення оригінального та вихідного відео разом з медіа-файлами.

У наступних скріншотах демонструється основний функціонал відеоредактора. Почнемо з секції “Базове редагування”, але для початку імпортуємо відеофайл.

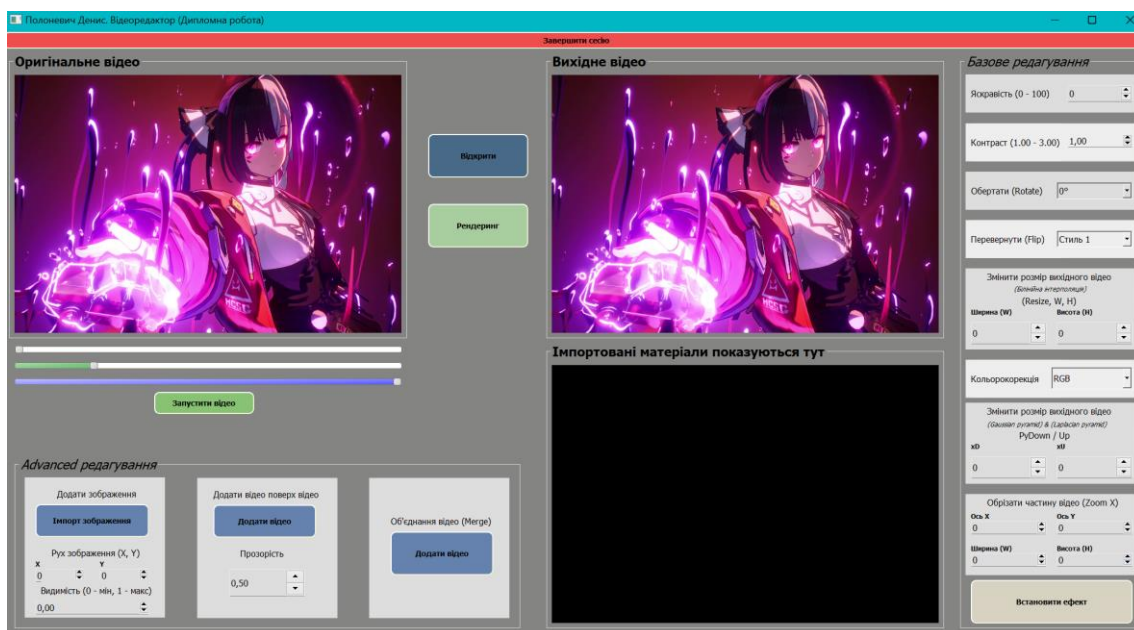


Рис. 3.2 – Успішно імпортований відеофайл

Після імпортування відеофайлу до редактора, користувач має змогу побачити оригінальний файл зліва, та вихідне відео для редагування зправа, саме у цьому вікні будуть показуватися дії по редагуванню.

Наступні скріншоти показують демонстрацію функціоналу бокової панелі “Базове редагування”, почнемо з ефекту “Зміна яскравості” (рис. 3.3).

Користувач має можливість вписати у відповідне поле значення від 0 до 100 включно, число по собі відповідає за інтенсивність, чим нижче – яскравість слабша, і навпаки. Для успішного прийняття ефекту після вводу варто натиснути на кнопку “Встановити ефект”, що містить функціонал сигнал – слот від Qt та успішно виконає дію.

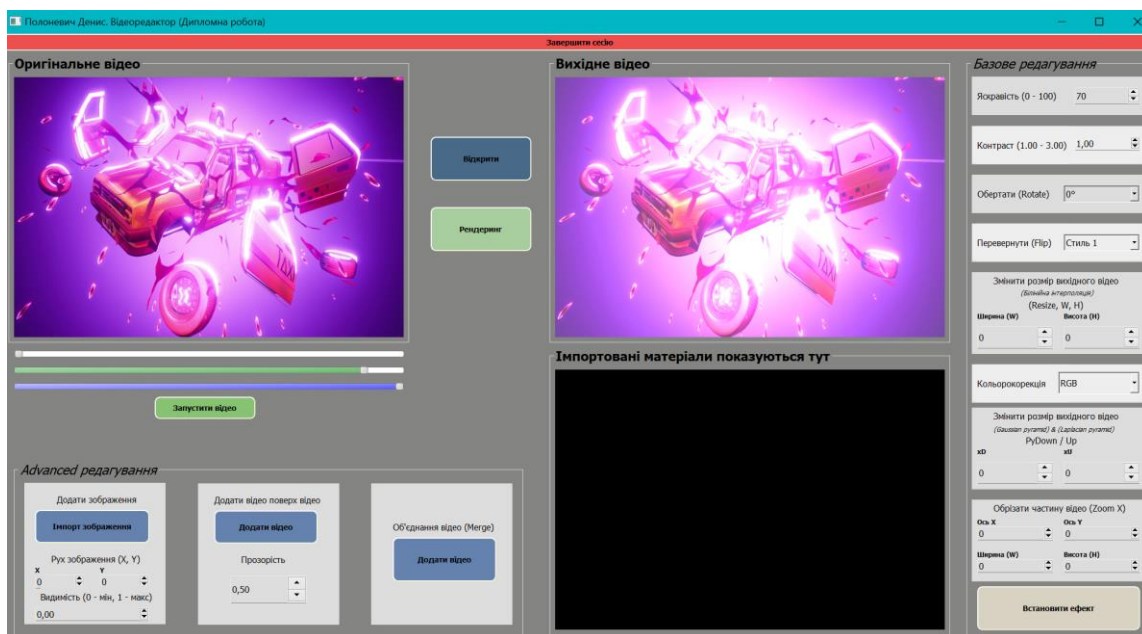


Рис. 3.3 – Функціонал “Зміна яскравості”

Ефект “Зміна контрасту” (рис 3.4). Користувач має можливість вписати у відповідне поле значення від 1 до 3 включно, число по собі відповідає за інтенсивність, чим нижче – контрастність слабша, і навпаки. Для успішного прийняття ефекту після вводу варто натиснути на кнопку “Встановити ефект”, що містить функціонал сигнал – слот від Qt та успішно виконає дію.

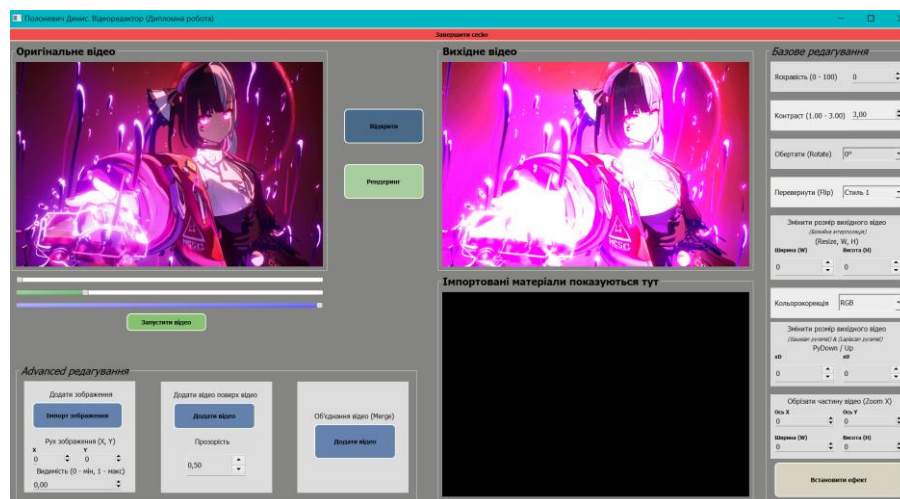


Рис. 3.4 – Функціонал “Контраст”

Ефект “Обернути відео” (рис 3.5). Користувач має можливість обрати один з чотирьох заготовлених пресетів по оберту відео, від 0 до 360 градусів. Для успішного прийняття ефекту після вводу варто натиснути на кнопку “Встановити ефект”, що містить функціонал сигнал – слот від Qt та успішно виконає дію.

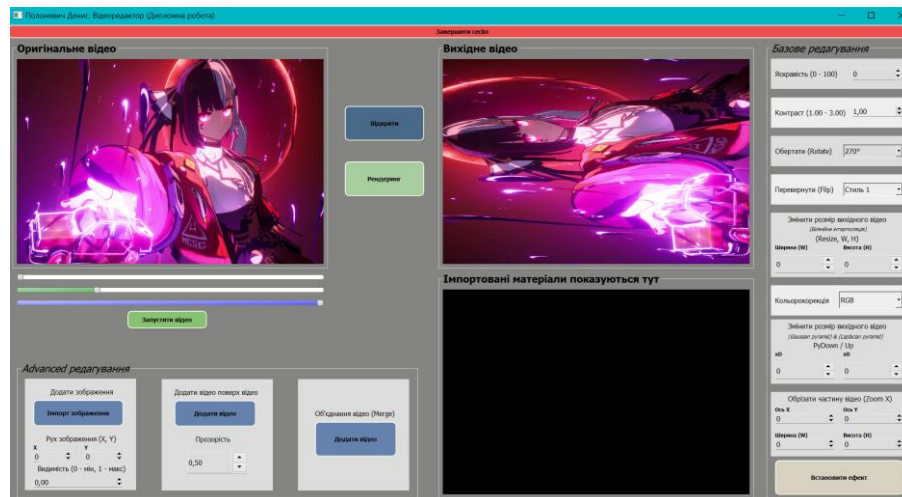


Рис. 3.5 – Функціонал “Обернути відео”

Ефект “Віддзеркалити відео” (рис 3.6). Користувач має можливість обрати один з чотирьох заготовлених пресетів по віддзеркаленню відео. Для успішного прийняття ефекту після вводу варто натиснути на кнопку “Встановити ефект”, що містить функціонал сигнал – слот від Qt та успішно виконає дію.

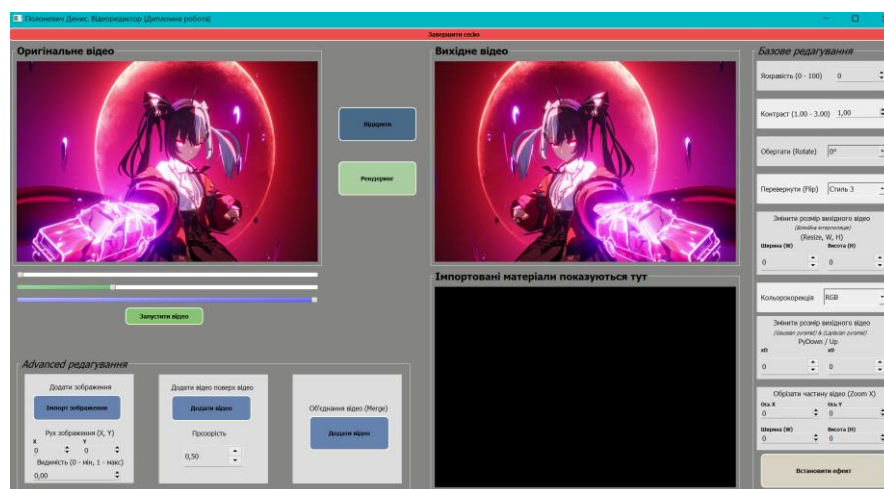


Рис. 3.6 – Функціонал “Віддзеркалити відео”

Ефект “Зміна кольорокорекції” (рис 3.7). Користувач має можливість змінити кольорокорекцію відео, з RGB до B&W, і навпаки (англ. RGB – RedGreenBlue канали, англ. B&W – BlackAndWhite канали). Для успішного прийняття ефекту після вводу варто натиснути на кнопку “Встановити ефект”, що містить функціонал сигнал – слот від Qt та успішно виконає дію.

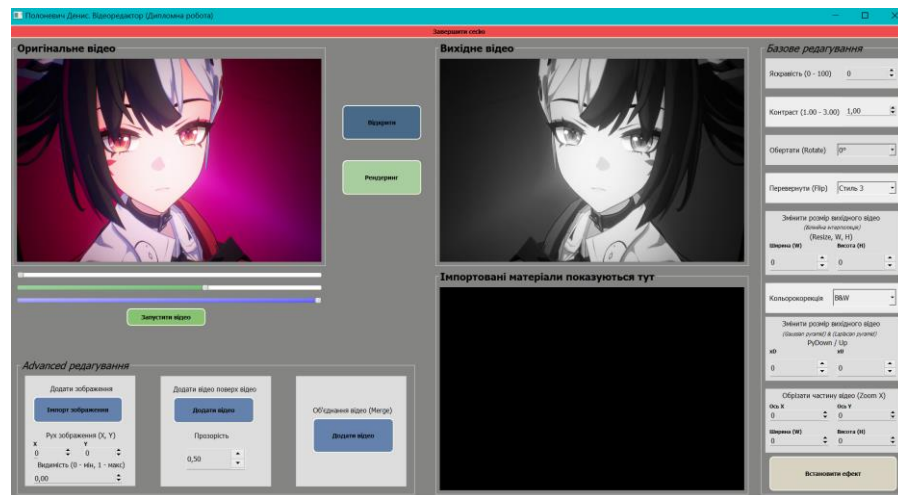


Рис. 3.7 – Функціонал “Кольорокорекція: чорно-білий ефект”

Ефект “Зміна вихідного розміру відео, використання методу Resize” (рис 3.8). Користувач має можливість змінити вихідний розмір відео за допомогою методу `Resize()` від бібліотеки `OpenCV`. Це одна з двох реалізацій по зміні вихідного розміру у редакторі, інша реалізація включає у себе піраміди Гауса та Лапласа відповідно. Для успішного прийняття ефекту після вводу варто натиснути на кнопку “Встановити ефект”, що містить функціонал сигнал – слот від `Qt` та успішно виконає дію.

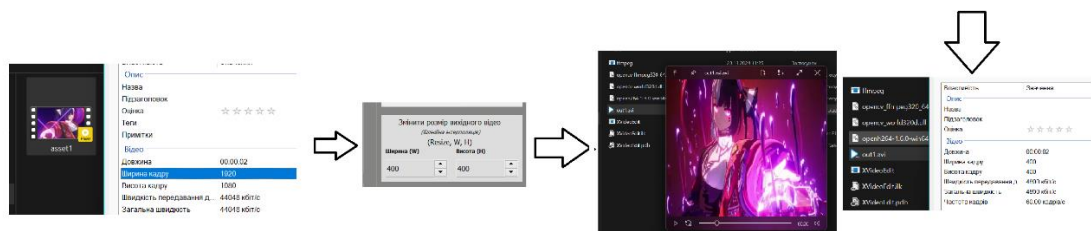


Рис. 3.8 – Функціонал “Зміна розміру”

Ефект “Зміна вихідного розміру відео, використання пірамідних зображень” (рис 3.9). Користувач має можливість змінити вихідний розмір відео за допомогою пірамідних зображень, кодова реалізація була продемонстрована у розділі 2.

Користувач може збільшити або зменшити роздільну здатність кратно, умовно кажучи, з відео 1920x1080 зробити погіршення в 2 рази, та отримати вихідне 960x540, так само і в кращу сторону, створити 4K відео.

Для успішного прийняття ефекту після вводу варто натиснути на кнопку “Встановити ефект”, що містить функціонал сигнал – слот від Qt та успішно виконає дію.

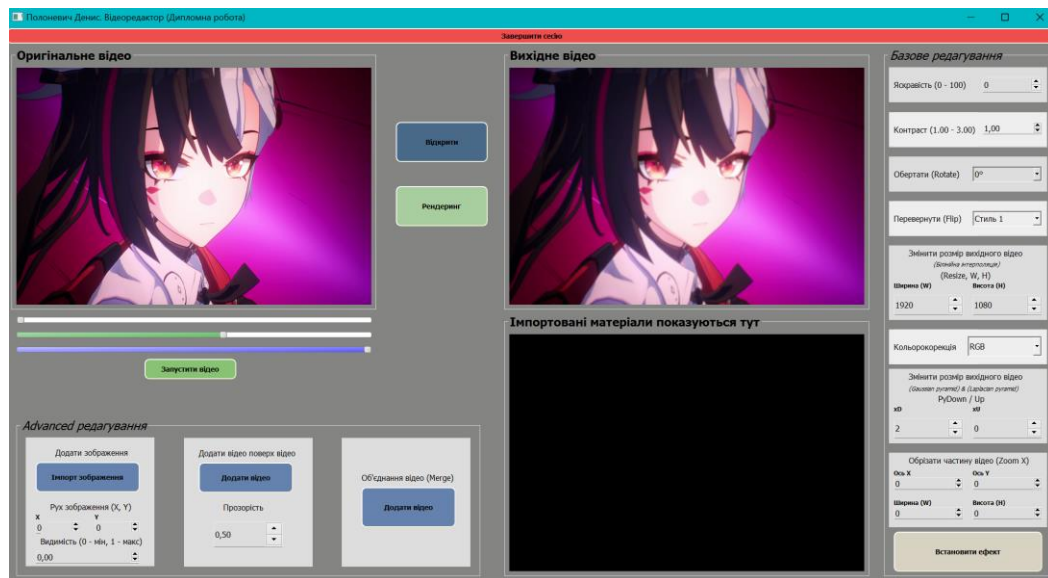


Рис. 3.9 – Функціонал “Зміна розміру: пірамідне вирішення”

Можна помітити невеличке розмиття на вихідному відео, це вказує на те, що ефект з погіршенням був успішно виконаний.

Ефект “Обрізати частину відео” (рис. 3.10) надає користувачеві можливість обрізати конкретну область відео для подальшої обробки. Для цього користувач може вказати координати прямокутної області у відповідних полях інтерфейсу. Цей функціонал дозволяє фокусуватися на важливих фрагментах відео, усуваючи зайві елементи, що можуть відволікати увагу глядача. Кодова реалізація цього ефекту була продемонстрована у розділі 2.

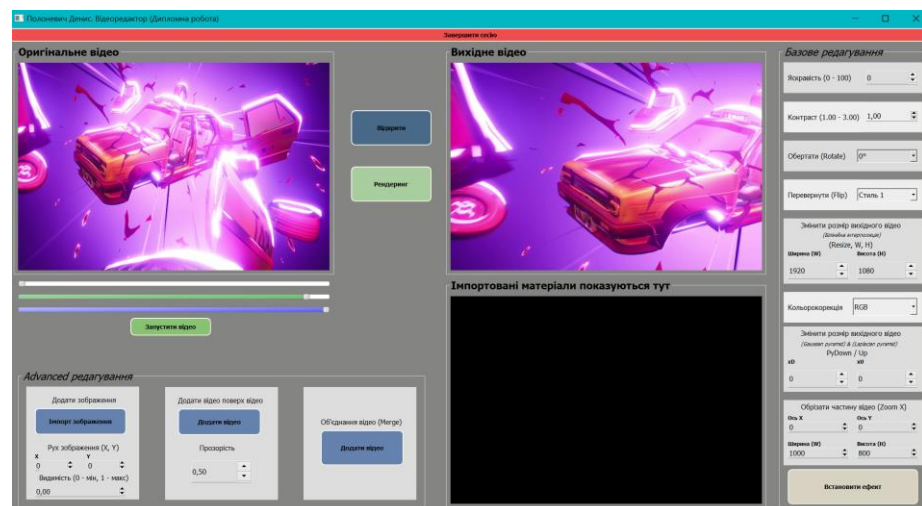


Рис. 3.10 – Функціонал “Обрізання частини відео”

Редактор також дає змогу користувачеві комбінувати ефекти та отримувати відеофайл з більшою історією редагування. На рис. 3.11 було продемонстровано виконання більшості ефектів з панелі “Базове редагування”.

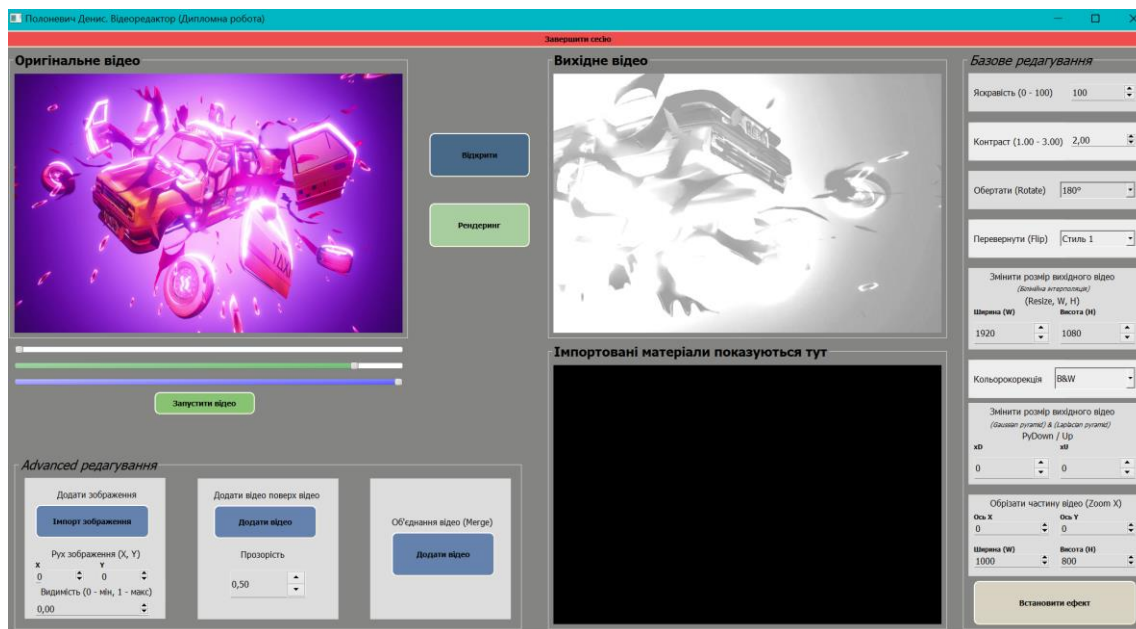


Рис. 3.11 – Функціонал “Комбінування ефектів”

У висновку панель “Базове редагування” надає користувачу можливість провести базове редагування відео, задля успішних змін після встановлення ефектів відео потрібно відрендерити за допомогою відповідного функціоналу, відредаговане відео буде зберігатися у папці *bin* на ПК.

Функціонал “Програш відео, можливості слайдеру редактора”, що показано на рис. 3.12 дає можливість програвати відео та переходити до відповідного кадру за допомогою різнокольорових повзунків. Жовтим кольором позначений початковий кадр, зеленим – теперішній, синім – останній.

Функціонал також розширений до можливості користувачу виділити окремий фрагмент відео та зарендерити умовні 3 секунди з 10 секундного відео. Демонстрація цього функціоналу буде наявна у відео до захисту кваліфікаційної роботи.

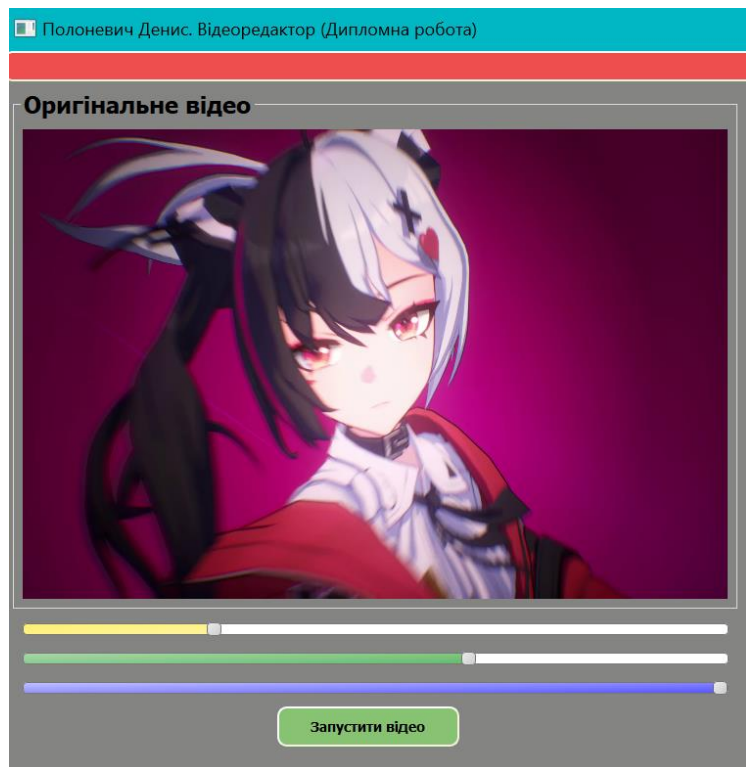


Рис. 3.12 – Функціонал “Програш відео та слайдер”

Функціонал “Додавання зображення поверх відео” (рис. 3.13). Користувач має можливість імпортувати зображення до проекту та провести з ним маніпуляції, наприклад, коли необхідно замалювати небажану частину відео, або просто невдалу. Користувач може змінювати позицію зображення по осі X та Y, додатково змінити прозорість зображення. Медіа-файли додатково показуються у відповідному вікні.

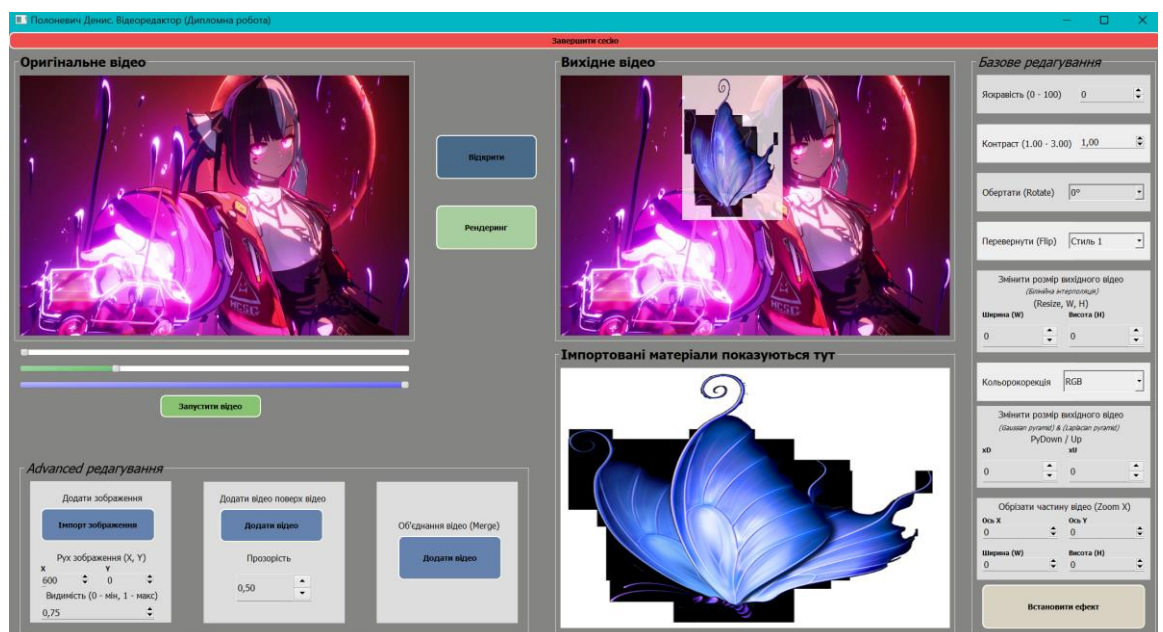


Рис. 3.13 – Функціонал “Додати зображення”

Функціонал “Додати відео поверх відео” (рис. 3.14) може бути корисним для створення різних сцен з головною ідеєю спогадів, або створення якогось нового арт-стилю відео. Головною ідеєю ефекту є комбінування двох відеофайлів в одне з відповідною прозорістю.

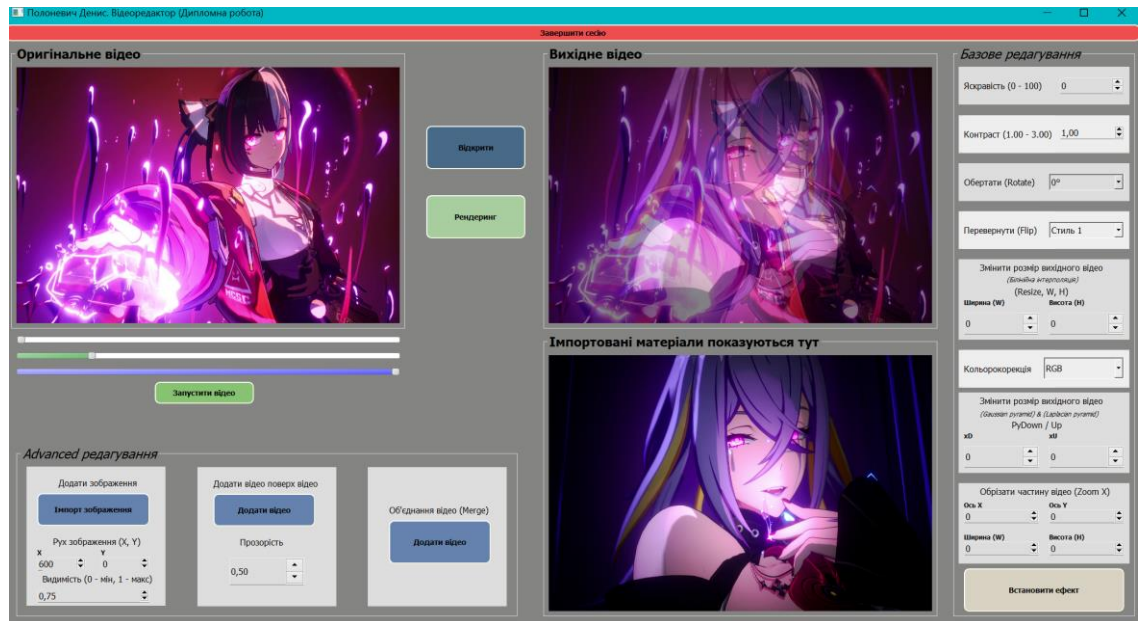


Рис. 3.14 – Функціонал “Додати відео поверх відео”

Функціонал “Об’єднання відео за пропорційним коефіцієнтом” (рис. 3.15). Користувач може об’єднати два вихідних відео в одне з заданою пропорцією, це може бути корисним для створення об’єднаних сцен з різних камер.

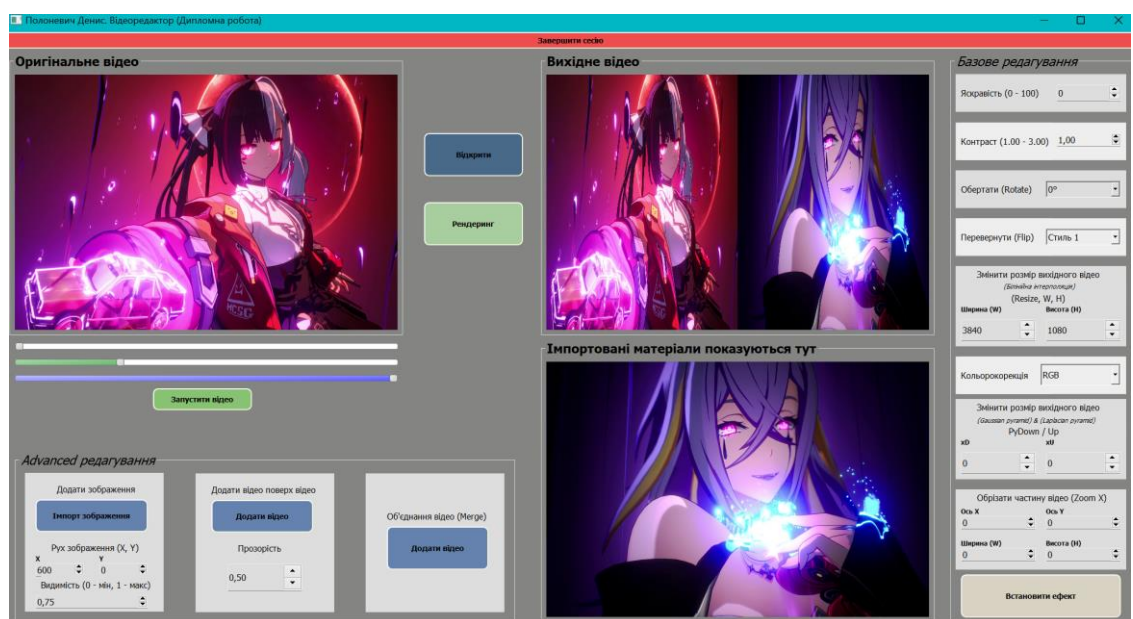


Рис. 3.15 – Функціонал “Об’єднання відео”

У редакторі також реалізовано функціонал роботи з аудіо, якщо воно є у оригінальному відео. Будь-які маніпуляції з відео збережуть його вихідне аудіо. Додатково, після редагування такого відео на персональному комп'ютері користувач отримає два окремі файли, умовно .mp3 та .mp4-форматів. Це дуже зручно, коли головною ціллю є просте розділення каналів, аудіо та відео.

У цьому підрозділі ми розглянули інтерфейс і робочий процес нелінійного відеоредактора. Ці візуальні матеріали дають уявлення про зручний дизайн і функціональні можливості, які пропонує система, дозволяючи користувачам ефективно користуватись редактором.

3.2 Тестування нелінійного відеоредактора

Тестування є критичним етапом для забезпечення якості та надійності програмного забезпечення. Воно дозволяє виявити помилки та недоліки на ранніх стадіях розробки, що зменшує витрати на їх виправлення та покращує користувацький досвід. Зокрема, тестування допомагає переконатися, що всі функції працюють коректно, а також оцінити продуктивність та стабільність програми [40 – 41].

Ініціюємо процес тестування програми. Почнемо з аналізу функціоналу створення робочого проекту:

Таблиця 3.1

Тестування процесу створення робочого проекту

<i>Номер тесту</i>	<i>Очікуваний результат</i>	<i>Фактичний результат</i>	<i>Пройдено/не пройдено (P/F)</i>
1	Імпортоване відео включає у себе коректне ім'я, латинські літери або кирилицю, без спеціальних символів, розмір та тип відео – підтримується редактором	Як і очікувалося	P

2	Імпортоване відео включає некоректне ім'я або його тип не підтримується редактором	З'явиться допоміжне вікно, що показуватиме тип помилки	F
---	--	--	---

Перейдемо до аналізу функціоналу редагування відео:

Таблиця 3.2

Тестування процесу редагування відео

<i>Номер тесту</i>	<i>Очікуваний результат</i>	<i>Фактичний результат</i>	<i>Пройдено/не пройдено (P/F)</i>
1	Редагування відео відбувається коректно, з урахуванням можливостей ефектів. У додатку написані функціональні можливості при кожній дії,	Як і очікувалося	P
2	Редагування відео вийшло за межі можливого, некоректні записи у полях, зміна розміру відео до мінусових чисел, додавання веб-файлів зображень та інше	З'явиться допоміжне вікно, що показуватиме тип помилки	F

Наостанок розберемо рендер відео.

Таблиця 3.3

Тестування процесу рендеру відео

<i>Номер тесту</i>	<i>Очікуваний результат</i>	<i>Фактичний результат</i>	<i>Пройдено/не пройдено (P/F)</i>
--------------------	-----------------------------	----------------------------	-----------------------------------

1	Процес редагування – успішний, цей тест напряму залежить від минулого. Користувач натискає на кнопку рендеру, вводить назву та тип вихідного файлу та чекає на процес створення нового, відредагованого файлу	Як і очікувалося	P
2	Процес редагування – успішний, цей тест напряму залежить від минулого. Користувач натискає на кнопку рендеру, вводить некоректну назву або неіснуючий тип відео, та отримує відповідну помилку про неможливість рендеру	З'явиться допоміжне вікно, що показуватиме тип помилки	F

Проводячи тестування роботи програмного забезпечення, ми можемо переконатися, що розроблюваний продукт, відповідає необхідним функціональним можливостям і оптимально працює в різних умовах.

Ці тести допомагають виявити та усунути будь-які потенційні проблеми або помилки, забезпечуючи стабільну роботу редактора.

Висновки до розділу 3

Загалом третій розділ був присвячений інтерфейсу та порядку роботи з нелінійним відеоредактором. Він охоплює два основні напрямки: структуру інтерфейсу з подальшим функціоналом та тестування роботи програмного комплексу.

Перший розділ був присвячений структурі інтерфейсу відеоредактора. У ньому було представлено огляд різних компонентів і модулів, що складають інтерфейс системи, пропонуючи користувачам комплексне розуміння того, як вони взаємодіють з системою. Структура інтерфейсу забезпечує зручний та інтуїтивно зрозумілий досвід роботи, що сприяє ефективній роботі з редактором.

В рамках тестування відеоредактора були випробувані наступні елементи: ініціювання створення нового проекту, процес редагування відеофайлу, процес рендеренгу. В результаті, нелінійний відеоредактор демонструє стабільну роботу і виконує всі поставлені завдання.

ВИСНОВКИ

За час роботи над випускною магістерською роботою було детально досліджено оптимізаційні методи при створенні нелінійного відеоредактора для підвищення продуктивності та покращення користувацького досвіду, результатом роботи став функціональний прототип нелінійного відеоредактора. Проєкт охоплює три ключові аспекти – аналіз та написання оптимізаційних методів, вибір архітектурних та інструментарних рішень прототипу, а також відповідну технічну реалізацію відеоредактора.

У першому розділі проєкту було сформульовано задачу та окреслено головні проблеми, що вивчаються в кваліфікаційній роботі, а також описано функціональні можливості прототипу відеоредактора, який планується розробити. Потім проведено аналіз існуючих аналогів програмного забезпечення, зокрема таких відеоредакторів, як Kdenlive та Sony Vegas Pro. Окремо розглянуто технологічний стек для розробки прототипу програмного забезпечення, а також інструменти для написання коду, включаючи програмне забезпечення для створення графічного інтерфейсу. Особливу увагу приділено одній з основних задач кваліфікаційної роботи — аналізу методів оптимізації при створенні відеоредактора. Розглянуто функціональні методи роботи з відео, зокрема зміна розміру, обрізання, корекція кольору, об'єднання відеофрагментів та реалізація слайдера для прокручування відеокадрів.

Другий розділ зосереджувався на розробці та реалізації нелінійного відеоредактора. Спочатку було визначено варіанти його використання та створено об'єктно-орієнтовану структуру відеоредактора. Ретельний аналіз допоміг виокремити ключові функціональні можливості, які забезпечать задоволення потреб різних користувачів. Далі було перейдено до розробки та впровадження алгоритмів системи, спрямованих на оптимізацію графічних перетворень. Створюючи ефективні та дієві алгоритми та застосовуючи найкращі практики, було за мету підвищити продуктивність системи та забезпечити її надійну роботу.

Третій розділ присвячено інтерфейсу та методам роботи з нелінійним відеоредактором. Спочатку було описано структуру інтерфейсу відеоредактора, детально розглядаючи різні компоненти та модулі, які його складають. В межах тестування відеоредактора перевірялися такі аспекти, як ініціювання нового проєкту, процес редагування відео та рендеринг. В результаті, відеоредактор показав стабільну роботу та успішне виконання всіх завдань.

Отже, реалізований нелінійний відеоредактор є вагомим внеском у сферу роботи з відео, демонструючи як оптимізаційно-функціональні методи можуть бути використані для створення власного прототипу додатку. Система показує великий потенціал для подальшого розвитку та адаптації під інші операційні системи, відкриваючи нові можливості до редагування відео.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Аналіз можливостей фреймворка MLT для створення нелінійних відеоредакторів / Полоневич Д.В., Петросян Р.В. // Тези V Всеукраїнської науково-технічної конференції “Комп’ютерні технології: інновації, проблеми, рішення”. - 01-02 грудня 2022. – С. 215-216. URL: <https://conf.ztu.edu.ua/wp-content/uploads/2023/02/40.pdf>
2. Розробка відеоредактора для підвищення продуктивності та поліпшення користувацького досвіду в освітньому процесі / Полоневич Д.В., Вакалюк Т.А. // XI Всеукраїнська науково-практична конференція молодих вчених “НАУКОВА МОЛОДЬ – 2023”. – 21 листопада 2023. – С. 141-143. URL: http://www.uintei.kiev.ua/sites/default/files/zbirnik_konferenciyi_naukova_molod_2023.pdf
3. Аналіз можливостей кроссплатформового набору інструментів GTK при створенні відеоредактора / Полоневич Д.В., Вакалюк Т.А. // Тези доповідей VI Всеукраїнської науково-технічної конференції “Комп’ютерні технології: інновації, проблеми, рішення”. - 29-30 листопада 2023. – С. 5-8. URL: <https://conf.ztu.edu.ua/wp-content/uploads/2024/01/5.pdf>
4. The Critical Role of Video Editing Skills in Today's Digital World [Електронний ресурс] – Режим доступу до ресурсу: <https://www.ecgprod.com/the-critical-role-of-video-editing-skills/>
5. Video Editing Problems Caused by Software Failures [Електронний ресурс] – Режим доступу до ресурсу: <https://diyvideoeditor.com/debugging-video-editing-software-problems/>
6. What are the most important video editing features to look for? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.linkedin.com/advice/0/what-most-important-video-editing-features-jmldc>
7. Шаблій Л. М. / СТВОРЕННЯ НАВЧАЛЬНИХ ВІДЕО ТА ЇХ ВИКОРИСТАННЯ. *Actual Problems in the System of Education: General Secondary Education Institution – Pre-University Training – Higher Education Institution*. 2021. № 1. С. 247–250. URL: <https://doi.org/10.18372/2786-5487.1.15879>

8. Understanding the importance of app performance optimization | MoldStud. *Custom software development company* | MoldStud. URL: <https://moldstud.com/articles/p-understanding-the-importance-of-app-performance-optimization>
9. Kdenlive Free and Open Source Video Editor. [Електронний ресурс] – Режим доступу до ресурсу: [Kdenlive - Video Editing Freedom](#)
10. Краліна, Г., & Костюченко, Л. (2020). ПЕРСПЕКТИВИ ВИКОРИСТАННЯ LINUX ДЛЯ РОЗРОБКИ ПЗ. *Збірник наукових праць ЛЮГОС*, 100-102. <https://doi.org/10.36074/18.09.2020.v1.36>
11. Video editing software designed for content creators. [Електронний ресурс] – Режим доступу до ресурсу: [VEGAS Pro: Video Editing Software for Content Creators \(vegascreativesoftware.com\)](#)
12. What is Non-Linear Video Editing (NLE)? [Електронний ресурс] – Режим доступу до ресурсу: [What is Non-Linear Video Editing \(NLE\), Exactly? \(softwarehow.com\)](#)
13. C++ Introduction [Електронний ресурс] – Режим доступу до ресурсу: [C++ Introduction \(w3schools.com\)](#)
14. OpenCV: OpenCV modules. *OpenCV documentation index*. URL: <https://docs.opencv.org/4.x/index.html>
15. Ffmpeg: A complete, cross-platform solution to record, convert and stream audio and video. [Електронний ресурс] – Режим доступу до ресурсу: [FFmpeg](#)
16. OpenCV: Core functionality. *OpenCV documentation index*. URL: https://docs.opencv.org/4.x/d0/de1/group_core.html
17. OpenCV: Image Processing. *OpenCV documentation index*. URL: https://docs.opencv.org/4.x/d7/dbd/group_imgproc.html
18. OpenCV: Video Analysis. *OpenCV documentation index*. URL: https://docs.opencv.org/4.x/d7/de9/group_video.html
19. OpenCV: High-level GUI. *OpenCV documentation index*. URL: https://docs.opencv.org/4.x/d7/dfc/group_highgui.html

20. OpenCV: Camera Calibration and 3DReconstruction. URL: https://docs.opencv.org/4.x/d9/d0c/group_calib3d.html
21. OpenCV: 2D Features Framework. *OpenCV documentation index*. URL: https://docs.opencv.org/4.x/da/d9b/group_features2d.html
22. OpenCV: Images stitching. *OpenCV documentation index*. URL: https://docs.opencv.org/4.x/d1/d46/group_stitching.html
23. libavformat. [Електронний ресурс] – Режим доступу до ресурсу: https://ffmpeg.org/doxygen/4.2/group_libavf.html
24. libavcodec. [Електронний ресурс] – Режим доступу до ресурсу: https://ffmpeg.org/doxygen/6.0/group_libavc.html
25. libavfilter. [Електронний ресурс] – Режим доступу до ресурсу: https://ffmpeg.org/doxygen/7.0/group_lavfi.html
26. libavutil. [Електронний ресурс] – Режим доступу до ресурсу: https://ffmpeg.org/doxygen/6.0/group_lavu.html
27. libswscale. [Електронний ресурс] – Режим доступу до ресурсу: https://ffmpeg.org/doxygen/6.1/group_libsws.html
28. libswresample. [Електронний ресурс] – Режим доступу до ресурсу: https://ffmpeg.org/doxygen/6.0/group_lswr.html
29. libavdevice. [Електронний ресурс] – Режим доступу до ресурсу: https://ffmpeg.org/doxygen/6.0/group_lavd.html
30. What Is Qt framework, Why to Use It, and How? [Електронний ресурс] – Режим доступу до ресурсу: <https://lembargsolutions.com/blog/why-use-qt-framework>
31. What is Visual Studio? [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022>
32. Image Pyramids: OpenCV [Електронний ресурс] – Режим доступу до ресурсу: https://docs.opencv.org/4.x/d4/d1f/tutorial_pyramids.html
33. ВОЙЧУР, Ю., & МЕДЗАТИЙ, Д. (2024). МЕТОД АНАЛІЗУ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ПРЕДМЕТ ПОШУКУ ЗНАЧЕНЬ

АТРИБУТИВ ЯКОСТІ. *MEASURING AND COMPUTING DEVICES IN TECHNOLOGICAL PROCESSES*, (1), 146–151. <https://doi.org/10.31891/2219-9365-2024-77-18>

34. Варіанти використання та сценарії (Use Cases and Scenarios) [Електронний ресурс] – Режим доступу до ресурсу: <https://www.maxzosim.com/use-cases-and-scenarios/>

35. Що таке об'єктно-орієнтоване програмування: принципи, переваги та недоліки [Електронний ресурс] – Режим доступу до ресурсу: <https://goit.global/ua/articles/shcho-take-ob-iektno-oriientovane-prohramuvannia-pryntsypy-perevahy-ta-nedoliky/>

36. Діаграма активності UML [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mindonmap.com/uk/blog/uml-activity-diagram/>

37. Що таке специфікація вимог: визначення, найкращі інструменти та методи? [Електронний ресурс] – Режим доступу до ресурсу: <https://promoboz.com/journal/2024/2-101-may/spetsyfikatsiya-vymog-korystuvacha/>

38. C++ Memory Address [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/cpp/cpp_references_memory.asp

39. Windows [Електронний ресурс] – Режим доступу до ресурсу: [What is Windows? \(computerhope.com\)](https://computerhope.com/what-is-windows/)

40. Що таке тестування програмного забезпечення? [Електронний ресурс] – Режим доступу до ресурсу: <https://qalight.ua/baza-znaniy/shho-take-testuvannya-programnogo-zabezpechennya/>

41. Модульне тестування [Електронний ресурс] – Режим доступу до ресурсу: <https://qalight.ua/baza-znaniy/modulne-testuvannya/>

ДОДАТКИ

Клас *XAudio* з кодом по роботі з аудіо, використання бібліотеки Ffmpeg:

```
#include "XAudio.h"
#include <iostream>
#include <QMutex>

using namespace std;

struct XTime
{
    XTime(int tms)
    {
        h = (tms / 1000) / 3600;
        m = ((tms / 1000) % 3600) / 60;
        s = (tms / 1000) % 60; //
        ms = tms % 1000;
    }

    std::string ToString()
    {
        char buf[16] = { 0 };
        sprintf(buf, "%d:%d:%d.%d", h, m, s, ms);
        return buf;
    }

    int h = 0;
    int m = 0;
    int s = 0;
    int ms = 0;
};

class CXAudio : public XAudio
{
public:
    QMutex mutex;

    bool ExportA(std::string src, std::string out, int beginMs, int
outMs)
    {
        string cmd = "ffmpeg ";
        cmd += " -i ";
        cmd += src;
        cmd += " ";
        if (beginMs > 0)
        {
            cmd += " -ss ";
            XTime xt(beginMs);
            cmd += xt.ToString();
        }
    }
};
```

```

    }
    if (outMs > 0)
    {
        cmd += " -t ";
        XTime xt(outMs);
        cmd += xt.ToString();
    }
    cmd += " -vn -y ";
    cmd += out;
    cout << cmd << endl;

    mutex.lock();
    system(cmd.c_str());
    mutex.unlock();
    return true;
}

bool Merge(std::string v, std::string a, std::string out)
{
    string cmd = "ffmpeg -i ";
    cmd += v;
    cmd += " -i ";
    cmd += a;
    cmd += " -c copy ";
    cmd += out;
    cout << cmd << endl;

    mutex.lock();
    system(cmd.c_str());
    mutex.unlock();
    return true;
}
};

XAudio* XAudio::Get()
{
    static CXAudio ca;
    return &ca;
}

XAudio::XAudio()
{
}

XAudio::~XAudio()
{
}

```

Клас *XVideoThread* по роботі з відео потоком:

```
#include "XVideoThread.h"
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <iostream>
#include "XFilter.h"

using namespace std;
using namespace cv;

static VideoCapture cap1;
static VideoCapture cap2;
static VideoWriter vw;
static bool isexit = false;

bool XVideoThread::StartSave(const std::string filename, int width,
int height, bool isColor)
{
    cout << "Start export" << endl;
    Seek(begin);
    mutex.lock();

    if (!cap1.isOpened())
    {
        mutex.unlock();
        return false;
    }

    if (width <= 0)
        width = cap1.get(CAP_PROP_FRAME_WIDTH);
    if (height <= 0)
        height = cap1.get(CAP_PROP_FRAME_HEIGHT);

    vw.open(filename,
        VideoWriter::fourcc('X', '2', '6', '4'),
        this->fps,
        Size(width, height), isColor
    );

    if (!vw.isOpened())
    {
        mutex.unlock();
        cout << "start save failed!" << endl;
        return false;
    }
}
```

```

        this->isWrite = true;
        desFile = filename;
        mutex.unlock();
        return true;
    }

void XVideoThread::StopSave()
{
    cout << "stop export" << endl;
    mutex.lock();
    vw.release();
    isWrite = false;
    mutex.unlock();
}

void XVideoThread::SetBegin(double p)
{
    mutex.lock();
    double count = cap1.get(CAP_PROP_FRAME_COUNT);
    int frame = p * count;
    begin = frame;
    mutex.unlock();
}

void XVideoThread::SetEnd(double p)
{
    mutex.lock();
    double count = cap1.get(CAP_PROP_FRAME_COUNT);
    int frame = p * count;
    end = frame;
    mutex.unlock();
}

bool XVideoThread::Seek(double pos)
{
    double count = cap1.get(CAP_PROP_FRAME_COUNT);
    int frame = pos * count;
    return Seek(frame);
}

bool XVideoThread::Seek(int frame)
{
    mutex.lock();
    if (!cap1.isOpened())
    {
        mutex.unlock();
        return false;
    }

    int re = cap1.set(CAP_PROP_POS_FRAMES, frame);

```

```

        if (cap2.isOpened())
            cap2.set(CAP_PROP_POS_FRAMES, frame);
        mutex.unlock();
        return re;
    }

double XVideoThread::GetPos()
{
    double pos = 0;
    mutex.lock();
    if (!cap1.isOpened())
    {
        mutex.unlock();
        return pos;
    }
    double count = cap1.get(CAP_PROP_FRAME_COUNT);
    double cur = cap1.get(CAP_PROP_POS_FRAMES);
    if (count > 0.001)
        pos = cur / count;
    mutex.unlock();
    return pos;
}

bool XVideoThread::Open2(const std::string file)
{
    Seek(0);
    cout << "open2 :" << file << endl;
    mutex.lock();
    bool re = cap2.open(file);
    mutex.unlock();
    cout << re << endl;
    if (!re)
        return re;

    width2 = cap2.get(CAP_PROP_FRAME_WIDTH);
    height2 = cap2.get(CAP_PROP_FRAME_HEIGHT);
    return true;
}

bool XVideoThread::Open(const std::string file)
{
    cout << "open :" << file << endl;
    Seek(0);
    mutex.lock();
    bool re = cap1.open(file);
    mutex.unlock();

    cout << re << endl;
    if (!re)
        return re;
}

```

```

        fps = cap1.get(CAP_PROP_FPS);
        width = cap1.get(CAP_PROP_FRAME_WIDTH);
        height = cap1.get(CAP_PROP_FRAME_HEIGHT);
        if (fps <= 0) fps = 25;
        src1file = file;
        double count = cap1.get(CAP_PROP_FRAME_COUNT);
        totalMs = (count / (double)fps) * 1000;
        return true;
    }

void XVideoThread::run()
{
    Mat mat1;
    while (!isexit)
    {
        mutex.lock();
        if (isexit)
        {
            mutex.unlock();
            break;
        }
        if (!cap1.isOpened())
        {
            mutex.unlock();
            msleep(5);
            continue;
        }
        if (!isPlay)
        {
            mutex.unlock();
            msleep(5);
            continue;
        }

        int cur = cap1.get(CAP_PROP_POS_FRAMES);
        if ((end > 0 && cur >= end) || !cap1.read(mat1) ||
mat1.empty())
        {
            mutex.unlock();
            if (isWrite)
            {
                StopSave();
                SaveEnd();
            }

            msleep(5);
            continue;
        }

        Mat mat2 = mark;
        if (cap2.isOpened())

```

```

        {
            cap2.read(mat2);
        }

        if (!isWrite)
        {
            ViewImage1(mat1);
            if (!mat2.empty())
                ViewImage2(mat2);
        }

        Mat des = XFilter::Get()->Filter(mat1, mat2);

        if (!isWrite)
            ViewDes(des);

        int s = 1000 / fps;
        if (isWrite)
        {
            s = 1;
            vw.write(des);
        }

        msleep(s);
        mutex.unlock();
    }
}

XVideoThread::XVideoThread()
{
    start();
}

XVideoThread::~XVideoThread()
{
    mutex.lock();
    isexit = true;
    mutex.unlock();
    wait();
}

```

Клас *XVideoWidget* по роботі з головним вікном додатку, операції малювання, зчитування відео та функціонал сигнал – слот:

```
#include "xvideowidget.h"
#include <QPainter>
#include <opencv2/imgproc.hpp>

using namespace cv;

void XVideoWidget::SetImage(cv::Mat mat)
{
    QImage::Format fmt = QImage::Format_RGB888;
    int pixSize = 3;

    if (mat.type() == CV_8UC1)
    {
        fmt = QImage::Format_Grayscale8;
        pixSize = 1;
    }

    if (img.isNull() || img.format() != fmt)
    {
        delete img.bits();
        uchar* buf = new uchar[width() * height() * pixSize];
        img = QImage(buf, width(), height(), fmt);
    }

    Mat des;

    cv::resize(mat, des, Size(img.size().width(),
img.size().height()));

    if (pixSize > 1)
        cv::cvtColor(des, des, COLOR_BGR2RGB);

    memcpy(img.bits(), des.data, des.cols * des.rows *
des.elemSize());

    update();
}

XVideoWidget::XVideoWidget(QWidget* p) : QOpenGLWidget(p)
{
}

void XVideoWidget::paintEvent(QPaintEvent* e)
{
    QPainter p;
```

```
p.begin(this);  
p.drawImage(QPoint(0, 0), img);  
p.end();  
}  
  
XVideoWidget::~XVideoWidget()  
{  
  
}
```