Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео»

Виконав: студент 2-го курсу, групи 1ПІ-23м
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Голубенко Р.Р.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ:
Катєльніков Д.І.
(прізвище та ініціали)
«09» грудня 2024 р.

Опонент к.т.н., ст. викл. каф. ОТ:
Богомолов С.В
(прізвище та ініціали)
«09» грудня 2024 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н. проф. Романюк О.Н.
(прізвище та ініціали)
«09» грудня 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 інформаційні технології
Спеціальність – 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«18» вересня 2024 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙУ РОБОТУ СТУДЕНТУ

Голубенку Роману Руслановичу

1. Тема роботи – розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео.

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від
«17» вересня 2024 року № 310.

2. Строк подання студентом роботи
3 грудня 2024 року

3. Вихідні дані до роботи: середовища розробки Visual Studio Code, мова розробки Rust, програмний пакет Davinchi Resolve Studio 13, програмний пакет ffmpeg 4.2, операційна система – Windows 11.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану розвитку систем створення VHS-ефектів та постановка задачі; розробка методів та засобів реалізації симуляції аналогового шуму та артефактів; розробка засобу створення вінтажних ефектів VHS-плівки; тестування додатку; економічна частина; висновки; перелік посилань; додатки.

5. Перелік графічного матеріалу: блок-схеми алгоритмів роботи; тестування додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Катєльніков Д. І, к.т.н., доцент кафедри ПЗ	19.09.24	2.12.24
5	Причепа І. В. к.е.н., доцент кафедри ЕПВМ	29.10.24	10.11.24

7. Дата видачі завдання 19 вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку систем створення VHS-ефектів та постановка задач дослідження	20.09.2024 – 30.09.2024	Вик.
2	Розробка методів та засобів реалізації симуляції аналогового шуму та артефактів	01.10.2024 – 10.10.2024	Вик.
3	Розробка засобу для створення вінтажних ефектів VHS-плівки	11.10.2024 – 25.10.2024	Вик.
4	Тестування програмного додатку	26.10.2024 – 15.11.2024	Вик.
5	Економічна частина	16.11.2024 – 30.11.2024	Вик.

Студент _____ (підпис) Голубенко Р.Р. (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ (підпис) Катєльніков Д.І. (прізвище та ініціали)

Анотація

УДК 004.912.032.26

Голубенко Р.Р. Розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2024. 134 с.

На укр. мові. Бібліогр.: 22 назв; рис.: 46; табл. 12.

У магістерській кваліфікаційній роботі проведено детальний аналіз методів та алгоритмів створення VHS ефектів.

Основну увагу приділено аналізу ключових етапів побудови таких ефектів, які дозволяють точно відтворити властивості аналогового запису, включаючи кольорові спотворення, артефакти розділення полів, шум та втрату деталізації.

Розроблено алгоритм, який включає перетворення кольорового простору з RGB у YIQ для ізольованої роботи з яскравістю та кольоровими компонентами відео. Запропоновано метод генерації синтетичних шумів для симуляції нестабільності кольорового сигналу, а також низькочастотну фільтрацію для зменшення високочастотних деталей, характерних для цифрового зображення. Алгоритм також враховує динамічні артефакти, такі як горизонтальне зміщення рядків, які є типовими для записів на VHS-касетах.

Для реалізації методу було розроблено програмне забезпечення з використанням мови програмування Rust, що забезпечує високу продуктивність і надійність при обробці відеофайлів. У рамках роботи також створено модуль для нормалізації та корекції полів відео, який дозволяє обробляти вхідні кадри та застосовувати до них зазначені ефекти.

Результати, отримані в ході дослідження, продемонстрували ефективність алгоритму у створенні реалістичних VHS-ефектів, що імітують властивості

старих аналогових записів. Використані підходи можуть знайти застосування у творчих проектах, кінематографі, створенні рекламних відео та цифровому реставруванні матеріалів. Розроблене програмне забезпечення є гнучким і може бути інтегроване в існуючі системи обробки відео, а також використане як окремий інструмент для генерації вінтажних ефектів.

Ключові слова: VHS ефекти, симуляція аналогового відео, генерація відеошуму, обробка цифрового відео, відео ефекти.

Abstract

Holubenko R.R. Development of a method and software protection for creating vintage VHS tape effects in digital video. Vinnytsia: VNTU, 2024. – 134 p.

In Ukrainian language. Bibliographer: 22 titles; fig.: 46; tabl. 12.

In the master's qualification thesis, a detailed analysis of methods and algorithms for creating VHS effects has been conducted.

The primary focus is on analyzing the key stages of constructing such effects that accurately reproduce the characteristics of analog recordings, including color distortions, field separation artifacts, noise, and detail loss.

An algorithm has been developed that includes color space conversion from RGB to YIQ to enable isolated manipulation of brightness and chrominance components in video. A method for generating synthetic noise to simulate color signal instability has been proposed, along with low-pass filtering to reduce high-frequency details characteristic of digital images. The algorithm also accounts for dynamic artifacts such as horizontal displacement or "line skipping," typical of VHS tape recordings.

To implement the method, software was developed using the Rust programming language, ensuring high performance and reliability in video file processing. As part of the project, a module for video field normalization and correction was created, allowing input frames to be processed and the described effects applied.

The results of the research demonstrated the algorithm's effectiveness in creating realistic VHS effects that mimic the properties of old analog recordings. The approaches used can be applied in creative projects, cinematography, advertising videos, and digital material restoration. The developed software is flexible and can be integrated into existing video processing systems or used as a standalone tool for generating vintage effects.

Keywords: VHS effects, analog video simulation, video noise generation, digital video processing, video effects.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Візуальні ефекти є одним із найбільш цінних інструментів у створенні цифрового контенту, адже їх можна використати для надання відеоматеріалам унікального вигляду, відтворення естетики певної епохи чи імітації пошкоджень, вибухів та інших фізичних процесів. Окремим художнім напрямком стали ефекти ретро-плівки, які викликають у глядачів цілий спектр специфічних почуттів і слугують засобом налаштування глядача при сприйнятті відеоконтенту на певне занурення в епоху, яка асоціюється з технологіями 80-х та 90-х років. Завдяки розвитку цифрових технологій процес створення ефіектів став значно простішим, що дозволяє досягати високої точності відтворення аналогових вінтажних ефектів VHS-плівки у цифрових відеоматеріалах. Проблема автоматизації накладання таких ефектів є досить гострою, оскільки виникає необхідність створювати нові ефективні методи обробки відеоряду для симуляції недосконалості відеоплівки, псування з часом, особливостей, зокрема, у стилі VHS-касет.

Завдання створення VHS-ефектів належить до класу задач візуальної обробки відео, де необхідно імітувати пошкодження, властиві аналоговим системам запису. У цифровому відео ці дефекти накладаються за допомогою спеціальних фільтрів і алгоритмів, які імітують фізичні артефакти старих записів, таких як розмагнічування плівки або нестабільність сигналу. Наприклад, у роботі «Creating VHS Effects Using Digital Filters and Signal Processing Techniques»[1] розглянуто використання складних цифрових фільтрів для створення ефекту пошкодженої плівки. Ці методи дозволяють відтворювати нестабільні кольорові смуги, мерехтіння зображення та шуми, що притаманні відео з касет VHS.

Багато сучасних систем для створення VHS-ефектів інтегровані у професійні відеоредактори у вигляді плагінів. Проте такі рішення іноді обмежуються лише базовими наборами ефектів, ефективність яких можна покращити через налаштування або розробку власних інструментів. Наприклад,

використання платформ на основі OpenFx дозволяє створювати високоналаштовані плагіни, які підтримують різноманітні ефекти та забезпечують більшу гнучкість у роботі з відеоматеріалами. Однак в цьому випадку вимоги до кваліфікації користувача значно зростають. Тому розробка власного ефективного плагіна для генерації вінтажних ефектів VHS-плівки є актуальною задачею, оскільки це дозволяє отримати більш точний контроль над процесом накладання візуальних артефактів і досягти значно кращих результатів.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою магістерської кваліфікаційної роботи є розширення функціональних можливостей набору інструментів відеорежисера і підвищення естетичної цінності відеоконтенту за рахунок використання модулів генерації вінтажних ефектів VHS-плівки у цифрових відео.

Основними задачами роботи є:

- провести аналіз існуючих методів і засобів створення вінтажних ефектів з метою визначення шляхів їх покращення;
- розробити метод створення вінтажних ефектів VHS-плівки;
- розробити алгоритм створення вінтажних VHS-ефектів;
- розробити програмний модуль для реалізації розробленого алгоритму;
- розробити плагін для інтеграції із відеоредакторами;
- провести тестування програмного додатку.

Об’єктом дослідження є процес додання вінтажних відеоефектів при редагуванні цифрового відеоконтенту.

Предметом дослідження є методи та засоби генерації вінтажних ефектів VHS-плівки у цифрових відео.

Методи дослідження. У процесі досліджень використовувались: теорія інформації і обробка сигналів для розробки алгоритмів генерації шуму, кольорових фільтрів для імітування вигоряння кольорів та алгоритмів Фур'є для створення деяких відео-ефектів; теорія кольористики для розробки методів обробки та аналізу кольорових моделей, зокрема YIQ і RGB, для точної передачі вицвітання кольорів і кольорових спотворень; комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

– подальшого розвитку отримав метод створення вінтажних ефектів VHS-плівки у цифровому відео, який, на відміну від існуючих, використовує комбінацію можливостей RGB моделі, що функціонують з колірною інформацією у вигляді бітів, і моделі YIQ колірного аналогового телебачення, яка дозволяє компенсувати недоліки людського зору при розпізнаванні близьких кольорових відтінків. У результаті отримано нову - глибоку колірну модель на основі YIQ і RGB, яка дозволяє більш точно відтворювати характерні спотворення кольорів і дефектів зображення на відеоплівці;
– подальшого розвитку отримав метод симуляції шуму та артефактів, який, на відміну від існуючих, додатково включає етап регуляризації та інтерполяції піксельних даних, завдяки чому досягається ефект природного спотворення, що властиво аналоговому запису на VHS.

Практична цінність отриманих результатів полягає в тому, що на основі отриманих у роботі теоретичних положень запропоновано алгоритми і розроблено програмні засоби для створення вінтажних VHS-ефектів у цифрових відео.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У науковій роботі, опублікованій у співавторстві, автору належать такі результати: модуль

створення цифрових дисторсій [4]; алгоритм створення цифрових вінтажних VHS-ефектів та інтеграція у бібліотеку OpenFx.

Апробація матеріалів магістерської кваліфікаційної роботи. Результати магістерської кваліфікаційної роботи доповідалися та обговорювалися на:

– міжнародній науково-практичній конференції молодих вчених та студентів ««Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації»» (Одеса 2024);

Публікації. Основні результати дослідженні опубліковані в наступних наукових роботах

– тези доповіді на конференції «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації» [10];

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п’яти розділів, висновків, списку літератури, що містить 23 найменування, 4 додатки. Робота містить 63 ілюстрацію, 9 таблиць.

1 АНАЛІЗ СТАНУ РОЗВИТКУ СИСТЕМ СТВОРЕННЯ VHS-ЕФЕКТІВ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану вінтажних ефектів VHS-плівки у цифрових відео

Стан вінтажних ефектів VHS-плівки у цифрових відео є актуальним питанням у контексті сучасних тенденцій медіавиробництва. Зростання інтересу до ретроестетики та ностальгічних мотивів у візуальному мистецтві, зокрема в кіно та рекламних матеріалах, стимулювало розробку і застосування цифрових ефектів, що імітують візуальні характеристики старих VHS-плівок. Ці ефекти часто використовуються для створення відчуття автентичності та стилізації відео під аналогові часи.

Основні характеристики VHS-плівки[1] включають: зношення кадру, шуми, нестабільність зображення, кольорові артефакти, ефекти розсинхронізації та порушення ліній відображення. Важливим аспектом таких ефектів є те, що вони можуть бути як навмисно перебільшені для естетичних цілей, так і досить точно відтворені.

Рисунок 1.1 – Відеофрагмент знятий на VHS-плівку

Попри те, що сучасні алгоритми та програмне забезпечення дають змогу досягти високого рівня реалістичності вінтажних ефектів, виникають питання щодо точності їх відтворення, особливо при застосуванні у високій роздільній здатності. Старі аналогові системи мали свої обмеження, які не завжди коректно імітуються в цифрових середовищах через різницю у фізичних носіях та форматах.

Багато сучасних режисерів та людей причетних до кіновиробництва досі надають перевагу зйомці саме на аналогові формати, оскільки вони дають певні як вони кажуть «теплі тони та відчуття натуральності».

Наприклад відомий режисер Квентін Тарантіно у зйомці на плівку[2] замість цифрових технологій зумовлена його глибокою повагою до автентичності та художньої цілісності цього методу. Він порівнює роботу з плівкою із смакуванням стейка — багатий і наповнений досвід, тоді як цифровий формат для нього нагадує вегетаріанський бургер — замінник, що не має сутності оригіналу.

На думку Тарантіно, плівка є неперевершеною у здатності передавати органічну, фактурну якість зображення. Вона надає історії відчуття тактильності, закріплюючи візуальний ряд у середовищі, яке здається більш "живим". Натуральна зернистість, контраст і глибина кольору, які забезпечує плівка, створюють унікальну кінематографічну атмосферу, яку важко відтворити за допомогою цифрових технологій. Тарантіно цінує цю властивість плівки, яка підсилює настрій і візуальний вплив його історій, глибше занурюючи глядачів у сюжет.

Окрім естетики, він цінує дисципліну, яку плівка накладає на процес створення фільму. Зйомка на плівку вимагає ретельного планування, точного освітлення та продуманого кадрування, оскільки можливості для помилок обмежені. Такий підхід сприяє формуванню культури майстерності на знімальному майданчику, вимагаючи відданості від режисера, оператора і всієї команди. У порівнянні з цим, цифрові технології дозволяють створити більш

розслаблену атмосферу, де помилки можна виправити на етапі постпродакшну — динаміку, яку Тарантіно вважає такою, що послаблює художність кіновиробництва.

Для зйомки на плівку потрібна велика кількість плівки та специфічні камери, що зазвичай мають зовсім не маленьку вагу та об’єм. Для великої кількості плівки потрібна велика котушка-носій(див. рис 1.2) які самі по собі теж важать немало, але навіть такі обмеження не зупиняють режисера обирати аналогове, а не цифрове.

Рисунок 1.2 – Зйомка «Одного разу в Голівуді» на камеру із плівкою

Для Тарантіно обмеження плівки є її сильними сторонами — вона змушує кінематографістів бути цілеспрямованими та зосередженими в творчому процесі. Це відчуття дисципліни відповідає його філософії кіно як мистецтва, яке має бути іммерсивним, тактильним і нефільтрованим, як багатий досвід смакування автентичного стейка.

Ще один всесвітньо відомий кінорежисер Крістофер Нолан аргументує використання плівки тим, що вона передає сутність світла і деталей так, як цифрові формати не можуть. Він цінує природну, органічну якість плівки, яка,

на його думку, більше відповідає тому, як людське око сприймає реальність. Ця відповідність посилює його здатність створювати іммерсивні та візуально автентичні історії. Для Нолана плівка має емоційний і інтуїтивний резонанс[2], який поглиблює наратив, дозволяючи йому точніше передати своє художнє бачення.

Аналогова природа плівки також вимагає дисциплінованого підходу до кінематографії, стимулюючи ретельне продумування кожного кадру. Така свідомість відображає прихильність Нолана до майстерності, забезпечуючи, щоб його творчі рішення були осмисленими й продуманими. Крім того, текстура, зернистість і тонкі недосконалості плівки створюють кінематографічну естетику, яка здається позачасовою, збагачуючи досвід сприйняття історії глядачами.

Нолан як і Тарантіно наголошує, що обмеження плівки є частиною її чарівності. Ці обмеження змушують кінематографістів безпосередньо взаємодіяти з матеріалом, ухвалюючи рішення, які посилюють художній і емоційний вплив фільму. Працюючи з плівкою, він зберігає тактильний зв’язок із мистецтвом створення кіно, який, на його думку, часто втрачається через зручність і гнучкість цифрових форматів.

Ба-більше Нолан знімає на IMAX[3] формат плівки, яка дозволяє збільшити покриваючий кадр самої плівки, та таким чино захопити більше зображення на саму плівку. Її основна інновація полягає у використанні унікального формату плівки 15/70, де кожен кадр має висоту 70 мм і ширину 15 перфорацій, що приблизно у 10 разів більше, ніж стандартний кадр 35 мм. Такий значно більший розмір плівки дозволяє IMAX захоплювати та проектувати зображення з винятковою чіткістю, роздільною здатністю і деталізацією, навіть на гігантських екранах до 37 метрів у ширину і понад 25 метрів у висоту.

Рисунок 1.3 – Плівкова IMAX камера на зйомці «Оппенгеймера»

Особливі співвідношення сторін формату, такі як 1.43:1 і 1.90:1, забезпечують максимальне використання висоти екрану, заповнюючи більшу частину поля зору глядачів. Завдяки цьому IMAX став улюбленим серед режисерів для створення грандіозних сцен. Однак висока вартість формату, великі розміри камер і логістичні труднощі (див рис 1.4) означають, що його використовують вибірково, зазвичай для окремих сцен або високопрофільних проектів, таких як "Темний лицар" і "Оппенгеймер" Крістофера Нолана.

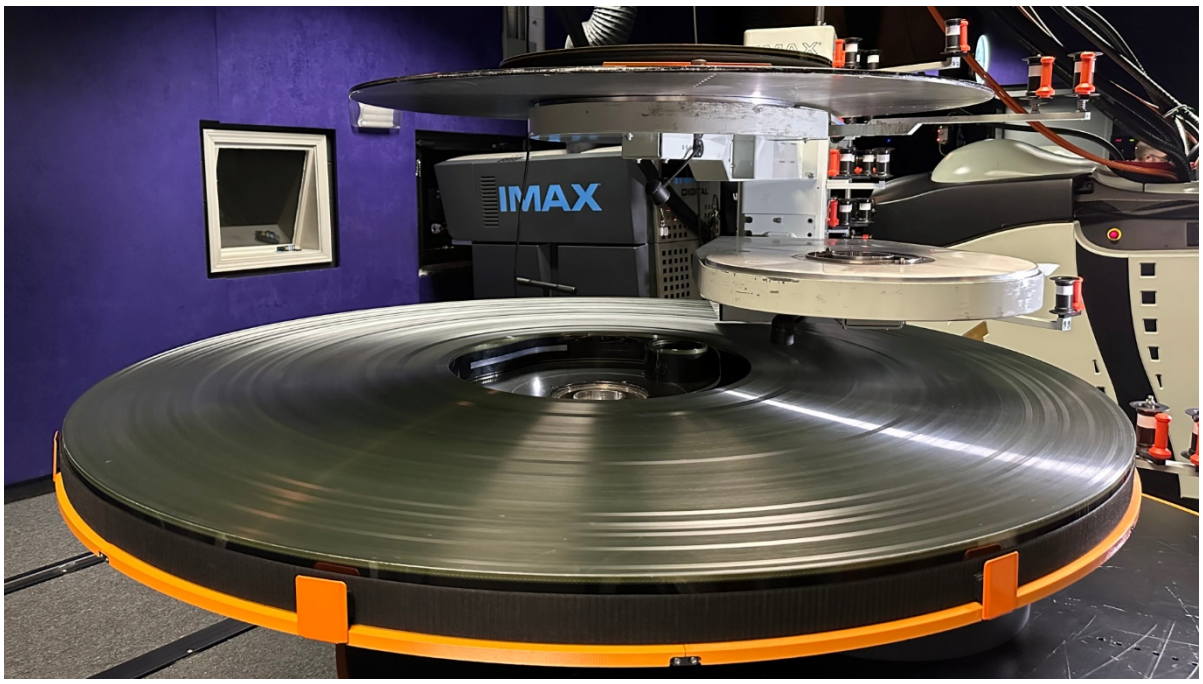

Рисунок 1.4 – IMAX плівка із фільмом «Опенгеймер»

Ще один із культових режисерів Дені Вільньов використав новітню техніку створення аналогових ефектів для кінофільму «Дюна 2». Дені Вільньов використав унікальний гібридний процес [4]: зйомка велася цифровим способом, після чого відзнятий матеріал переносили на плівку 35 мм, а потім відсканували назад у цифровий формат.

Такий підхід був спрямований на пом’якшення різкості цифрового зображення, надаючи йому мальовничої текстури, яка поєднує точність цифрових технологій із теплом аналогового формату. Оператор-постановник Грейг Фрейзер пояснив, що цей метод створив візуальну естетику, яку неможливо досягти за допомогою лише одного з цих носіїв. Це посилило відчуття занурення та епічність фільму, об'єднавши передові технології з класичним кінематографічним мистецтвом.

Рисунок 1.5 – Кадр із кінострічки «Дюна 2»

Також цього року вийшла кінострічка яка використала у створенні як ефект плівки, так і VHS. Кінострічка "Учень. Історія Трампа" (2024) розповідає нам історію становлення та шляху до успіху однієї з найбільш суперечливих постатей сучасної політики — Дональда Трампа. Фільм охоплює різні етапи його життя, починаючи з дитинства у заможній сім'ї забудовників у Нью-Йорку, де формувалися перші амбіції, і до часів його президентства, які назавжди змінили політичний ландшафт Сполучених Штатів.

Особливу увагу приділено періоду 1980–1990-х років, коли Трамп ставав символом американської мрії, будуючи хмарочоси, гральні комплекси та впроваджуючи себе в масову культуру через численні телевізійні виступи.

Оскільки сюжет кінострічки розгортається протягом тривалого часу, для візуального показу зміни епох було використано ефект кіноплівки (70-і роки) та згодом VHS (80-і та 90-і роки).

Як зазначає оператор-постановник, значна увага була приділена експериментам із різними типами носіїв: "Ми провели багато тестів, використовуючи плівку формату 16 мм — як негативну, так і реверсивну... Потім випробовували аналогові відеокамери того періоду. Інтуїтивно, робота з

реальними інструментами тієї епохи — якщо їх ще можна було знайти — була найбільш логічною... Але ми поступово зіткнулися з дуже низькою роздільною здатністю кількох камер, які збереглися з того часу. Хоча 16 мм, звісно, не становила проблеми, зйомка мастер-копій у форматі 625 рядків поставила нас у дуже незручне становище, коли мова зайшла про створення копії, придатної для сучасних стандартів"[5].

Рисунок 1.6 – Ефект VHS у фільмі «Учень. Історія Трампа»

У ході цих експериментів було розроблено спеціальний робочий процес. Для забезпечення бажаного візуального ефекту зйомка проводилася на камеру Alexa 35, оснащену старими об'єктивами для 16 мм плівки, що зменшувало ефективний розмір роботи сенсора.

Для досягнення кінцевого результату були застосовані два різних LUT-файли: один для імітації 16 мм та інший — для аналогових відео-шумів.

Таким чином вдалось зекономити кошти на усіх процесах створення аналогового відео та досягти вінтажної естетики з використанням сучасних технологій.

Таким чином, створення такої технології дозволяє не тільки зберігати вінтажні ефекти як частину історичної спадщини відеопродукції, але й активно використовувати їх у сучасних креативних проектах, де вони поєднуються з новітніми цифровими підходами.

1.2 Порівняльний аналіз аналогів

Як правило, ефекти для створення вінтажних VHS-ефектів є частинами програмного комплексу для обробки цифрового відео та створення візуальних ефектів, або представляють із себе окремі цифрові пакети. Розглянемо такі застосунки створення вінтажних ефектів:

- Adobe Premier Pro;
- 1967: Retro Filters & Effects;
- Hipstamatic;

Adobe Premier Pro – це потужний інструмент для професійного відеомонтажу, що пропонує широкий набір ефектів для роботи з відео та аудіо. Ефекти в Premiere Pro дозволяють користувачам вносити візуальні та аудіо-зміни до кліпів, коригувати колір, стабілізувати відео, додавати переходи й створювати візуальні спецефекти. Однією з основних категорій ефектів є корекція кольору. Premiere Pro пропонує глибокі налаштування кольору через інструмент Lumetri Color, який дає змогу змінювати експозицію, контраст, насиченість і баланс кольорів, а також використовувати LUT-файли[6] для швидкої корекції (рисунок 1.1).

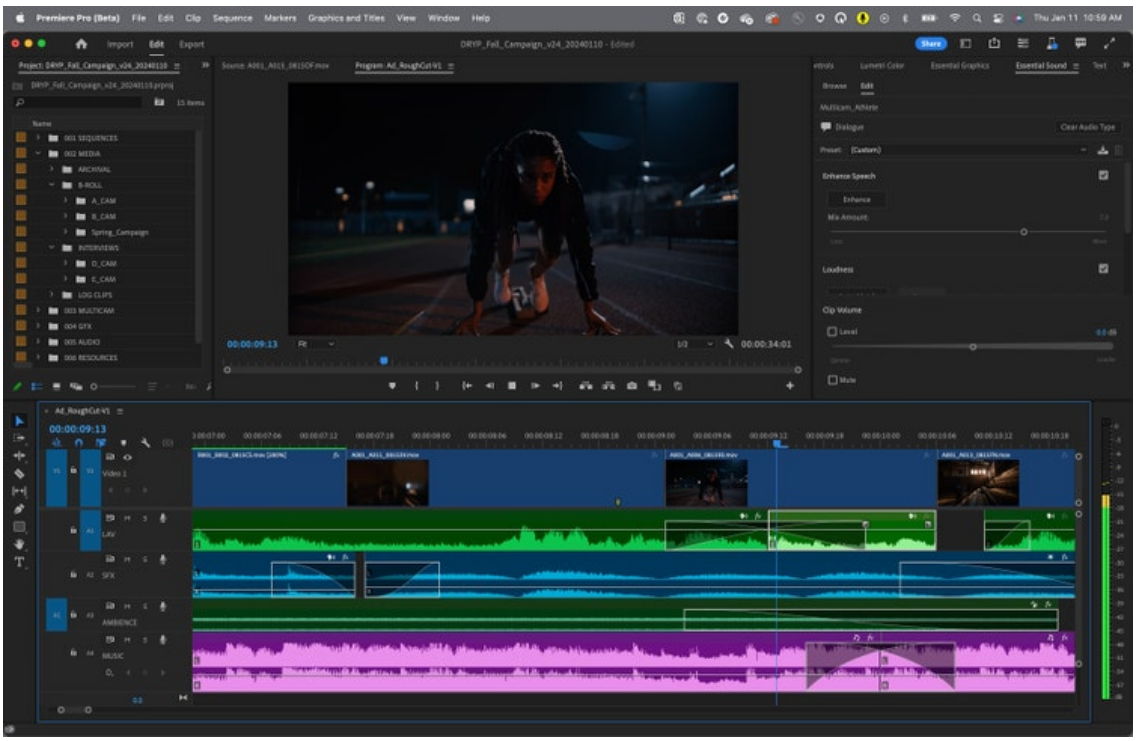

Рисунок 1.7 – Інтерфейс Adobe Premiere Pro

Завдяки великій кількості доступних ефектів, Adobe Premiere Pro[7] дозволяє користувачам максимально налаштовувати відео та аудіо, створюючи професійний кінцевий продукт, який задовольняє вимоги навіть найвибагливіших спеціалістів у галузі монтажу.

1967: Retro Filters & Effects – це популярний додаток для редагування фотографій, який пропонує широкий набір фільтрів і ефектів, натхненних естетикою 60-х і 70-х років. Програма створена для того, щоб користувачі могли легко додавати вінтажні стилізації до своїх фотографій, створюючи унікальні образи з ретро-атмосферою. Однією з основних особливостей 1967 є його колекція фільтрів, що відтворюють вигляд старих плівкових фотографій, зі знебарвленими тонами, м'якими кольорами і характерною зернистістю.

Рисунок 1.8 – Інтерфейс 1967: Retro Filters & Effects

Додаток пропонує простий інтерфейс, який дозволяє швидко застосовувати різні ефекти та редагувати зображення, додаючи знебарвлення, світлові витоки та віньєтування, що допомагають створити автентичний вигляд старих фотографій. Користувачі можуть також налаштовувати інтенсивність фільтрів для досягнення бажаного результату, що робить процес редагування дуже інтуїтивним і швидким.

Завдяки цьому додатку, кожен може легко перетворити свої сучасні знімки на стильні ретро-фотографії, що нагадують зображення з архівів 60-х років. 1967: Retro Filters & Effects ідеально підходить для тих, хто хоче додати своїм фотографіям вінтажного шарму[8] без складних налаштувань чи професійних знань в області фотографії.

Hipstamatic – це один із перших додатків для мобільної фотографії, який здобув популярність завдяки своїй здатності створювати вінтажні фотографії, що імітують класичні плівкові камери. Додаток надає користувачам можливість вибирати різноманітні "об'єктиви", "плівки" та "спалахи", які впливають на те, як виглядатиме кінцеве фото. Кожна комбінація створює унікальні ефекти, додаючи зернистість, віньєтування, зміщення кольорів або знебарвлені тони, що відтворюють автентичну естетику старих плівкових фотографій.

Рисунок 1.9 – Інтерфейс Hipstamatic

Однією з ключових особливостей Hipstamatic[9] є його гнучкість: користувачі можуть експериментувати з різними налаштуваннями ще до того, як зроблять знімок, що надає відчуття роботи з фізичною плівковою камерою. Крім того, додаток має колекцію класичних фільтрів і стильних ефектів, що допомагають створити атмосферу 60-х, 70-х і 80-х років. Hipstamatic також дає можливість купувати додаткові пакети з новими ефектами й фільтрами для ще ширших можливостей творчості.

Ще одним важливим аспектом Hipstamatic є його інтеграція з соціальними мережами, що дозволяє користувачам легко ділитися своїми ретро-знімками з широкою аудиторією. Це робить додаток не лише інструментом для створення зображень, але й платформою для творчих експериментів, де користувачі можуть черпати натхнення один у одного. Завдяки цьому, Hipstamatic став частиною сучасної культури цифрової фотографії, поєднуючи в собі ностальгію за аналоговою епохою з можливостями миттєвого поширення та обміну у цифрову еру.

Таблиця 1.1 – Порівняння застосунків

Критерій	Adobe Premier Pro	1967: Retro Filters & Effects	Hipstamatic	Власний додаток
Кількість фільтрів та ефектів	+	-	-	+
Простота використання	-	+	+	+
Підтримка користувацьких налаштувань фільтрів	+	-	+	+
Швидкість роботи	-	+	-	+
Доступність на різних платформах	+	-	-	+
Сумарна кількість балів	3	2	2	5

Таблиця порівняльних характеристик показала, що розробка програмного продукту є доцільною. В результаті отримаємо продукт, що покриває недоліки існуючих рішень та забезпечує кращу, у порівнянні з аналогами, ефективність.

1.3 Аналіз методів розв’язання поставленої задачі

Для розв'язання задачі створення вінтажних ефектів VHS-плівки у цифрових відео існує кілька підходів, кожен із яких має свої переваги та

недоліки. Одним із найпростіших підходів є використання попередньо створених фільтрів або шаблонів, які імітують VHS-ефекти[10], такі як шум, лінії сканування, колірні спотворення та збої в кадрі. Цей метод дає змогу швидко досягти бажаного результату, але обмежений тим, що фільтри мають статичний характер і можуть виглядати однаково при повторному застосуванні.

Більш складним, але гнучким рішенням є програмна реалізація процедурних ефектів, де кожен з елементів (шум, розмитість, візуальні артефакти) генерується алгоритмічно. Такий підхід дозволяє створювати динамічні, унікальні ефекти, адаптовані до різних відеоматеріалів, і забезпечує більший контроль над налаштуваннями кожного з параметрів. Це дає змогу імітувати не лише загальні VHS-ефекти, а й специфічні недоліки конкретних пристроїв, таких як старі відеомагнітофони.

Ще одним підходом є використання технологій машинного навчання для аналізу та відтворення VHS-стилю. Наприклад, нейронні мережі можуть бути навчені на реальних відеозаписах з VHS для створення фільтрів, які точно відтворюють артефакти плівки. Такий підхід дозволяє автоматично підлаштовувати ефекти під різні умови відео, але потребує значних обчислювальних ресурсів та навчальних даних.

Для вирішення задачі створення ефектів VHS було обрано метод програмної генерації процедурних ефектів. Він дозволяє гнучко налаштовувати параметри і створювати більш автентичні та унікальні ефекти, що відповідають оригінальній естетиці VHS.

1.4 Постановка задачі для створення VHS-ефектів

Проаналізувавши питання розробки систем для відтворення ефектів пошкодження відео у стилі VHS, було визначено завдання, які необхідно виконати для розробки програмного додатку:

- визначити найбільш ефективний підхід до симуляції візуальних артефактів, характерних для відео VHS (шум, перешкоди, спотворення кольорів, лінії затримки тощо);
- створити алгоритм генерації та накладання ефектів, притаманних відео з касет VHS;
- розробити плагін з використанням OpenFx для інтеграції VHS-ефектів у професійні відеоредактори;
- реалізувати параметри керування інтенсивністю та типами дефектів для точного налаштування ефекту;
- провести тестування програмного додатку.

1.5 Висновки

У першому розділі було розглянуто стан питання створення ефектів у стилі VHS у цифровому відео на сьогоднішній день.

Проаналізовано існуючі аналоги програм для створення візуальних дефектів, характерних для VHS-касет, та проведено їх порівняння з розроблюваним продуктом. У результаті порівняння було доведено доцільність розробки власного програмного рішення для створення VHS-ефектів.

Проведено аналіз існуючих підходів до симуляції артефактів старих відео, таких як шум, дисторсія зображення, нестабільність сигналу тощо. У результаті аналізу було обрано комбінацію методів накладення цифрових спотворень та шумів із можливістю керування параметрами ефекту.

Сформульовано основні завдання, які необхідно виконати для розробки плагіна на основі OpenFx[11].

2 РОЗРОБКА МЕТОДІВ І ЗАСОБІВ РЕАЛІЗАЦІЇ СИМУЛЯЦІЇ АНАЛОГОВОГО ШУМУ ТА АРТЕФАКТІВ

2.1 Вибір методів отримання та збереження зображення

Аналогові методи отримання зображення передбачають використання фізичних процесів для фіксації та передачі візуальної інформації, що відображає об’єкти в реальному світі. Класичні приклади таких методів включають фотографію та кінематографію на плівці, де вхідні дані (світло, що відбивається від об'єктів) потрапляють на світлочутливий матеріал, утворюючи зображення на поверхні. У цьому процесі світлові хвилі перетворюються на хімічні або електричні сигнали, які можуть бути закріплені на матеріалах, таких як фотоплівка або аналогові кінокамери.

Для створення фільмів застосовували кіноплівку, на якій кожен кадр послідовно записувався, створюючи ілюзію руху при швидкій зміні кадрів під час відтворення. Аналогові телевізори також відображали зображення через сканування електронним променем по екрану, що передавало яскравість і кольори зображення. Важливо зазначити, що ці методи вимагали високої точності, адже фізичні властивості матеріалів могли впливати на якість зображення: наприклад, зернистість плівки чи відбитки на папері визначали чіткість і деталізацію.

Однією з проблем аналогових методів є обмежена здатність зберігати та передавати інформацію без втрати якості[12]. Наприклад, кожне відтворення плівкового фільму може призвести до зносу матеріалу та втрати деяких деталей зображення, тоді як цифрові формати зберігають дані без змін навіть після багаторазового копіювання. Однак аналогові технології здатні відтворювати м’якіші градації кольорів і контрастів завдяки безперервному характеру аналогового сигналу, що унікальним чином впливає на візуальне сприйняття.

Технічна особливість аналогових методів полягає в тому, що зображення зберігається не у вигляді дискретних даних, як у цифрових технологіях, а як

неперервний сигнал. Таким чином, аналогові методи, такі як фотографія на плівці чи кінематографічна зйомка, здатні передавати унікальні деталі та відтінки зображення, забезпечуючи природну передачу кольорів і текстур.

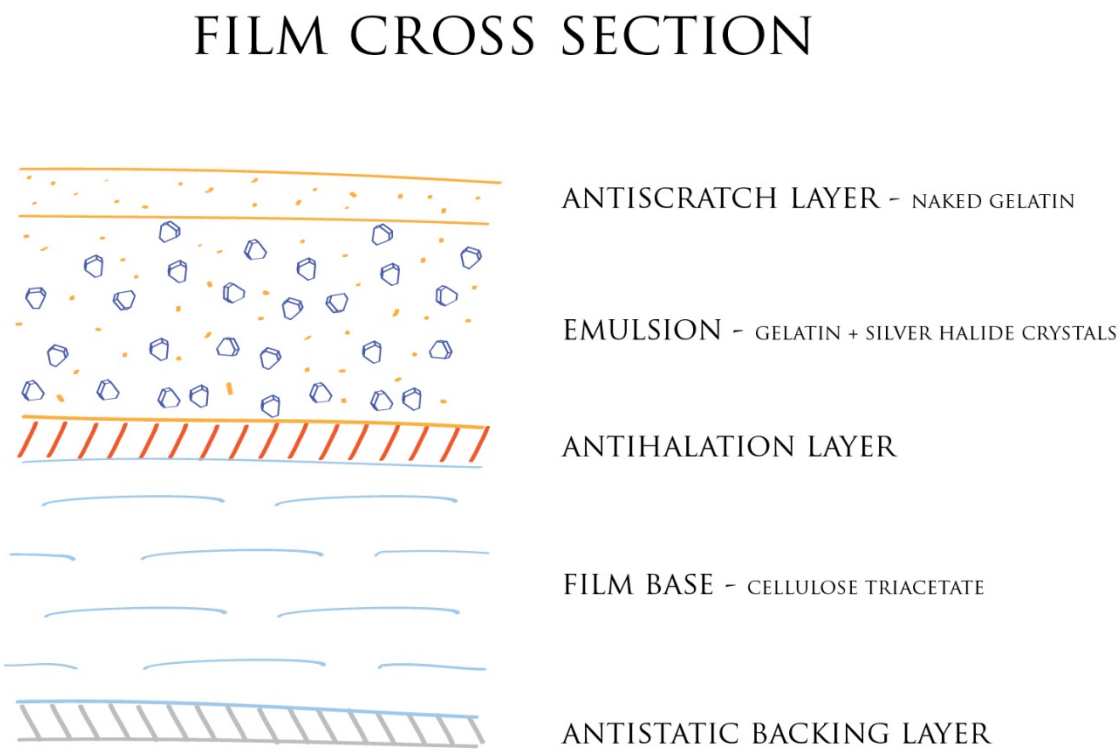

Рисунок 2.1 – Схема шарів плівки

Існують також аналогові методи такі як VHS-касети, що свого часу були революційними у збереженні та відтворенні відео, проте вони мають низку суттєвих недоліків, що обмежують їхню сучасну ефективність. Одним із ключових обмежень VHS є те, що інформація зберігається на магнітній стрічці у вигляді аналогового сигналу, через що якість відео знижується при кожному перегляді або копіюванні касети[13].

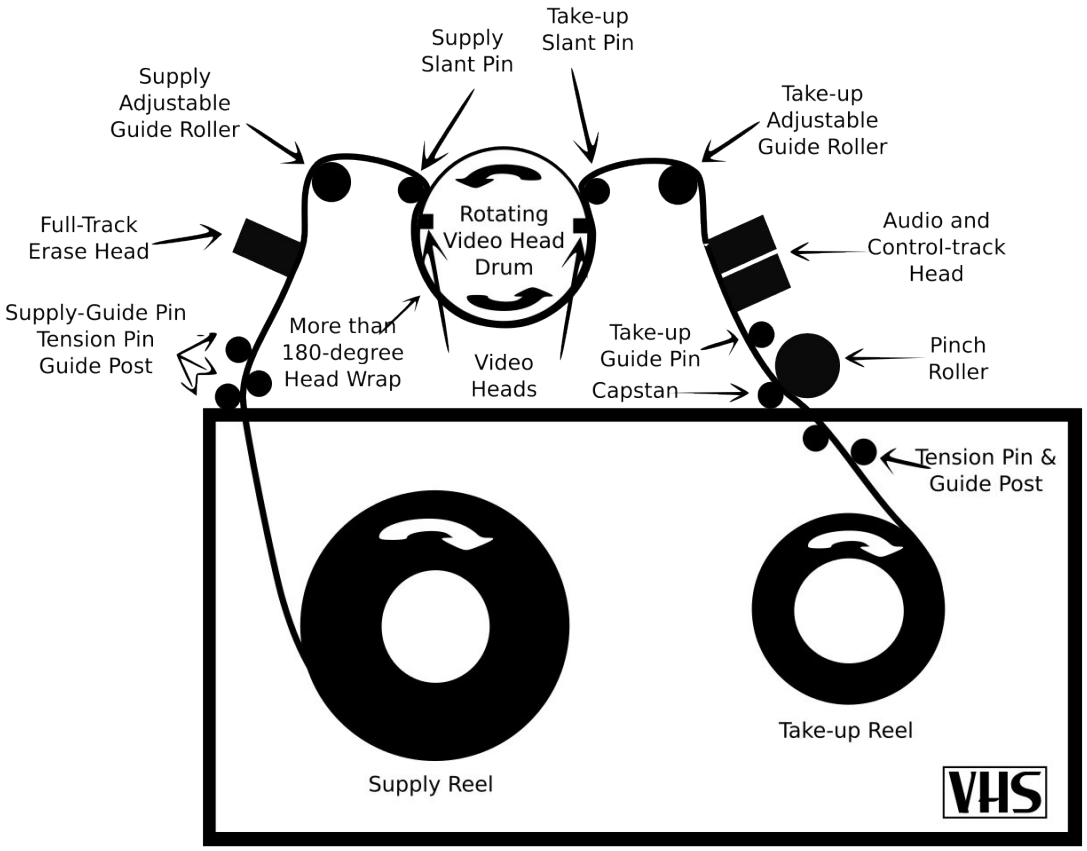

Рисунок 2.2 – Схема роботи касети

Також однією із найрозповсюдженіших проблем аналогових носіїв є віньєтування.

Віньєтування — це ефект, при якому зменшується яскравість або насиченість кольорів у краях зображення, порівняно з його центром. У контексті аналогових технологій, таких як VHS або кіноплівка, цей ефект проявляється у вигляді затемнення або "розмиття" периферійних ділянок кадру. Віньєтування може виглядати природно в деяких художніх композиціях, але в аналогових відеосистемах воно зазвичай вважається недоліком, оскільки знижує якість і чіткість зображення.

Віньєтування на VHS-касетах часто виникає через обмеження оптичної системи камери, записуючого обладнання або навіть самого телевізійного сигналу. Аналогова камера зазвичай використовувала об'єктиви, які не завжди могли рівномірно передавати світло на всю поверхню світлочутливого елемента.

У результаті центральна частина кадру отримувала більше світла, ніж краї, що створювало ефект затемнення.

Рисунок 2.3 – Приклад фото із негативним віньєтуванням

На кіноплівці віньєтування виникало через властивості оптичних лінз, які спричиняли нерівномірне відбиття або заломлення світла. Це було особливо помітно у старих або дешевих об'єктивах, які використовувалися у масових камерах для запису домашніх відео. На плівці цей ефект міг також посилюватися через механічні фактори, такі як нерівномірний рух плівки у проекторі або камері.

На VHS віньєтування також посилювалося через обмеження магнітної записуючої системи. Магнітна стрічка не завжди рівномірно захоплювала інформацію по всій ширині, особливо ближче до її країв. Це могло бути спричинено магнітними шумами або неточностями в налаштуванні головки запису та відтворення.

Крім того, аналогова система передачі відеосигналу мала низьку роздільну здатність і частково розподіляла інформацію про яскравість і кольори. Через це краї кадру зазнавали більшої деградації, що візуально виглядало як затемнення.

Віньєтування у VHS і плівкових системах є недоліком через те, що воно погіршує сприйняття зображення. Краї кадру стають менш деталізованими, втрачають яскравість, а це створює враження неякісного відео. У контексті професійного відео та кіно це могло знижувати естетичну цінність матеріалу, оскільки об'єкти, розташовані ближче до країв, виглядали менш чітко.

Окрім того, віньєтування могло впливати на кольоровий баланс, адже затемнення країв часто супроводжувалося зміною відтінків через нерівномірне накладення кольорів. У цифрових системах ці недоліки можна виправити на стадії обробки кадрів, але у випадку VHS та плівки це було майже неможливо без втрати додаткової якості.

Попри недоліки, віньєтування часто асоціюється з "аналоговим шармом". У сучасному цифровому мистецтві цей ефект іноді спеціально додається для створення ретро-атмосфери, яка нагадує про епоху VHS або кіноплівки. Таким чином, віньєтування є водночас історичним недоліком і елементом естетики минулих поколінь.

Зараз же існують більш досконалі методи збереження та обробки відео, такі як цифрові формати, дозволяють зберігати дані без втрати якості, тоді як аналогові формати, включаючи VHS, зберігають зображення у вигляді безперервного сигналу, що вразливий до шумів і фізичного зносу стрічки. VHS-касети використовують магнітний запис для кодування відеоінформації, що дозволяє створювати неперервне зображення, але також обмежує роздільну здатність і точність кольорів. Це означає, що деталі та контрастність відео можуть бути помітно нижчими, особливо при повторному відтворенні запису.

Сучасні методи збереження відеоданих використовують цифрові алгоритми та методи зберігання, що дає змогу отримати векторне представлення кадрів та враховувати кольорові й просторові особливості окремих пікселів,

забезпечуючи високу точність зображення. Цифрові формати дозволяють відтворювати та зберігати точні копії зображень, тоді як аналогові VHS-касети з часом втрачають як деталі, так і насиченість кольорів. Завдяки можливості збереження даних у вигляді послідовностей кадрів з постійною якістю, цифрові формати значно перевершують VHS за надійністю та довговічністю, що особливо важливо для архівування та професійного використання відеоматеріалів.

Аналіз показує, що хоча VHS-касети здатні забезпечити цілісне відеозображення та зберегти важливі моменти, цей метод є менш придатним для сучасних завдань через обмеження, пов'язані з роздільною здатністю, зношуванням та чутливістю до зовнішніх впливів.

Цифрові методи отримання та зберігання відеозображення дозволяють досягти значно більшої точності та надійності порівняно з аналоговими системами. Цифровий підхід базується на використанні числового представлення зображення у вигляді послідовностей пікселів[14], де кожен піксель має чітке значення яскравості та кольору, зберігаючись у стандартизованих форматах, як-от JPEG, PNG, або для відео — MPEG, AVI тощо. Завдяки цьому цифрове відео зберігає свою якість незалежно від кількості копій, не піддаючись деградації під час відтворення, що є значною перевагою перед аналоговими касетами.

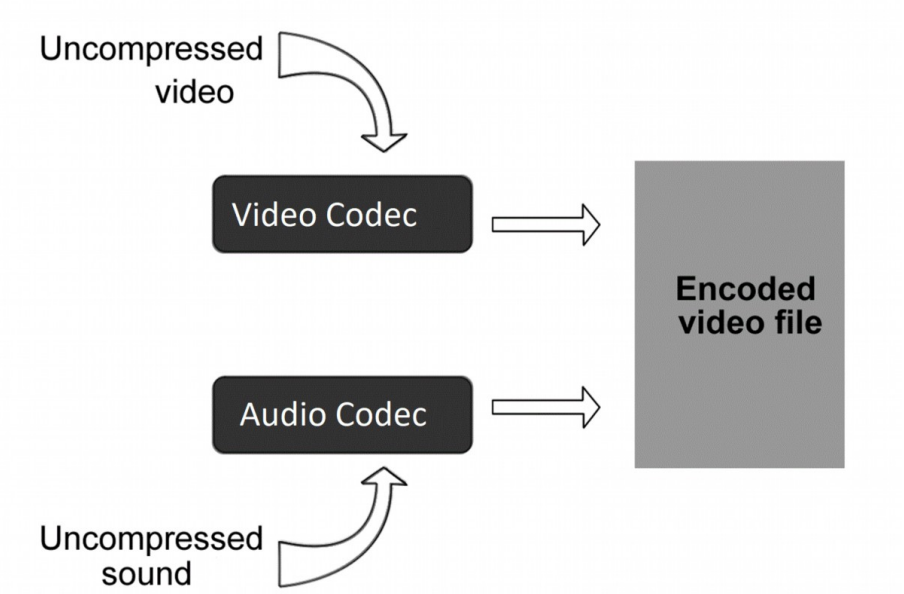

Рисунок 2.4 – Схема компресії цифрового відео

На сьогодні цифрові системи запису здатні зберігати велику кількість метаданих, як-от інформація про колірну глибину, частоту кадрів, роздільну здатність, що значно розширює можливості для подальшої обробки та редагування відео. Сучасні алгоритми стиснення[15], такі як H.264 та H.265, забезпечують збереження високої якості зображення при значному зменшенні обсягу даних, що робить зберігання відео в цифровому форматі економічно ефективним.

Разом із цим стрімкий розвиток штучного інтелекту дозволяє його застосування у цифрових технологіях відеозапису, що дозволяє штучний обробляти та покращувати якості зображення навіть в реальному часі.

Однією із таких технологій є DLSS — е технологія від NVIDIA, що використовує штучний інтелект для підвищення якості зображення та продуктивності у відеоіграх і графічних додатках.

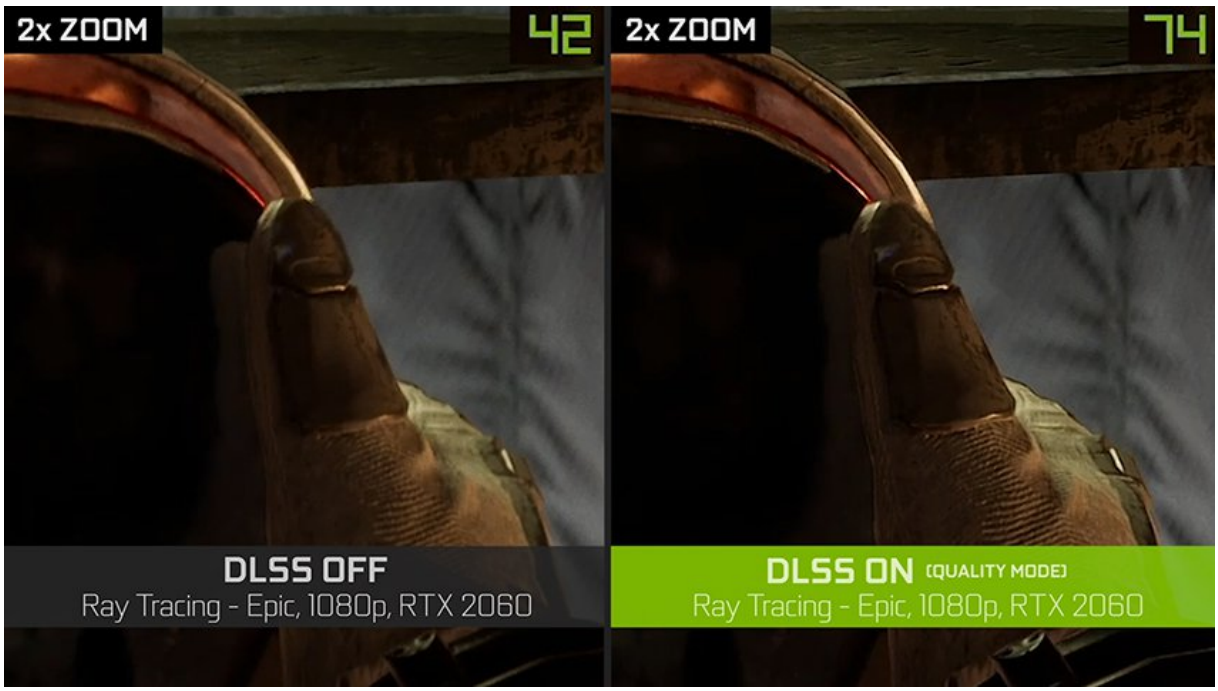

Рисунок 2.5 – Приклад роботи DLSS

DLSS[16] працює за принципом масштабування зображення: гра рендериться у нижчій роздільній здатності, ніж цільова, а потім спеціальні алгоритми на основі глибинного навчання інтерполюють його до вищої якості. У цьому процесі використовується штучна нейронна мережа, яка натренована на мільйонах високоякісних зображень, що дозволяє їй відтворювати точні деталі та відновлювати різкість на рівні зображення у більшій роздільній здатності.

DLSS отримує базові дані від графічного процесора[17], обробляє їх на рівні окремих пікселів, а потім доповнює, враховуючи інформацію з попередніх кадрів. Завдяки цьому технологія забезпечує плавне відтворення ігрових сцен навіть на високих налаштуваннях графіки, не завантажуючи занадто багато обчислювальних ресурсів.

Це особливо корисно для рендерингу складних сцен із великою кількістю дрібних деталей, таких як текстури або тіні, де класичні методи масштабування призвели б до розмитих або нечітких зображень.

Ключовою перевагою DLSS є підвищення продуктивності та FPS (кількості кадрів в секунду) без втрати якості, що робить цю технологію особливо корисною для ресурсомістких ігор та додатків віртуальної реальності. Нейронна мережа DLSS постійно оновлюється та вдосконалюється, що дозволяє з часом ще краще адаптувати її для нових ігор і ситуацій, розширюючи функціональність і знижуючи вимоги до системних ресурсів.

DLSS є новою технологією масштабування зображень, розробленою NVIDIA, що поєднує штучний інтелект і графічну обробку для покращення якості виведеного зображення у реальному часі. В її основі лежить використання спеціальних тензорних ядер, вбудованих у сучасні графічні процесори серії RTX[18]. Ці ядра оптимізовані для виконання матричних обчислень, що необхідні для роботи нейронних мереж. Завдяки цьому DLSS може зменшувати навантаження на основні обчислювальні потужності, делегуючи складні обчислення окремим апаратним блокам, зберігаючи при цьому плавність і стабільність зображення.

Технологія працює за допомогою комплексної взаємодії кількох компонентів. Вхідні дані включають рендеринг базового кадру в низькій роздільній здатності, буфер глибини, вектор руху (Motion Vectors), який визначає зміщення об’єктів між кадрами, та контекстні дані, що описують особливості сцени. Алгоритми на основі глибинного навчання аналізують ці дані, передбачають, які деталі потрібно відновити, і відтворюють зображення високої роздільної здатності. Використання векторів руху дозволяє нейронній мережі відслідковувати зміни між кадрами та уникати появи артефактів, таких як розмиття або "хвилі" на рухомих об’єктах.

DLSS також інтегрує процес згладжування під час масштабування, що забезпечує усунення ефектів "зубців" на краях об’єктів, типових для стандартного рендерингу у низькій роздільній здатності. Це досягається шляхом прогнозування і корекції піксельних даних, які можуть бути втрачені або спотворені через брак деталізації. У результаті, технологія не тільки підвищує

чіткість зображення, але й створює візуальний ефект, що наближається до нативної роздільної здатності.

Ефективність DLSS залежить від якісного навчання нейронної мережі, що відбувається у спеціалізованих центрах NVIDIA. Для цього використовуються тисячі годин тренувань, де модель порівнює результати рендерингу низької якості з еталонними зображеннями високої роздільної здатності. Це дозволяє моделі поступово вивчати закономірності у даних та знаходити оптимальні способи їх відновлення. У фінальній версії DLSS впроваджується у графічний драйвер, де вона працює у реальному часі, адаптуючись до потреб конкретної гри або застосунку.

Сучасні версії DLSS йдуть ще далі, додаючи технологію генерації проміжних кадрів. Це дозволяє значно підвищити FPS[19], адже не всі кадри створюються шляхом класичного рендерингу. Замість цього нейронна мережа генерує нові проміжні кадри, спираючись на вектори руху та аналіз існуючих кадрів. У поєднанні з розширеними алгоритмами для обробки світла, тіней і текстур, DLSS перетворюється на потужний інструмент, що дозволяє геймерам насолоджуватися винятковою якістю графіки навіть на середніх чи бюджетних апаратних платформах.

Також спеціалізовані алгоритми розпізнавання та покращення зображення можуть відновлювати деталі, які були втрачені під час запису або стискання відео, адаптуючи зображення для різних пристроїв. Відеодані також можуть бути збережені у форматах, що дозволяють швидке сортування та пошук за кадрами або сценами, що полегшує роботу з великими обсягами відеоінформації.

Зважаючи на зростаючу складність сучасних завдань обробки та зберігання відео, цифровий підхід є найбільш ефективним та універсальним. На відміну від аналогових систем, де кожне відтворення знижує якість зображення, цифрові методи забезпечують постійну якість незалежно від кількості переглядів та зберігають можливість обробки відеоматеріалу без втрат.

Таким чином, цифрові технології є оптимальним вибором для роботи з відеозображенням, що дозволяє максимально зберегти початкову якість та значно спрощує роботу з відеоматеріалами.

2.2 Розробка алгоритмів роботи системи

Першим етапом розробки є проектування її загального алгоритму роботи. Алгоритм роботи полягає у перетворенні цифрового відеосигналу у сигнал схожий до аналогового із домішками хроматичних аберацій шумів та неточностей. вирішено використовувати конвертацію кольорового простору, що дозволяє імітувати характерний вигляд аналогового сигналу.

Перетворення RGB в YIQ дозволяє поділити сигнал на яскравість і колірні компоненти, що зручно для накладення шумів та кольорових спотворень Після перетворення кадру додається випадковий шум, імітуючи цифрові артефакти, а також зернистість, що створює ефект нестабільності аналогового сигналу. На основі цих елементів налаштовується яскравість і контраст зображення, надаючи відео м’якого затемнення і відтворюючи атмосферу низької якості запису.

На наступному етапі додаються скануючі лінії[20], характерні для записів з VHS-касет, і створюються випадкові кольорові зсуви, що надає кадрам ефект хроматичної аберації. Після цього додаються статичні перешкоди для створення ілюзії шумів, що інколи з’являються через слабкість сигналу. ля цього Y-компонента залишається незмінною, тоді як I та Q компоненти зазнають невеликого випадкового зсуву.

Рисунок 2.6 – Алгоритм роботи системи створення VHS-ефектів

Наступний етап – розробка алгоритму обробки запитів. Першим етапом алгоритму є отримання вхідних даних із запиту. Далі необхідно виконати обробку даних перед передачею їх до мережі на аналіз. Наступний етап – обробка даних мережею. Дані, отримані від мережі також необхідно перетворити у більш зручний для сприйняття формат. Останній етап – надсилання відповіді на запит. Блок-схема алгоритму зображена на рисунку 2.6.

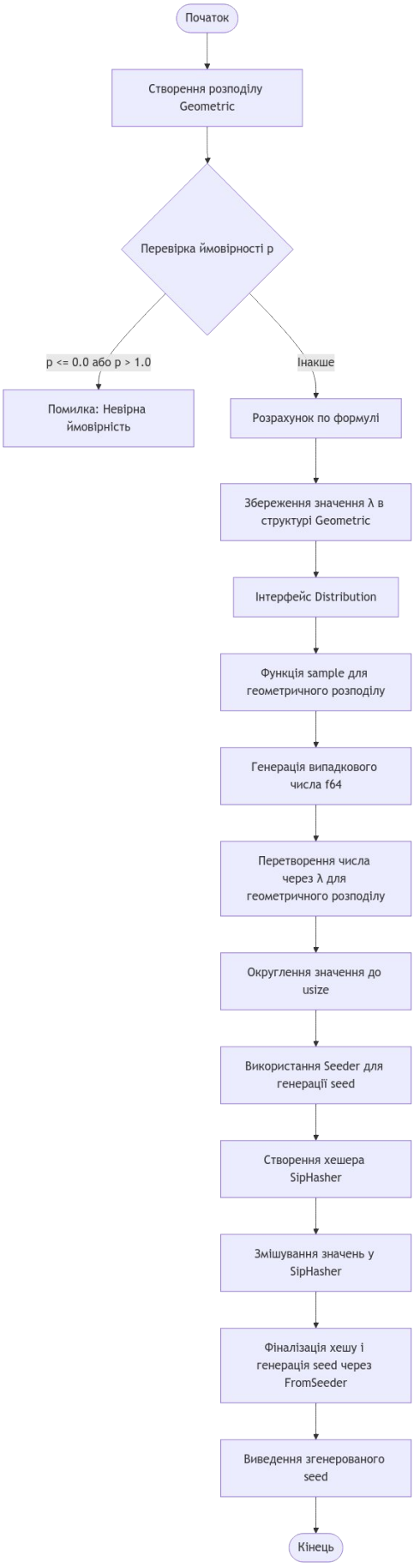

Рисунок 2.7 – Алгоритм роботи створення випадкової конфігурації ефектів

2.3 Розробка алгоритму ініціалізації плагіну

Через особливості розробки система зав’язана на обовязковому використанні відеоредактору. Тому для створення плагіну було потрібно використати вбудовану бібліотеку OpenFX. Застосування OpenFX дозволило спростити взаємодію між кодом плагіна та середовищем редактора, що скоротило час на інтеграцію і тестування. Завдяки цим підходам плагін забезпечує плавну роботу і надійне відображення ефектів у різних версіях редактору.

OpenFX надає гнучкий набір інструментів і функцій для доступу до відеоряду, обробки зображень і забезпечення взаємодії з інтерфейсом користувача. OpenFX включає інтерфейси для роботи з ефектами в режимі реального часу, що дозволяє обробляти кадри відео безпосередньо під час редагування. Це дає можливість створювати інтуїтивно зрозумілі та ефективні інструменти для користувача, який може одразу переглядати зміни у відеоряді.

Блок-схема алгоритму зображена на рисунку 2.8.

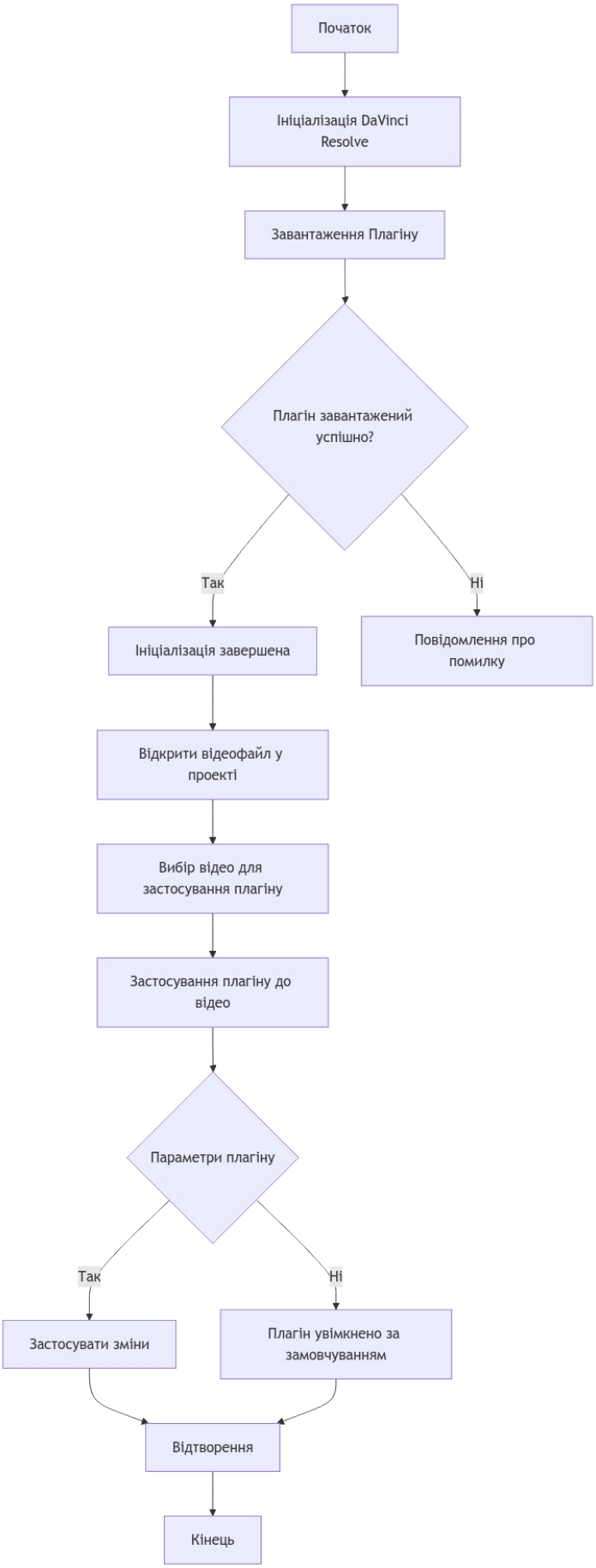

Рисунок 2.8 – Блок-схема ініціалізації плагіну

Основні етапи ініціалізації плагіну наступні:

1. Початок процесу.
2. Запуск та ініціалізація відеоредактору.
3. Завантаження плагину із дерикторії.
4. Успішне завантаження плагіну.
5. Помилка ініцалізації плагину.
6. Відкриття файлу на доріжці таймлайну.
7. Вибір часу відтворення відеофрагменту.
8. Застосування плагину із вікна ефектівна відеофрагмент.
9. Застосування змін у вкладці налаштувань плагіну.
10. Застосування стандартних налаштувань плагіну.
11. Відтворення відеофрагменту із застосованим плагіном.
12. Завершення.

Першим етапом обробки для створення ефекту VHS є поділ вхідного відео на окремі кадри. Цей крок є базовим і дозволяє опрацювати кожен кадр індивідуально, забезпечуючи точність у застосуванні ефектів. Окремий аналіз кадрів необхідний через те, що вінтажні VHS-ефекти, як правило, впливають на структуру зображення в межах одного кадру, а не на відео в цілому.

Далі кожен кадр проходить процес розділення на поля, що відповідає інтерлейсованій структурі сигналу, характерній для аналогових відеосистем. Поле може містити парні або непарні рядки пікселів, що дозволяє відтворити горизонтальне зміщення або артефакти, характерні для VHS-записів. На цьому етапі важливо дотримуватися коректного порядку обробки полів, щоб уникнути помилок у відображенні відео.

Наступним кроком є перетворення кольорового простору кадру з RGB у YIQ. Це перетворення виконується для точнішого управління кольорами, адже аналогові VHS-записи зберігають колірні компоненти у вигляді яскравості (Y) і двох кольорових компонентів (I та Q). Такий підхід дозволяє окремо

маніпулювати яскравістю та кольорами для створення ефектів, як-от зміна насиченості, колірних шумів чи спотворень.

Для кожного пікселя в кадрі виконується нормалізація кольорових компонентів. Це забезпечує коректну обробку даних у різних форматах і дозволяє моделювати характерний "гул" кольорів або розмиття, притаманне VHS. Важливим моментом є симуляція випадкових шумів, що додаються до кольорових компонентів, які вносять ефект нестабільності та деградації якості.

Щоб підсилити ефект, застосовується низькочастотний фільтр (lowpass filter), що імітує втрату високочастотних деталей. Параметр фільтрування регулює рівень розмиття: при зниженні частотності кадр стає менш деталізованим, набуваючи характерного для VHS м'якого вигляду. Наприклад, зменшення параметра до 0.1 призводить до суттєвого розмиття, що додає реалістичності ефекту.

Після всіх обчислень кадр перетворюється назад з YIQ у RGB, а поля об'єднуються в один кадр. Для збереження стабільності додатково застосовується нормалізація яскравості та корекція шумів, щоб отримати якісну імітацію VHS.

У результаті роботи алгоритму на виході формується відеопотік із характерними артефактами VHS, такими як кольорові спотворення, шум, розмиття та зсув полів. Ці ефекти створюють відчуття вінтажного відео, яке імітує властивості старих аналогових записів.

2.5 Висновки

У другому розділі було проведено аналіз методів та засобів реалізації створення VHS-ефектів. Проведено порівняння різних методів отримання вінтажних ефектів.

Розроблено загальні алгоритми та методи роботи.

Розроблено алгоритми роботи системи та ініціалізації.

Розроблено та проілюстровано метод роботи засобу створення VHS-ефектів у цифровому відео.

Обрано бібліотеку OpenFx для розробки та інтеграції плагіну у середовище відеоредакторів.

3 РОЗРОБКА ПРОГРМАНОГО ЗАСОБУ ДЛЯ СТВОРЕННЯ ВІНТАЖНИХ ЕФЕКТІВ VHS-ПЛІВКИ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного засобу.

Вибір мови програмування для розробки залежить від різних факторів, таких як: продуктивність, безпека, гнучкість, підтримка спільноти та інші технічні вимоги. У контексті сучасних тенденцій, Rust виглядає найбільш перспективною завдяки своїм унікальним характеристикам, які вирізняють її серед таких популярних мов.

Rust – мова програмування системного рівня, яка орієнтована на забезпечення високої продуктивності та безпеки пам’яті без необхідності використання автоматичного збирача сміття. Її основними особливостями є:

- Статична типізація;
- Система позичання і володіння пам’яттю, що виключає витоки пам’яті та помилки, пов'язані з доступом до пам'яті;
- Сувора обробка помилок через Result і Option;
- Висока підтримка конкурентності через безпечні багатопоточні механізми;
- Повна кросплатформеність.

Rust активно використовується для розробки системних додатків, веб-сервісів, та ігрових рушіїв, адже її ключові переваги дозволяють поєднати продуктивність C++ з безпекою сучасних мов високого рівня. Завдяки системі управління пам'яттю на основі позичання й володіння, Rust виключає класичні проблеми, з якими стикаються розробники на C++: витоки пам’яті, сегментаційні помилки і некоректні вказівники. Це робить її ідеальною для проєктів, де потрібно дбати про продуктивність і надійність одночасно.

Крім того, Rust підтримує безпечну конкурентність, дозволяючи ефективно працювати з багатопоточними процесами без ризику виникнення race

condition або deadlock, що є критично важливим для сучасних додатків. У Python, наприклад, це не так легко досягається через наявність Global Interpreter Lock (GIL), що обмежує багатопоточну продуктивність.

Екосистема Rust постійно зростає, пропонуючи безліч бібліотек для різноманітних задач – від роботи з мережами до веброзробки. Завдяки активній спільноті та постійній підтримці, Rust стає чудовим вибором для довготривалих проектів, де важлива стабільність і можливість обслуговування.

C++ – мова програмування системного рівня, яка забезпечує високу продуктивність і максимальний контроль над ресурсами системи. Вона широко використовується для створення програмного забезпечення, що потребує прямого доступу до апаратного забезпечення або суворого контролю над пам’яттю. Основні особливості C++ включають:

- Статична типізація;
- Ручне управління пам’яттю через виділення й звільнення ресурсів;
- Підтримка об’єктно-орієнтованого, процедурного та шаблонного програмування;
- Можливість написання як високорівневого, так і низькорівневого коду;
- Висока продуктивність завдяки прямому доступу до апаратного забезпечення.

C++ використовується в розробці програмного забезпечення для операційних систем, ігрових рушіїв, драйверів, компіляторів і фінансових додатків. Завдяки своїй багатофункціональності та потужності, мова дозволяє створювати програми з високою швидкістю виконання і гнучкістю, що робить її вибором для критично важливих задач.

Окрім того, конкурентність у C++ потребує додаткових зусиль для забезпечення безпеки багатопотокових операцій, що може збільшувати складність розробки. Хоча C++ пропонує потужні інструменти для багатопотоковості, забезпечення безпеки при цьому лягає на плечі розробника.

Враховуючи всі вище перечисленні особливості та переваги, мова Rust є найбільш доцільною для розробки засобу.

.

3.2 Вибір середовища розробки

Середовище розробки — це програмний інструмент, що забезпечує розробнику зручність написання коду, налагодження, тестування та запуску додатків. Існує безліч різних середовищ розробки, кожне з яких має свої особливості. Розглянемо декілька з них.

Visual Studio Code — безкоштовний редактор від Microsoft, що підтримує велику кількість мов програмування та має безліч розширень для будь-яких потреб розробника. Це легкий, швидкий і гнучкий інструмент, який активно підтримується спільнотою. Він забезпечує зручну інтеграцію з Git, можливості відладки та автоформатування, а також налаштовується під будь-які вимоги завдяки широкому спектру плагінів.

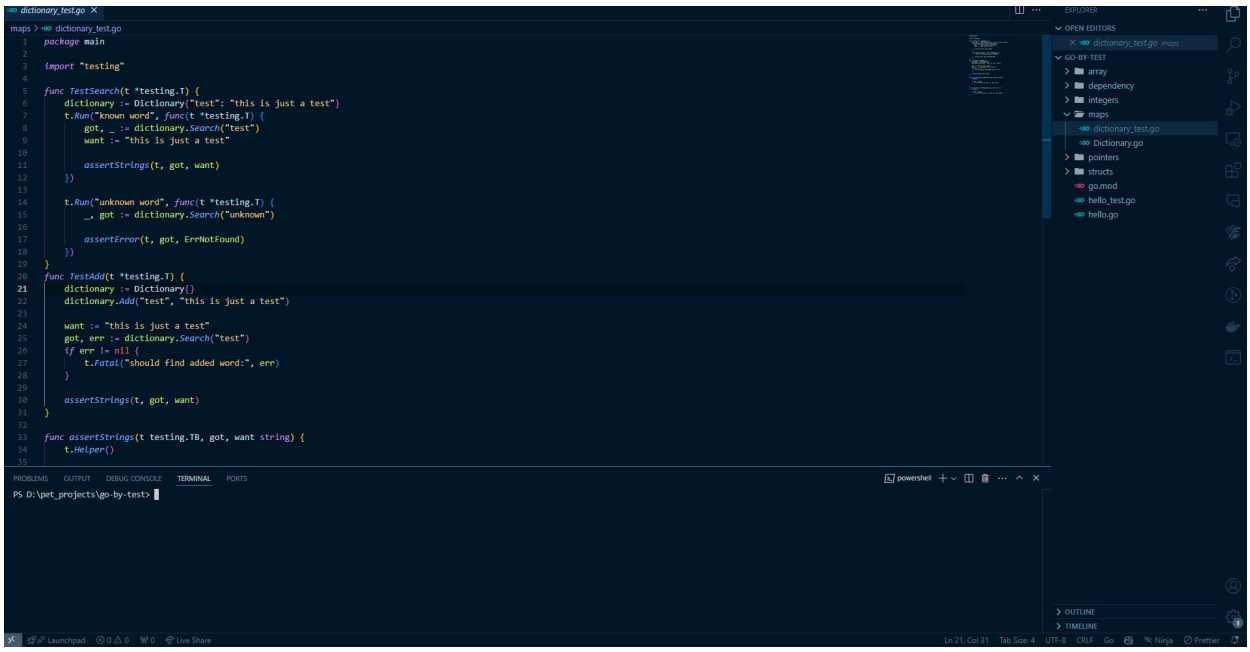

Рисунок 3.1 – Visual Studio Code

WebStorm — це потужне середовище розробки, яке спеціалізується на JavaScript і вебтехнологіях. Воно пропонує багатофункціональні інструменти

для інтеграції з фреймворками та бібліотеками, має потужну систему автозавершення коду та інтеграцію з системами контролю версій, такими як Git. Проте його використання потребує підписки.

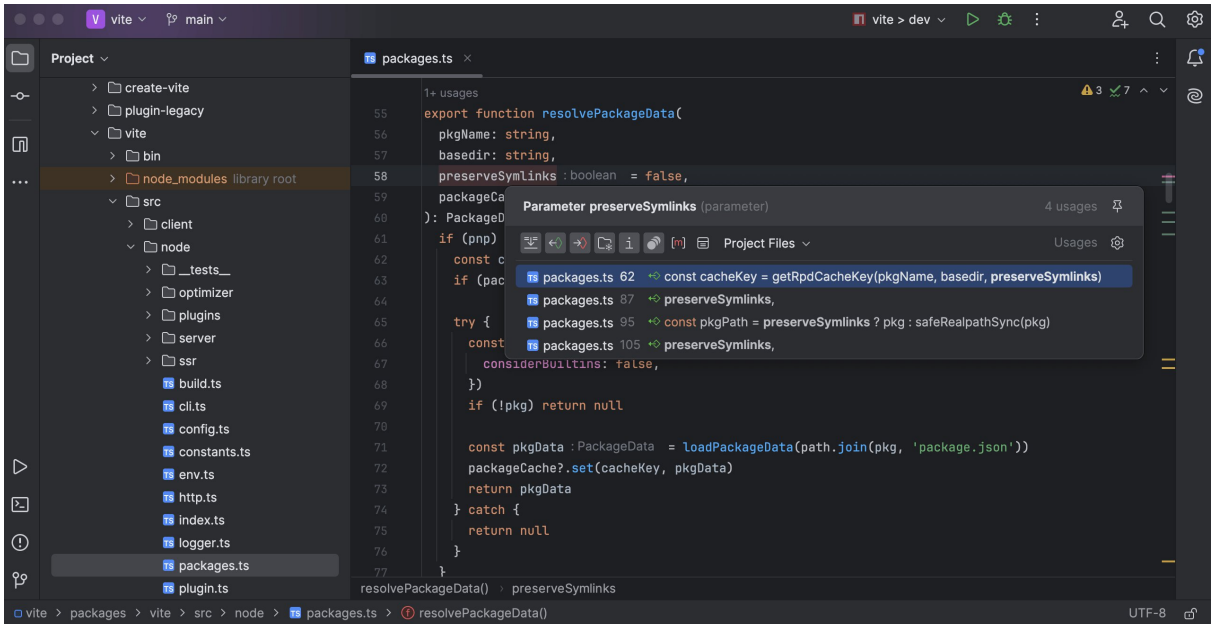

Рис 3.2 – JetBrains Webstorm

Vim — класичний текстовий редактор, який цінують за швидкість і легкість. Хоча Vim вимагає навчання для ефективного використання, після освоєння гарячих клавіш, цей редактор забезпечує надзвичайно швидке редагування коду. Однак, через складність налаштування та відсутність деяких сучасних інструментів для відладки та автозавершення, він поступається іншим середовищам у зручності.

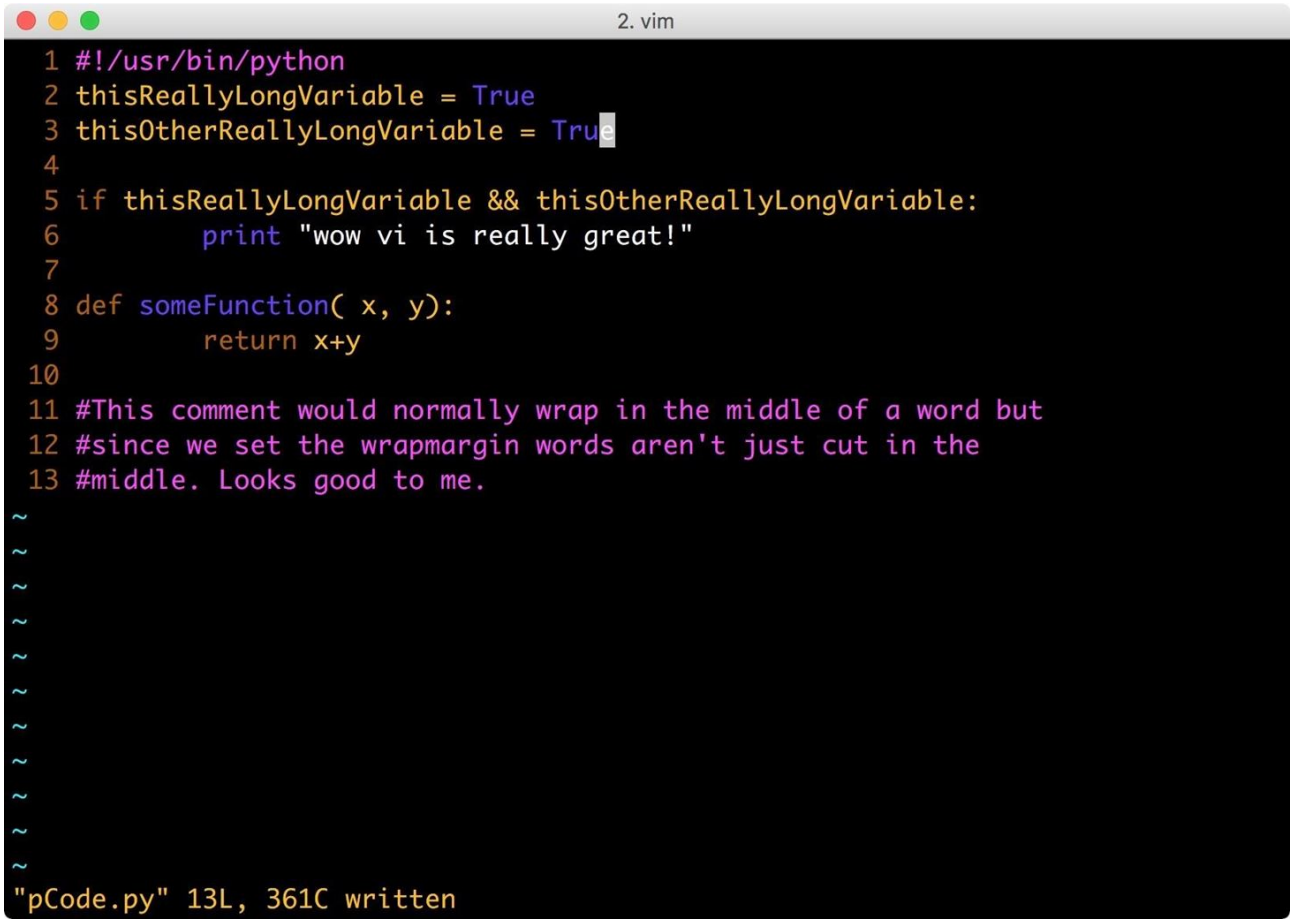

Рис 3.3 – Редактор тексту Vim

Підсумовуючи, Visual Studio Code було обрано як оптимальне середовище розробки завдяки його безкоштовності, підтримці великої кількості мов, можливості інтеграції з розширеннями, високій продуктивності, та зручній роботі з системами контролю версій.

3.3 Розробка модуля

Наступним етапом розробки додатку це розробка функціоналу створення ефектів. За допомогою алгоритмів викладених вище розробимо функції, що перетворять відеоряд у відеоряд із вінтажними ефектами.

Перший крок — перетворення значень кольорів між колірними просторами RGB та YIQ за допомогою функції rgb_to_yiq. Ця функція приймає масив із трьох компонентів кольору у форматі f32; 3, який відповідає за канали червоного, зеленого та синього кольорів. Після цього виконується множення

матриці YIQ_MATRIX на вектор із компонентами кольору, що дозволяє отримати нові значення у колірному просторі YIQ.

Далі слід застосувати функцію yiq_to_rgb, якщо потрібно перетворити кольори назад у формат RGB. Подібно до попереднього кроку, ця функція використовує спеціальну матрицю RGB_MATRIX для обчислення компонентів кольору за допомогою множення. Як результат, значення YIQ перетворюються назад у стандартний формат RGB, який зручно використовувати для візуалізації чи подальшої обробки зображень.

Наступний етап — налаштування полів кадру за допомогою структури YiqField, яка дозволяє вибрати один із кількох варіантів зчитування полів зображення: верхнє, нижнє, обидва поля, а також чергування між ними. Наприклад, вибір значення Upper дозволяє обробляти лише парні рядки, що імітує ефект інтерлейсингу.

Для коректного зберігання значень кольорів необхідно виконати нормалізацію різних форматів пікселів за допомогою трейтів, зокрема Normalize. Цей трейт визначає методи from_norm і to_norm, які перетворюють значення до нормалізованого формату f32, що є основою для обчислень у системі координат від 0 до 1. Наприклад, Normalize for u8 перетворює значення від 0 до 255 у нормалізований формат для полегшення подальшої обробки.

Для форматів зображень, що використовуються в Adobe After Effects, передбачено спеціальну обробку завдяки структурі AfterEffectsU16. Під час роботи з такими зображеннями значення кольорів варіюються в діапазоні від 0 до 32768, що потребує унікальної нормалізації. Реалізація методу to_norm для цього типу враховує максимальне значення в 32768, забезпечуючи коректне масштабування і перетворення кольорових даних для сумісності з After Effects.

На наступному етапі визначаємо порядок розташування пікселів у буфері за допомогою енумератора SwizzleOrder. Цей перелік задає формати піксельних даних, такі як Rgbx, Xrgb, Bgrx, і використовується для ефективного зберігання кольорових даних у форматах RGB і BGR. За допомогою функції

num_components, яка повертає кількість компонентів у кожному форматі, можна визначити, скільки байтів потрібно для одного пікселя. Це дозволяє оптимізувати обробку кольорів у різних форматах.

Далі реалізується трейт PixelFormat, який визначає формат збереження даних пікселя і специфікує порядок кольорових каналів та тип даних кожного пікселя. Важливим елементом є макрос impl_pix_fmt, що автоматизує створення структур для кожного формату, таких як Rgbx8, Xrgb8, Bgrx8. Цей підхід спрощує код і дозволяє легко розширювати підтримку додаткових форматів пікселів шляхом виклику макроса для нових типів.

Наступний етап – налаштування режиму деінтерлейсингу, представлений переліком DeinterlaceMode. Вибір між режимами Bob, який інтерполює пропущені поля, та Skip, що залишає непроцесовані області без змін, дозволяє гнучко керувати обробкою відсутніх піксельних даних у зображеннях з інтерлейсингом. Це важливо для збереження якості зображення та мінімізації втрат.

Після цього визначається структура Rect, яка задає прямокутник для копіювання піксельних даних у буфері. Структура приймає координати верхньої, лівої, нижньої та правої меж зображення, що дозволяє точно визначати області для читання чи запису. Функції width і height допомагають обчислити ширину та висоту прямокутника, що спрощує обробку й маніпуляцію областями зображення.

Завершальний етап — налаштування структури BlitInfo, яка містить параметри для копіювання зображення в буфер YIQ або з нього. Поля структури, такі як rect, destination, row_bytes, та flip_y, дозволяють визначити, як і куди буде скопійована частина зображення. Функція from_full_frame автоматично налаштовує BlitInfo для копіювання всього зображення, що значно полегшує роботу, якщо обробляється цілий кадр.

3.4 Розробка плагіну

Після розробки алгоритму та модулів обробки відео необхідно розробити плагін, яким можна буде контролювати кількість та потужність ефектів.

Першим етапом ініціалізації даних у програмі є створення нового об’єкта типу SharedData, який використовує інформацію про хост для отримання різних службових інтерфейсів, необхідних для роботи плагіна з OpenFx API. У даному випадку це інтерфейси для роботи з властивостями, ефектами, пам'яттю та параметрами (див. приклад коду).

Для отримання цих інтерфейсів використовується метод fetchSuite, який викликається на інформації про хост. Кожен виклик fetchSuite намагається отримати потрібний інтерфейс та перетворює його на вказівник на відповідну структуру (наприклад, OfxPropertySuiteV1).

Далі код перевіряє, чи були інтерфейси отримані успішно за допомогою методу as_ref(), що перетворює вказівник на посилання, або повертає помилку, якщо інтерфейс недоступний. У випадку успішної ініціалізації, усі отримані інтерфейси зберігаються в структурі SharedData.

Перша функція action_load виконує операцію завантаження спільних даних плагіна. Вона використовує механізм блокування для забезпечення потокобезпеки доступу до загальних ресурсів. Конкретно, ця функція отримує блок запису для змінної shared_data, яка містить інформацію про плагін, після чого перевіряє, чи підтримується хостом плагіна обробка кількох глибин зображень. Це виконується через виклик функції propGetInt, яка зчитує відповідні властивості хосту та встановлює внутрішню змінну supports_multiple_clip_depths. Якщо підтримка наявна, вона зберігається в спільних даних, і функція успішно завершується.

Функція action_describe надає можливість описати властивості плагіна для хоста. Вона отримує дескриптор ефекту, який представляє плагін у середовищі хоста, і встановлює різноманітні властивості через функції встановлення значень властивостей propSetString та propSetInt. Це включає в себе призначення мітки

плагіна, визначення контекстів його використання (як, наприклад, контекст фільтру або загальний контекст), підтримку різних глибин пікселів та інших характеристик, таких як безпека рендерингу в багатопотоковому режимі. Зокрема, значення властивості kOfxImageEffectPluginRenderThreadSafety вказує на те, що рендеринг є повністю безпечним для роботи з потоками.

Функція map_params, яка відповідає за відображення налаштувань плагіна на параметри OpenFX. Вона приймає на вхід набір дескрипторів налаштувань (setting_descriptors) та ітеративно обробляє кожен із них. Залежно від типу налаштування, ця функція викликає відповідні процедури для його визначення. Наприклад, для перелікових налаштувань використовується тип параметра kOfxParamTypeChoice, де для кожної опції встановлюється її мітка та вибирається значення за замовчуванням.

Аналогічно, для налаштувань, що представляють відсоткові значення, функція визначає мінімальні та максимальні межі, використовуючи тип параметра kOfxParamTypeDouble. Важливим аспектом є обробка груп налаштувань. Кожна група має свій дескриптор, який містить дочірні налаштування, і створюється шляхом визначення параметра типу kOfxParamTypeGroup. Внутрішньо кожне налаштування в групі отримує відповідні властивості, включаючи мітки та значення за замовчуванням.

Одним із викликів при роботі з OpenFX API є управління строками C, що використовуються в зовнішньому інтерфейсі. Оскільки Rust має іншу модель управління пам’яттю, порівняно з C, в цьому коді використовуються спеціальні конструкції на основі CString для того, щоб гарантувати правильне виділення та звільнення пам'яті для строк при передачі їх зовнішнім бібліотекам. Це дозволяє уникнути витоків пам’яті та забезпечує коректне функціонування програми.

Функція map_params відповідає за створення та налаштування параметрів ефекту на основі переданих описів параметрів (setting_descriptors). Вона використовує API OFX для визначення нових параметрів через paramDefine, а

також встановлює властивості цих параметрів через propSetDouble, propSetInt і propSetString.

У цій функції обробляються різні типи параметрів: від перерахувань і логічних значень до діапазонів цілих та дійсних чисел. Для кожного з типів параметрів виконується відповідна послідовність операцій, включаючи визначення параметра, встановлення значень за замовчуванням, мінімальних і максимальних значень та інших властивостей. У випадку з груповими параметрами функція рекурсивно викликає саму себе для налаштування дочірніх параметрів, що дозволяє створювати вкладені структури налаштувань

Функція apply_params призначена для застосування параметрів ефекту під час його виконання. Вона отримує значення параметрів на заданий момент часу за допомогою paramGetValueAtTime і записує їх у структуру налаштувань ефекту (dst). Як і у функції map_params, вона обробляє різні типи параметрів: перерахування, логічні значення, цілі та дійсні числа. Ця функція також підтримує роботу з групами параметрів, для яких викликає саму себе рекурсивно. Основне завдання – отримати значення параметрів на вказаний момент часу та відобразити їх у структурі налаштувань ефекту, щоб вони могли бути використані під час рендерингу.

Функція action_describe_in_context використовується для опису контексту ефекту та його параметрів у OFX-плагіні. Вона визначає вхідні та вихідні кліпи для ефекту (наприклад, "Output" і "Source") та встановлює підтримувані формати зображення, такі як RGBA або RGB. Функція також визначає спеціальні параметри для завантаження і збереження налаштувань ("Load Preset" та "Save Preset") та додає параметри, що стосуються корекції гамма-сигналу.

Для кожного з параметрів використовується метод paramDefine, після чого їх властивості налаштовуються за допомогою функції map_params, яка налаштовує параметри на основі переданих описів.

Функція actions_get_regions_of_interest визначає область інтересу для ефекту. Вона отримує дескриптор кліпу і параметри з inArgs, а потім обчислює

область визначення (Region of Definition, RoD) для вихідного кліпу на основі часу. Спершу отримуються необхідні функції з OFX API, потім викликається функція clipGetHandle, яка отримує дескриптор кліпу під назвою "Source". Далі через clipGetRegionOfDefinition визначається RoD кліпу на основі переданого часу, після чого ці дані записуються у вихідні параметри outArgs.

Функція action_get_clip_preferences налаштовує параметри кліпів для ефекту, зокрема чи є кадри змінними (Frame Varying) та тип попереднього множення альфа-каналу (Pre-multiplication). Використовуючи API OFX, вона через виклики propSetInt і propSetString задає відповідні властивості, які вказують, що ефект може працювати з різними кадрами, а також що кліпи будуть оброблятися як непрозорі (Opaque).

Функція update_controls_disabled відповідає за керування тим, чи мають бути певні елементи керування увімкнені чи вимкнені. Вона отримує поточні параметри через API OFX та, залежно від того, чи параметри мають вкладені групи, рекурсивно викликає себе для оновлення дочірніх елементів. Параметри можуть належати до різних типів, таких як групи або логічні значення, і функція контролює, чи активні ці елементи на основі переданого значення enabled.

Функція set_controls_from_settings встановлює значення елементів керування на основі переданих налаштувань. Використовуючи API OFX, вона отримує дескриптори параметрів і значення, що мають бути встановлені, для різних типів параметрів, таких як перерахування, логічні значення, діапазони чисел та інші. Залежно від типу параметра (перерахування, числовий діапазон, логічний тощо), функція через paramSetValue встановлює відповідні значення параметрів у контексті OFX-плагіну.

Функція action_instance_changed відповідає за обробку змін параметрів ефекту. Вона використовує зовнішні бібліотеки для відкриття або збереження пресетів у форматі JSON.

Якщо користувач взаємодіє з параметром, який відповідає за завантаження або збереження пресетів, функція відкриває діалогове вікно для вибору файлу та

відповідно обробляє завантаження або збереження налаштувань. Якщо жодної дії з параметрами не відбувається, функція викликає оновлення стану контролів.

Реалізовані допоміжні функції для перетворення значень кольору між sRGB та лінійною гамою (srgb_gamma та srgb_gamma_inv), що дозволяє правильно обробляти кольори на різних етапах ефекту.

Структура OfxClipImage використовує механізм "RAII" для автоматичного звільнення зображення з кліпу при завершенні його використання. Це забезпечує безпечну обробку ресурсів в OFX.

Також присутня реалізація алокатора пам'яті OfxAllocator, що управляє виділенням та звільненням пам'яті за допомогою OFX API.

Дві основні структури — EffectApplicationParams та EffectStorageParam використовуються для зберігання параметрів застосування ефекту та проміжного збереження даних після обробки. Ці структури надають функції для застосування ефекту до зображення в YIQ-просторі та запису результату у вихідний буфер.

Цей фрагмент коду визначає та реалізує декілька перерахувань (enums) і механізмів для їх перетворення в межах плагіна для OpenFX (OFX). Ці перерахування представляють підтримувані типи глибин пікселів, компоненти зображення та формати пікселів, що є важливими для роботи з зображеннями і відеоматеріалами в рамках систем обробки графіки.

Одним із основних елементів є перерахування `SupportedPixelDepth`, яке відображає різні типи глибин пікселів, що можуть використовуватися під час обробки зображень. Це включає 8-бітові пікселі (`Byte`), 16-бітові пікселі (`Short`) та 32-бітові пікселі у форматі з плаваючою точкою (`Float`). Визначення таких параметрів є критично важливим для графічних операцій, оскільки різні глибини пікселів відповідають за точність відтворення кольорів і деталей зображення.

Для перетворення рядкових значень у підтримувані значення `SupportedPixelDepth`, реалізовано механізм `TryFrom`. Цей механізм перевіряє,

чи відповідає передане значення рядка одному з підтримуваних типів глибин пікселів. Якщо так, то повертається відповідне значення перерахування, інакше генерується помилка, яка сигналізує про непідтримуваний тип глибини пікселів. Це дозволяє безпечно та ефективно працювати з різними типами даних у графічних додатках.

Іншим важливим елементом є перерахування `SupportedImageComponents`, яке представляє типи компонентів зображення. Воно визначає, які канали присутні в зображенні: червоний, зелений та синій канали (RGB), або ж додатково альфа-канал (RGBA). Це дозволяє системі знати структуру зображення, що є важливим для правильного вибору методів його обробки. Застосований тут механізм перетворення також перевіряє вхідні дані на відповідність підтримуваним форматам і повертає помилку, якщо компонент зображення не підтримується.

Перерахування `SupportedPixelFormat` об’єднує глибину пікселів і компоненти зображення для створення конкретних форматів пікселів. Це дозволяє вибирати такі формати, як 8-бітові RGB або RGBA зображення, 16-бітові RGB або RGBA, або ж 32-бітові RGB або RGBA з плаваючою точкою. Комбінація цих характеристик надає плагіну можливість працювати з різноманітними типами зображень, що робить обробку більш гнучкою та ефективною.

Механізм перетворення `From` дозволяє автоматично визначати правильний формат пікселя на основі глибини пікселів і компонентів зображення. Це важливий аспект, оскільки він спрощує вибір формату пікселя на основі вхідних параметрів і дозволяє зменшити кількість помилок під час обробки зображень.

Таким чином механізм для ефективного визначення та вибору підтримуваних форматів зображення в плагіні OFX. Під час обробки зображень плагін має визначити, який тип пікселів і компоненти зображення обробляються (наприклад, 8-бітові RGB або 32-бітові RGBA зображення з плаваючою точкою).

Це дозволяє застосовувати правильні алгоритми для обробки кольорів, корекції або фільтрації зображень. Реалізація таких перерахувань і механізмів перетворення робить роботу з різними типами зображень стандартизованою, спрощуючи вибір формату пікселів і забезпечуючи безпечність даних в процесі їх обробки.

Основний фрагмент EffectStorageParams<'a> обробляє пікселі зображення, використовуючи колірну модель YIQ, і виконує рендеринг кадру з можливістю зворотного обертання (flip_y). Метод write_to_output забезпечує коректний запис піксельних даних в буфер в залежності від глибини кольору (Byte, Short, Float) і компонентів зображення (RGB, RGBA).

Рендеринг використовує кілька структур і параметрів, що містять інформацію про кліпи зображення, їхні компоненти та піксельну глибину, що дозволяє динамічно обробляти різні формати (8-бітні, 16-бітні та 32-бітні канали). Крім того, тут обробляються параметри для зворотного перетворення гамми (sRGB), що може застосовуватись під час рендерингу.

Наступним кроком буде визначення компонентів кольору для вхідного (джерела) та вихідного зображень. Функція propGetString зчитує властивість kOfxImageEffectPropComponents, яка містить інформацію про формат кольору, наприклад, RGB або RGBA. Це дозволяє алгоритму адаптуватися до зображень з різною кількістю кольорових каналів. Використання механізму try_from гарантує, що зчитане значення буде правильно інтерпретовано як підтримуваний формат.

Далі відбувається отримання глибини пікселя, використовуючи аналогічний підхід. Властивість kOfxImageEffectPropPixelDepth містить інформацію про формат даних пікселів, наприклад, 8-бітові, 16-бітові чи 32-бітові значення з плаваючою точкою. Це важливо для вибору відповідного алгоритму обробки, оскільки різні глибини потребують різного підходу до роботи з кольором та яскравістю.

Наступний крок — отримання просторових параметрів зображень. Для цього зчитуються такі характеристики, як кількість байтів у рядку (RowBytes), межі зображення (Bounds) і вказівник на дані зображення (Data Pointer). Ці параметри дозволяють працювати з конкретними фрагментами відеокадру, визначати його розмір у пам’яті та забезпечувати доступ до піксельних даних для подальших трансформацій.

Зчитані дані забезпечують базу для роботи з відеозображеннями різного формату. Вхідні параметри визначають, як саме будуть оброблятися пікселі, накладатися ефекти чи змінюватися яскравість. Завдяки стандартизованій обробці даних система забезпечує масштабованість і надійність, дозволяючи працювати з різноманітними форматами без значних змін у коді.

Така структура дозволяє плагіну гнучко адаптуватися до різних типів відео та забезпечувати високоточну обробку, яка відповідає характеристикам вихідного матеріалу. Наприклад, 8-бітові RGB зображення будуть оброблятися інакше, ніж 32-бітові RGBA, але загальна логіка залишатиметься сталою, що спрощує розробку та тестування.

Наступним кроком буде реалізація структури та ініціалізація набору параметрів через функцію getParamSet, яка дозволяє отримати доступ до налаштувань плагіна. Налаштування передаються до структури NtscEffectFullSettings за допомогою функції apply_params, що враховує часову змінність параметрів. Одним із важливих елементів є параметр SRGB_GAMMA_NAME, який визначає необхідність застосування корекції гамма sRGB. Дані зчитуються функцією paramGetValueAtTime, і, залежно від значення, встановлюється прапорець для подальшої обробки. Структура EffectApplicationParams спрощує передачу інформації про вхідні та вихідні дані, зокрема буфери зображення, межі кадрів і параметри ефекту, що дозволяє оптимізувати процеси обробки.

Подальша обробка виконується через динамічний вибір формату пікселів, що базується на глибині і компонентах зображення. Код підтримує різні

формати, такі як RGB8, RGBA32f, що дозволяє адаптувати алгоритми під конкретні формати даних. Залежно від вхідних і вихідних параметрів викликаються функції apply та write_to_output, які забезпечують коректне перетворення даних і збереження результатів. Головна функція main_entry обробляє запити від відеоредактора, такі як завантаження плагіна, опис його функціональності, зміну контексту або рендеринг кадрів. Інші допоміжні функції, зокрема OfxGetNumberOfPlugins і OfxGetPlugin, забезпечують інтеграцію плагіна у відеоредактор, надаючи інформацію про його ідентифікатор, версію та підтримувані API.

Таким чином, створено плагін для ефективного обробки відео, здатного працювати з різними форматами та налаштуваннями, забезпечуючи гнучкість і масштабованість для інтеграції в сучасні середовища редагування.

3.5 Висновки

У третьому розділі було обґрунтовано вибір мов програмування та технологій, які будуть використовуватися при розробці програмного продукту та наведено основні їх переваги.

У результаті аналізу було обрано мову програмування Rust та бібліотеку OpenFx для розробки плагіну.

Як середовище розробки обрано Visual Studio Code.

Було проведено розробку алгоритму та розроблено плагін на основі обраної бібліотеки OpenFX.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

4.1 Тестування системи

Одним із важливих етапів життєвого циклу будь-якого програмного продукту є тестування. Воно необхідне для того, щоб виявити дефекти у системі.

Обрано метод тестування білого ящика, оскільки він дозволяє досить ретельно перевірити та швидко змінити код для потрібної роботи.

Для перевірки візьмемо фрагмент відео у розширенні 1920 на 1080 пікселів та спробуємо надати йому ефекту VHS, базові настройки плагіну мають одразу показати різницю. Результат зображений на рисунку 4.2.

Рисунок 4.1 – Оригінальний відеоролик

Рисунок 4.2 – Відеоролик із увімкненим плагіном

Коректну роботу плагіну перевірено, відеоролик набув більш яскравих кольорів та отримав невеликі аналогові шуми по краях.

Далі необхідно протестувати один із елементів управління шумами. Змінюємо значення змінної «Snow» на значення 5. Результат тестування зображено на рисунку 4.3.

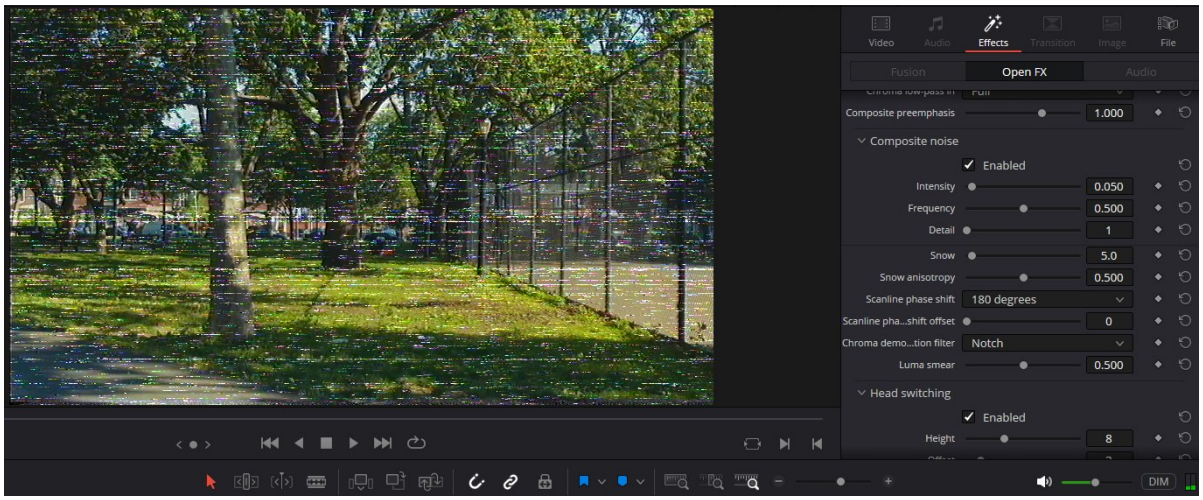

Рисунок 4.3 – Результат тестування

Результат тестування зміни значення змінної є успішним. Відеоряд змінився та аналогові шуми стали більш помітними, що і відповідає умовам роботи цієї функції.

Далі необхідно протестувати, як програма вестиме себе при збільшенні значення хроматичного фільтру. Змінимо значення із стандарту 1000 на 4000 Результат запиту зображено на рисунку 4.3.

Рисунок 4.4 – Результат тестування хроматичного фільтру

У результаті отримуємо збільшення хроматичних аберацій та збільшення контрасту, що і має відбуватись згідно функціоналу.

Наступним кроком перевіримо чи працює функція зміщення висоти касетних шумів, що відбувається через велику кількість перезаписів касетної плівки. Для цього змінимо початкове значення функції із 9 на 50.

Рисунок 4.5 – Відеофрагмент із значенням функції 9

Рисунок 4.6 – Відеофрагмент із значенням функції 50

У результаті роботи функції бачимо, що висота лінії шумів підійнялась, а значить функція працює вірно.

Наступною перевіримо функцію, що створює композитний шум. Для цього порівняємо фрагмент відео із увімкненим та вимкненою цією функцією.

Рисунок 4.7 – Відеофрагмент без композитного шуму

Рисунок 4.8 - Відеофрагмент із увімкненим композитним шумом

Далі варто перевірити функцію, що відповідає за насиченість композитним шумом. Для цього порівняємо два кадри із параметром насиченості 0.4 та 1.

Рисунок 4.9 – Відеофрагмент із параметром насиченості «0.4»

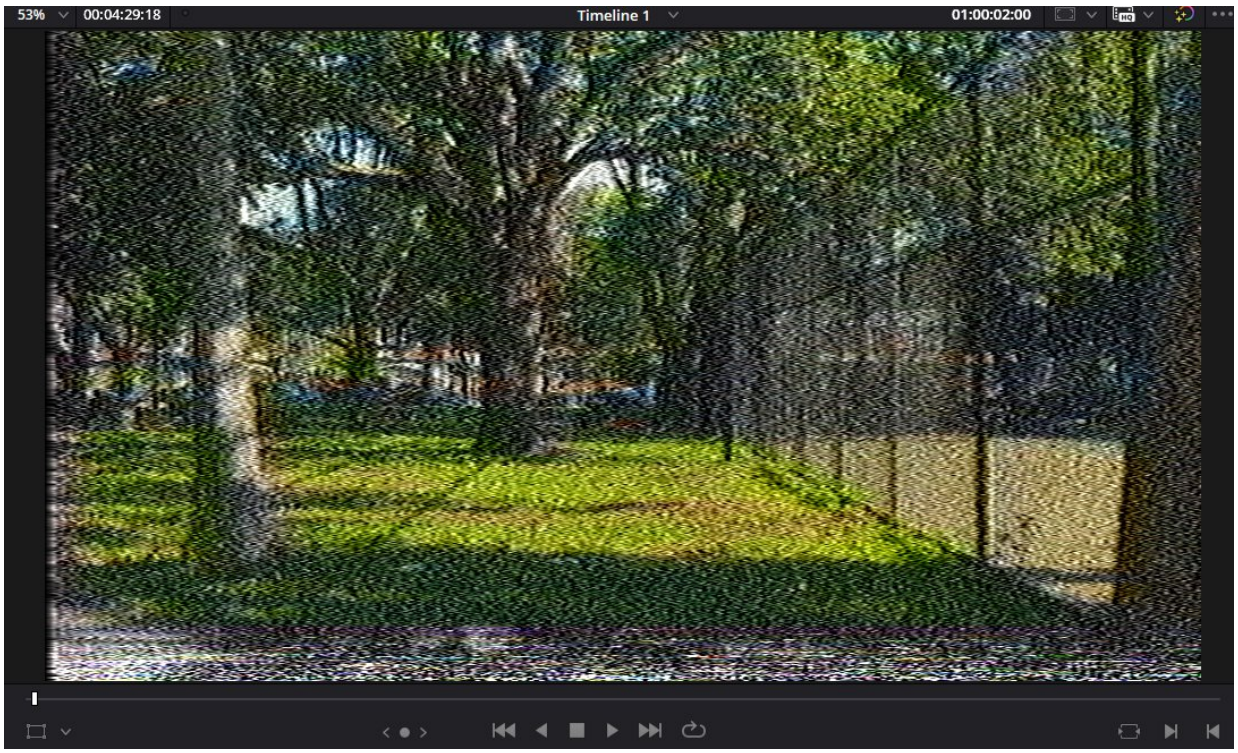

Рисунок 4.10 - Відеофрагмент із параметром насиченості «1»

У результаті роботи функції, відео що на рисунку 4.10 шумів багатократно більше та картинка стала ще більш шумною, що доводить вірну роботу цієї функції.

Наступним кроком перевіримо параметр частотності. Для цього порівняємо два кадри із параметром частотності 0.27 та 0.1.

Рисунок 4.11 – Відофрагмент із параметром частотності «0.27»

Рисунок 4.12 - Відофрагмент із параметром частотності «0.1»

У результаті роботи можемо бачити, що змінилась частота та форма композитного шуму та набула більш видної синусоїдної форми, що підтверджує роботу даної функції.

Наступним кроком перевіримо роботу функції низькочастотного фільтру. Для цього активуємо параметр Lowpass Filter у плагіні та встановлюємо його на функцію Константа (Blurry). Це дозволяє згладити високочастотні складові сигналу, що імітує ефект зниження деталізації зображення.

Рисунок 4.13 – Відеофрагмент до увімкнення низькочастотного фільтру

Рисунок 4.14 – Відеофрагмент із увімкненим низькочастотним фільтром

Відеофрагмент став дуже нечітким та «мильним», що і підтверджує правильність реалізації низькочастотного фільтру, який ефективно згладжує амплітуду високочастотних коливань.

Наступним елементом перевіримо втрату кольору, що відповідає зношенню аналогових носіїв. Спочатку хроматичні втрати із значенням «0», а потім «0.5»

Рисунок 4.15 – Відеофрагмент із увімкненим значенням хроматичної втрати «0»

Рисунок 4.16 - Відеофрагмент із увімкненим значенням хроматичної втрати «0.5»

У результаті можна спостерігати, що відеофрагмент втратив у контрасті та почав втрачати колір. Це підтверджує коректну роботу даної функції.

Наступною функцією протестовано помилку фази по контрасту, що теж доволі характерно для аналогового носія VHS. Було обрано значення «0» та «0.35».

Рисунок 4.17 - Відеофрагмент із помилкою фази контрасту «0»

Рисунок 4.18 - Відеофрагмент із помилкою фази контрасту «0.35»

Можна стверджувати, що колір на відеофрагменті змінився із зеленого на фіолетовий, оскільки відбувається збій у фазі контрасту. Таким чином функція працює вірно.

У результаті тестування було доведено, що розроблений функціонал працює відповідно до технічного завдання, та відповідає усім заявленим технічним характеристикам.

4.2 Розробка інструкції користувача

Першим етапом розробки інструкції користувача є визначення технічних вимог програмного продукту.

У таблицях 4.1 та 4.2 наведено мінімальну та рекомендовану конфігурацію персонального комп’ютера для запуску програми.

Таблиця 4.1 – Мінімальна конфігурація:

Тип процесора	64-розрядний (x86) або процесор з тактовою частотою 2 ГГц

Об’єм оперативної пам’яті	8 ГБ для 64-разрядної системи
Вільне місце на жорсткому диску	1 ГБ
Графічний прискорювач	Інтегрований графічний прискорювач, сумісний з DirectX9
Операційна система	Windows 10/Mac/Linux

Таблиця 4.2 – Рекомендована конфігурація:

Тип процесора	64-розрядний (x64) процесор з тактовою частотою 3 ГГц
Об’єм оперативної пам’яті	16 ГБ для 64-разрядної системи
Вільне місце на жорсткому диску	1 ГБ
Графічний прискорювач	Графічний прискорювач, сумісний з DirectX9 та об’ємом відеопам’яті 4 ГБ
Операційна система	Windows 10/Mac/Linux

Спочатку потрібно запустити відеоредактор, що підтримує роботу бібліотеки OpenFX. Рекомендується використовувати Davinchi Resolve 18 або вище.

Процес запуску зображено на рисунку 4.19

Рисунок 4.19 – Запуск DaVinchi Resolve

Наступний крок – додавання файлу бандлу для розпізнавання бібліотеки OpenFX. Для цього ми використаємо інструкцію на офіційному сайті бібліотеки та стандартну локацію для операційної системи Windows 11.

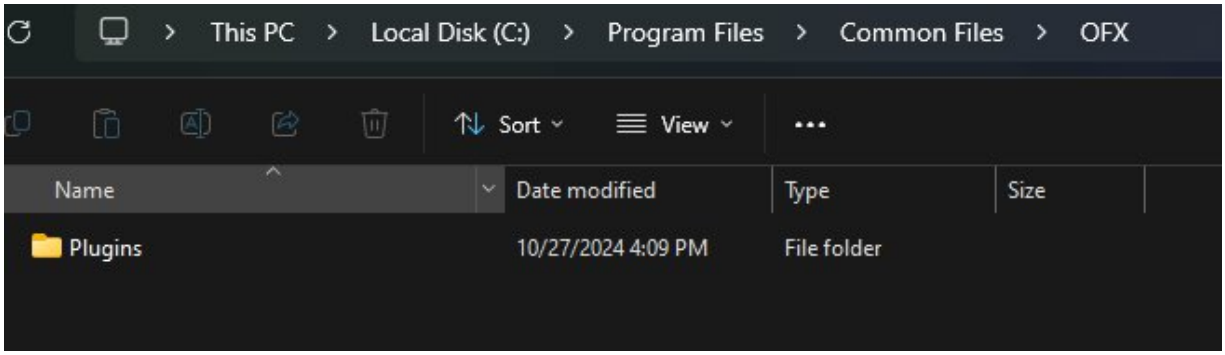

Рисунок 4.20 – Директорія «OFX»

Помістимо файл папку із бандлом у папку Plugins на локальному диску C та повернемось до відеоредактору.

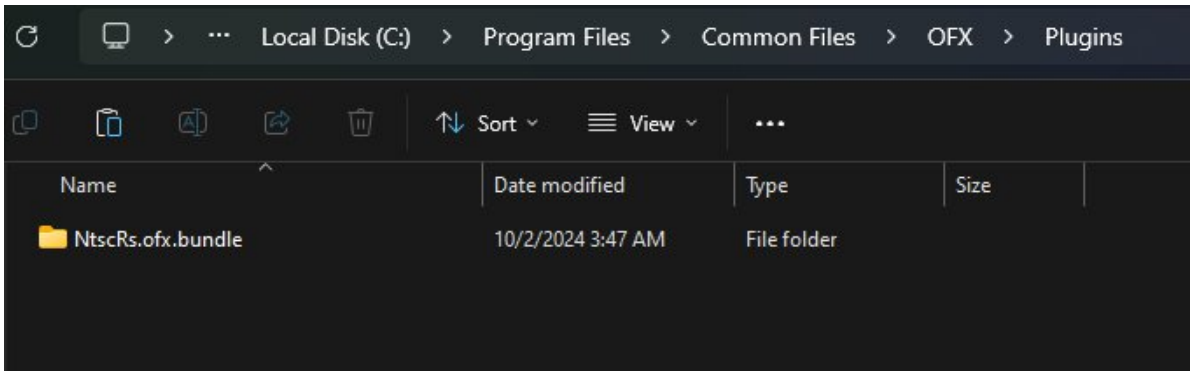

Рисунок 4.21 – Папка у директорії «OFX/Plugins»

Далі потрібно обрати у вікні вибору плагінів відеоредактора потрібно знайти плагін та додати його на відеоряд. Він повинен відобразитись у панелі ефектів відеоредактору у вкладці OpenFX. Якщо цього не відбувається потрібно пересканувати дерикторію плагінів у відеоредакторі.

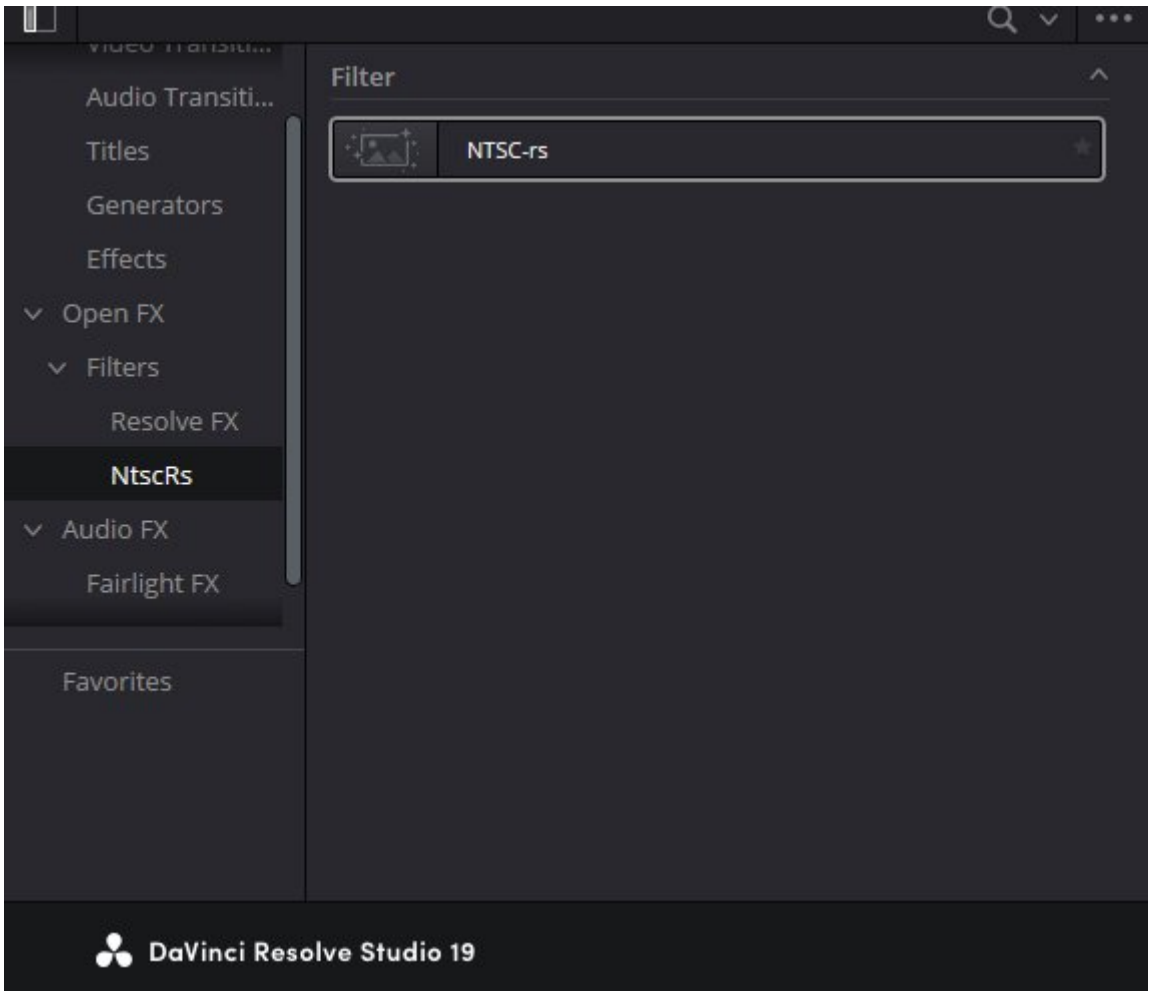

Рисунок 4.22 – Плагин у списку ефектів всередині відеоредактору

Наступним кроком потрібно під’єднати плагін до відеоряду. Для цього потрібно перетягнути на вкладку Edit відеоряд. Після успішного перетягування та імпортування відоряду на монтажний стіл перетягуємо плагін на відеоряд. Двічі натискаємо на відеоряд та у вкладці «Ефекти» бачимо результат – робочий плагін (див. рис. 4.24).

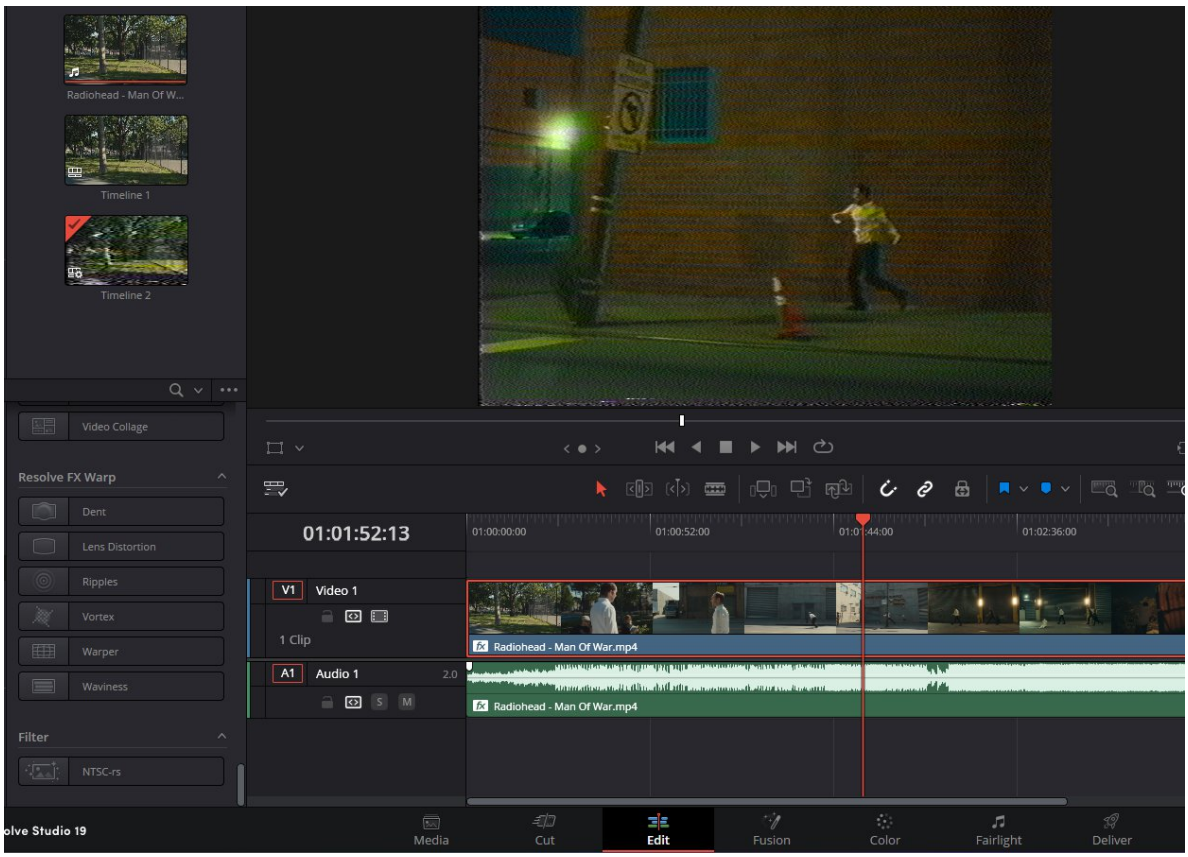

Рисунок 4.23 – Вкладка Edit

Ця вкладка є головною для редагування відео: тут робота із з таймлайном, можливість додавати кліпи, переходи, звукові доріжки та ефекти.

У вкладці Edit ви також можете застосовувати плагіни. Для цього спочатку імпортуємо відео на таймлайн, а потім перетягуємо потрібний плагін з розділу Ефекти на відеоряд. Подвійне натискання на кліп дозволяє отримати доступ до додаткових налаштувань, де є можливість переглянути результати застосування плагіна та внести зміни.

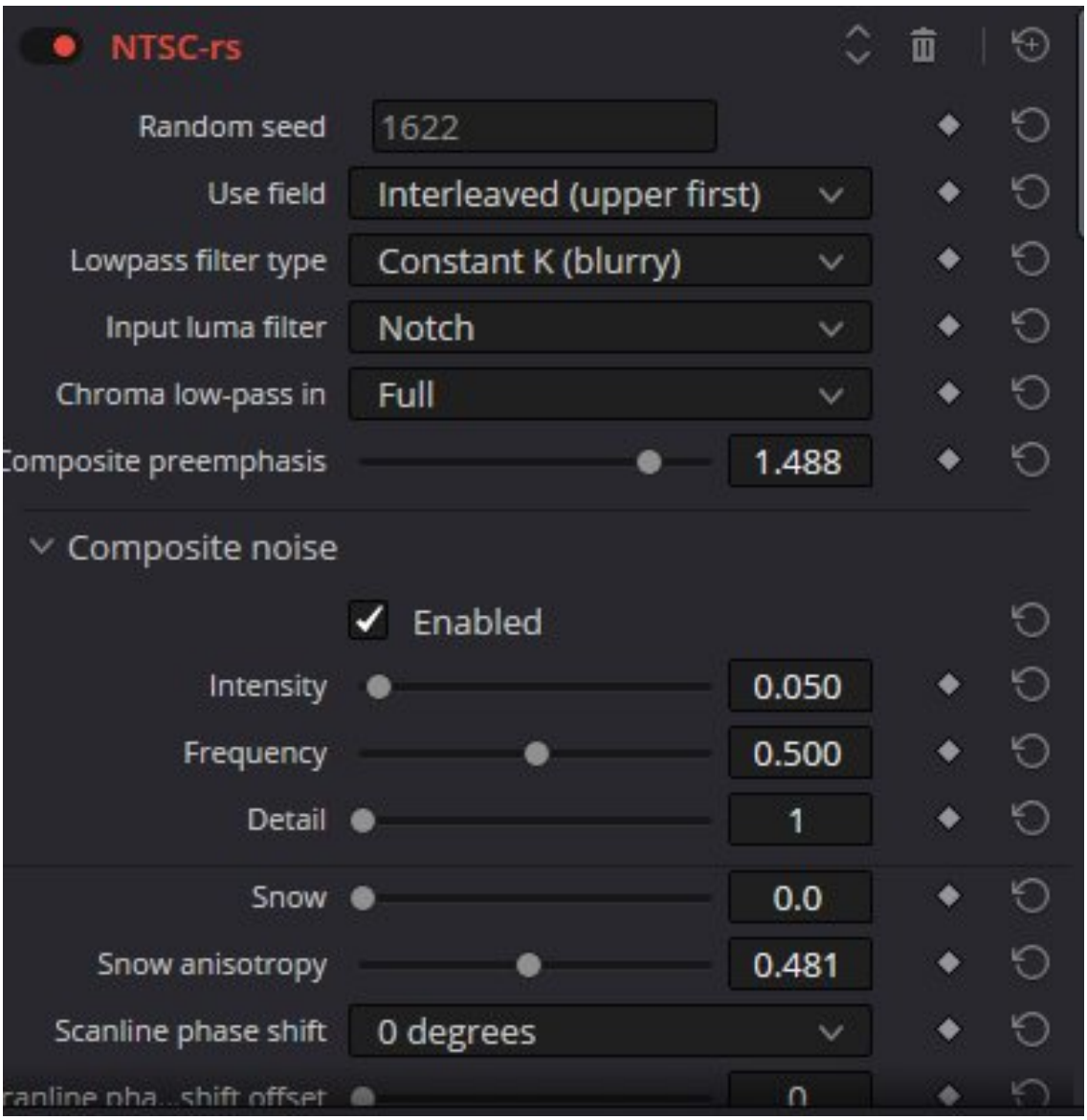

Рисунок 4.24 – Вкладка «Ефекти» на відеоряді

Наступним кроком буде внесення змін до змінних плагіну для отримання бажаного результату. Або можна скористатись генератором випадкових «сідів» який згенерує певні розташування шумів випадково.

Рисунок 4.25 – Значення сіду із значенням «1»

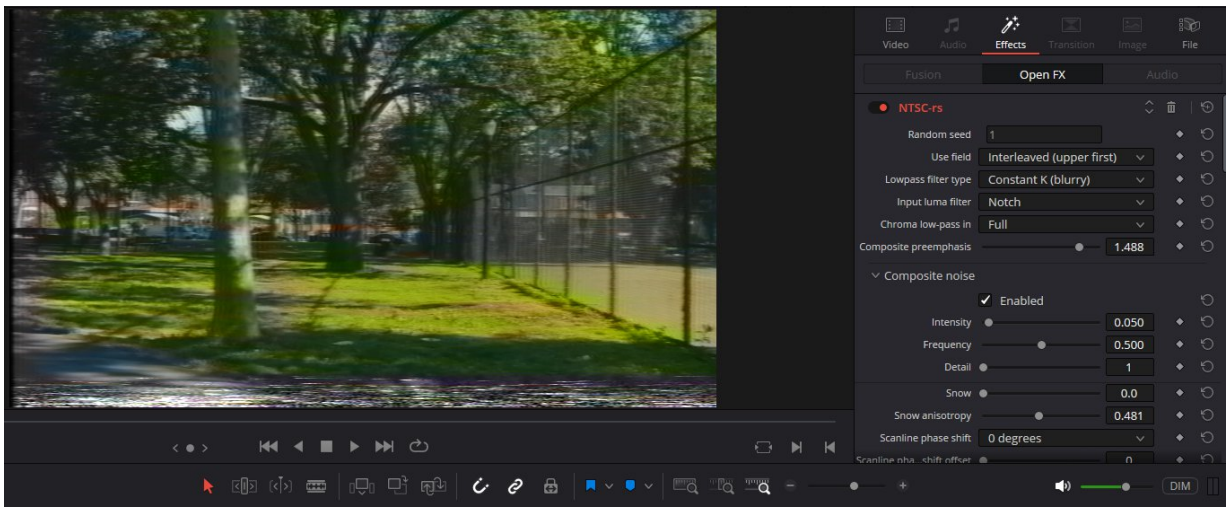

Рисунок 4.26 – Успішний запуск та робота програми

4.3 Висновки

У четвертому розділі було проведено тестування додатку, у результаті якого було доведено його повну працездатність та відповідність поставленому технічному завданню.

Розроблено інструкцію користувача по встановленню та використанню програмного продукту.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота за темою «Розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео » відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Розробка методу та програмного засобу для створення вінтажних

ефектів VHS-плівки у цифрових відео » є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1 [21].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка.

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	4	4
2. Ринкові переваги (наявність аналогів)	4	4	4
3. Ринкові переваги (ціна продукту)	3	3	3
4. Ринкові переваги (технічні властивості)	4	3	3
5. Ринкові переваги (експлуатаційні витрати)	3	3	3
6. Ринкові перспективи (розмір ринку)	3	4	3
7. Ринкові перспективи (конкуренція)	3	3	4
8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	3	3	3
10. Практична здійсненність (необхідність нових матеріалів)	3	3	3
11. Практична здійсненність (термін реалізації)	3	4	4
12. Практична здійсненність (розробка документів)	4	3	3
Сума балів	40	40	40
Середньоарифметична сума балів $СБ_c$	40		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 [21]

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ ,розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео » становить 40 балів, що, відповідно до

таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вищий середнього).

5.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео », під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.2.1 Витрати на оплату праці

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \times t_i}{T_p}, \quad (5.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дні;

T_p – середнє число робочих днів в місяці, T_p=21 день.

$$З_о = 40000{,}00 \cdot 22 / 21 = 41904{,}76 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	40000,00	1904,76	22,00	41904,76
Інженер-розробник програмного забезпечення	23200,00	1104,76	20,00	22095,24
Консультант-аналітик	14600,00	695,24	15,00	10428,57
Всього				74428,57

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \times t_i, \quad (5.2)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \times K_i \times K_c}{T_p \times t_{зм}}, \quad (5.3)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо *Мм*=8000,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [22];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 21 день;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 8000{,}00 \cdot 1{,}1 \cdot 1{,}65 / (21 \cdot 8) = 86{,}43$ грн.
$З_{р1} = 86{,}43 \cdot 8{,}00 = 691{,}43$ грн.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Встановлення	8	2	1,1	86,43	691,43

допоміжного офісного обладнання					
Монтаж робочого місця розробника графічних систем	12	2	1,1	86,43	1037,14
Інсталяція програмного забезпечення	5	5	1,7	133,57	667,86
Інші допоміжні роботи	10	3	1,1	86,43	864,29
Всього					3260,71

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \times \frac{Н_{дод}}{100\%}, \quad (5.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 12%.

$З_{дод}$ = (74428,57+ 3260,71) · 12 / 100% = 9322,71 грн.

5.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \times \frac{Н_{зп}}{100\%} \quad (5.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (74428,57+ 3260,71+ 9322,71) · 22 / 100% = 19142,64 грн.

5.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \times Ц_j \times K_j - \sum_{j=1}^{n} B_j \times Ц_{вj}, \quad (5.6)$$

де H_j – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

K_j – коефіцієнт транспортних витрат, (K_j = *1,1 … 1,15*);

B_j – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

$M_1 = 2 \cdot 200{,}00 \cdot 1{,}1 - 0{,}000 \cdot 0{,}00 = 440{,}0$ грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од, грн	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір Calipso Plus A4-500-80	200	2	0	0	440
Папір для записів Calipso Parers Light A5	170	1	0	0	187
Набір офісного працівника JOBMAX, с наповненням (16 предметів)	200	2	0	0	440
Flesh-пам'ять Kingston 32 GB	160	2	0	0	352
Всього					1419

5.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($K_в$)*, які використовують при проведенні НДР на тему «Розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео » відсутні.

5.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного

для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Витрати на спецустаткування, які використовують при проведенні НДР на тему «Розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео » відсутні.

5.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \times C_{прг.i} \times K_i \ , \qquad (5.7)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (K_i = *1,10…1,12);*

k – кількість найменувань програмних засобів.

$B_{прг} = 12400 \cdot 1 \cdot 1{,}12 = 13888$ *грн.*

Отримані результати зведемо до таблиці:

Таблиця 5.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
DaVinchi Resolve Studio	1	12400	13888
Всього			13888

5.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \times \frac{t_{вик}}{12}, \quad (5.8)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (59400{,}00 \cdot 1) / (5 \cdot 12) = 990{,}00$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
ПК (2 шт)	59400	5	1	990,00
Маршрутизатор	1870	3	1	51,94
DaVinchi Resolve Studio	6053,6	3	1	168,16
Приміщення лабораторії	212000	20	1	883,33
Всього				2093,43

5.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію ($B_е$) розраховуємо за формулою:

$$B_е = \sum_{i=1}^{n} \frac{W_{yi} \times t_i \times Ц_е \times K_{вni}}{h_i}, \quad (5.9)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e$ = 11,0 грн;

$K_{вп i}$ – коефіцієнт, що враховує використання потужності, $K_{вп i} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,3 \cdot 176,0 \cdot 11,0 \cdot 0,95 / 0,97 = 568,82$ грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Блок живлення ПК 1	0,3	176	568,82
Блок живлення ПК 2	0,3	160	517,11
Блок живлення маршрутизатора	0,02	60	12,93
Робоче місце дослідника	0,15	504	814,45
Всього			1913,32

5.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \times \frac{Н_{св}}{100\%}, \quad (5.10)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$В_{св}$ = (74428,57+ 3260,71) · 20 / 100% = 15537,86 грн.

5.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \times \frac{Н_{сп}}{100\%}, \quad (5.11)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 30%.

$В_{сп}$ = (74428,57+ 3260,71) · 30 / 100% = 23306,79 грн.

5.2.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \times \frac{Н_{ів}}{100\%}, \quad (5.12)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (74428,57+ 3260,71) · 50 / 100% = 38844,64 грн.

5.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів;

витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \times \frac{Н_{нзв}}{100\%}, \quad (5.13)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$B_{нзв} = (74428,57 + 3260,71) \cdot 100 / 100\% = 77\,689,29$ *грн.*

Витрати на проведення науково-дослідної роботи розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.14)$$

$B_{заг} = 280846,96$ *грн.*

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{h}, \quad (5.15)$$

де h - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо h=0,9.

ЗВ = 280846,96 / 0,9 = 312052,18 грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів

тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео » передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

1-й рік – 500 користувачів;

2-й рік – 450 користувачів;

3-й рік – 400 користувачів.

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 700 користувачів;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 2500 грн;

$\pm\Delta Ц_o$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 500 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [22]

$$\Delta П_i = (\pm\Delta Ц_o \times N + Ц_o \times \Delta N)_i \times t \times r \times (1 - \frac{J}{100}), \qquad (5.16)$$

деλ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2024 році ставка податку на додану вартість складає 20%, а коефіцієнт λ *=0,8333*;

r – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо r *=45%*;

φ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2024 році φ *=18%*;

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ =(700,00·500,00+3000,00·500)·0,83·0,45·(1-0,18/100%)= 568647,45 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2$ =(700,00·500,00+3000,00·(500+450))·0,83·0,45·(1-0,18/100%)=983606,4 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3$ =(700,00·500,00+3000,00·(500+450+400))·0,83·0,45·(1-0,18/100%)= 1352458,8 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+t)^i}, \qquad (5.17)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

t – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, t =0,3;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП$ =568647,45/(1+0,3)[1]+983606,4/(1+0,3)[2]+1352458,8/(1+0,3)[3]=1635030,18 грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \times ЗВ, \tag{5.18}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=5;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 312052,18 грн.

$PV = k_{інв} \times ЗВ$ = 2 · 312052,18 = 624104,3652 грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{5.19}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 1635030,18 грн;

PV – теперішня вартість початкових інвестицій, 624104,3652 грн.

$E_{абс} = ПП - PV$ = 1635030,18 - 624104,3652 = 1010925,82 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.20)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 1010925,82 грн;

PV – теперішня вартість початкових інвестицій, 624104,3652 грн;

$T_{ж}$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+1010925,82/624104,3652)^{1/3} = 0,38.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $t_{мін}$:

$$t_{мін} = d + f, \quad (5.21)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d =0,11;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,18.

$t_{мін}$ = 0,11+0,18 = 0,29 < 0,38 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео » доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \quad (5.22)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 0,38 = 2,64 року.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.4 Висновки до розділу

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео » становить 40 балів, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вищий середнього).

Також термін окупності становить 2,64 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео».

ВИСНОВКИ

У процесі виконання магістерської кваліфікаційної роботи було розроблено методи та засоби створення цифрових ефектів, що імітують візуальні властивості аналогових відеокасет формату VHS.

Для реалізації поставлених завдань використовувалися сучасні засоби розробки: мова програмування Rust, бібліотека OpenFX для створення плагінів та середовище розробки Visual Studio Code. Робота оформлена відповідно до методичних рекомендацій.

Було проведено аналіз сучасних підходів до створення цифрових відеоефектів, розглянуто існуючі аналоги, їхні особливості, недоліки та розроблено порівняння з власним програмним продуктом.

У ході роботи обрано мову Rust для забезпечення високої продуктивності та безпеки програмного коду. Також реалізовано підтримку стандарту OpenFX, що забезпечує інтеграцію плагіна з популярними відеоредакторами.

Розроблено алгоритми для перетворення колірних просторів RGB у YIQ, моделювання шумів, ефектів стрічкового спотворення та інших візуальних дефектів, характерних для VHS.

У результаті роботи було створено плагін, що дозволяє імітувати VHS-ефект на цифрових відеоматеріалах. Реалізовано ефекти хроматичної аберації, аналогових шумів, зміщення зображення за синусоїдальною траєкторією, скануючих (інтерлінгових) ліній, статичних перешкод, вертикальної розсинхронізації, зернистості, а також налаштування яскравості, контрасту та кольорового балансу, що разом забезпечують повноцінну симуляцію характерних візуальних артефактів відеокасет формату VHS.

Проведено тестування, яке підтвердило відповідність програмного продукту вимогам технічного завдання. Розроблено інструкцію користувача, що містить опис використання плагіна та його функціональних можливостей.

Створений продукт може бути використаний як у творчих проектах, так і для наукових досліджень, пов’язаних із відтворенням старих відеотехнологій у сучасних умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Матеріали КНУ імені Тараса Шевченка [Електронний ресурс]. URL: http://kinolab.knu.ua/inform1.shtml#:~:text=VHS%20(Video%20Home%20Standart)%2D,PAL%20%2D%2023.39%20%D0%BC%D0%BC%2F%D1%81%D0%B5%D0%BA.
2. YumCinema. Tarantino Shooting on Digital is Like Eating a Veggie Burger [Електронний ресурс]. URL: https://ymcinema.com/2022/10/14/tarantino-shooting-on-digital-is-like-eating-a-veggie-burger/#:~:text=Tarantino%20hates%20shooting%20with%20digital,a%20steak%20(film)%20vs.
3. NoFilmschool. What is IMAX? [Електронний ресурс]. URL: https://nofilmschool.com/what-is-imax
4. NoFilmschool. Why Dune was shot on digital, Transferred to 35mm, Then Scanned to Digital [Електронний ресурс]. Режим доступу до ресурсу: https://nofilmschool.com/Dune-Digital-Film-Process.
5. AFcinema. Kafen Tuxen explain technical choices [Електронний ресурс]. URL: https://www.afcinema.com/Kasper-Tuxen-DFF-explains-the-technical-choices-for-The-Apprentice-by-Ali-Abbasi.html
6. 3D LUT’s explained [Електронний ресурс]. URL: https://www.canon.ua/pro/stories/how-to-use-3d-luts/#:~:text=A%20LUT%20(Lookup%20Table)%20is,number%20in%20the%20other%20column.
7. Ben Goldsmith. Adobe Premiere Pro: A Complete Course and Compendium of Features (Course and Compendium, 4), 2021. 512 с.
8. 1967: Retro Filters & Effects. [Електронний ресурс]. URL: https://play.google.com/store/apps/details?id=com.retro.year1967.
9. Hipstamatic. [Електронний ресурс]. URL: https://hipstamatic.app/app.

10. Р.Р. Голубенко, Д.І. Катєльніков. Розробка Методу І Програмного Засобу для Створення Вінтажних Ефектів VHS-плівки У Цифрових Відео. Матеріали з конференції комп'ютерних ігор 26-27 вересня 2024 р. – Одеса. 2024. С. 248-249.
11. OpenFX. [Електронний ресурс]. URL: https://openeffects.org/.
12. Why Analog Designs Fail. [Електронний ресурс]. URL: https://semiengineering.com/making-analog-more-reliable/.
13. VHS and how it works. [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/VHS#:~:text=The%20tape%20is%20wrapped%20around,and%20the%20linear%20audio%20tracks..
14. Colin Bendell, Tim Kadlec, Yoav Weiss, Guy Podjarny, Nick Doyle, Mike McCall. High Performance Images: Shrink, Load and Deliver Images for Speed 1st Edition. – O'Reilly Media, 2017. 351 с.
15. Compression algorithms. H.264 Vs H.265 – Which Should You Use? [Електронний ресурс]. URL: https://accsoon.com/explore/h264-vs-h265-which-should-you-use/ /characterbert-reconciling-elmo-and-bert-for-word-level-open-vocabulary-representations-from-94037fe68b21.
16. Deep learning super sampling. [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/Deep_learning_super_sampling
17. Graphics processing unit. [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Graphics_processing_unit.
18. RTX. [Електронний ресурс]. URL: https://www.nvidia.com/en-us/design-visualization/technologies/rtx/.
19. What is FPS? [Електронний ресурс]. URL: https://www.lenovo.com/us/en/glossary/fps/?orgRef=https%253A%252F%252Fwww.google.com%252F&srsltid=AfmBOorX0NccmQ4w6II1uSoT8qdqUTl3pIpQBbEuVUjuTQDIxOiL7faR

20. Interlaced display. [Електронний ресурс]. URL: https://www.techtarget.com/whatis/definition/interlaced-display
21. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. 42 с
22. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с.

ДОДАТКИ

100

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"18" вересня 2024 р.

Технічне завдання
на магістерську кваліфікаційну роботу «Розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
_____ к.т.н., доц. Д.І. Катєльніков
"19" вересня 2024 р.
Виконав:
_____ студент гр. 1ПІ-23м Р.Р. Голубенко
"19" вересня 2024 р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео».

Галузь застосування – системи обробки відео.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол № 310 від « 17 » вересня 2024 р.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є розширення набору інструментів відеорежисера і підвищення естетичної цінності відеоконтенту за рахунок використання модулів генерації вінтажних ефектів VHS-плівки у цифрових відео.

Призначення роботи – підвищення ефективності та надійності автоматичної обробки текстової інформації.

4. Вихідні дані для проведення НДР

1. NoFilmschool. Why Dune was shot on digital, Transferred to 35mm, Then Scanned to Digital [Електронний ресурс]. Режим доступу до ресурсу: https://nofilmschool.com/Dune-Digital-Film-Process.

2. AFcinema. Kafen Tuxen explain technical choices [Електронний ресурс]. URL: https://www.afcinema.com/Kasper-Tuxen-DFF-explains-the-technical-choices-for-The-Apprentice-by-Ali-Abbasi.html

3. 3D LUT’s explained [Електронний ресурс]. URL: https://www.canon.ua/pro/stories/how-to-use-3d-luts/#:~:text=A%20LUT%20(Lookup%20Table)%20is,number%20in%20the%20other%20column.

4. Ben Goldsmith. Adobe Premiere Pro: A Complete Course and Compendium of Features (Course and Compendium, 4), 2021. 512 с.

5. Технічні вимоги

Вхідні дані – цифровий відеоматеріал – результат цифровий відеоматеріал із VHS-ефектом.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до магістерської кваліфікаційної роботи;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії і етапи розробки

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку систем створення VHS-ефектів та постановка задач дослідження	20.09.2024 – 30.09.2024
2	Розробка методів та засобів реалізації симуляції аналогового шуму та артефактів	01.10.2024 – 10.10.2024
3	Розробка засобу для створення вінтажних ефектів VHS-плівки	11.10.2024 –25.10.2024

4	Тестування програмного додатку	26.10.2024 – 15.11.2024
5	Економічна частина	16.11.2024 – 30.11.2024

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

104

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео**

Тип роботи: кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 1ПІ – 23м

Науковий керівник: к.т.н. доц. Катєльніков Д.І.

Turnitin	
Оригінальність	**94 %**
Схожість	6 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений з повним звітом подібності, який був згенерований Системою щодо роботи «Розробка методу та програмного засобу для створення вінтажних ефектів VHS-плівки у цифрових відео».

Автор _____ Голубенко Роман Русланович

Опис прийнятого рішення: **допустити до захисту**

Особа, відповідальна за перевірку _____ Черноволик Г. О.
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)

Додаток В – Лістинг програми

Filter_profile.rs

```
extern crate criterion;
use criterion::{black_box, criterion_group, criterion_main, Criterion};
use image::ImageReader;
use ntscrs::{ntsc::NtscEffect, yiq_fielding::Rgb8};
#[cfg(not(target_os = "windows"))]
use pprof::criterion::{Output, PProfProfiler};

fn criterion_benchmark(c: &mut Criterion) {
    let img = ImageReader::open("./benches/balloons.png")
        .unwrap()
        .decode()
        .unwrap();
    let img = img.to_rgb8();
    c.bench_function("full effect", |b| {
        b.iter(|| {
            let img = img.clone();
            let width = img.width() as usize;
            let height = img.height() as usize;
            let mut buf = img.into_raw();
            NtscEffect::default().apply_effect_to_buffer::<Rgb8>((width, height), &mut
buf, 0);
            black_box(&mut buf);
        })
    });
}

criterion_group! {
    name = benches;
    config = {
        #[cfg(not(target_os="windows"))]
        let config = Criterion::default().with_profiler(PProfProfiler::new(100,
Output::Flamegraph(None)));
        #[cfg(target_os="windows")]
        let config = Criterion::default();

        config
    };
    targets = criterion_benchmark
}
criterion_main!(benches);
```

f32x4.rs

```
use std::{
```

```
    fmt::Debug,
    ops::{Add, AddAssign, Div, DivAssign, Mul, MulAssign, Sub, SubAssign},
    sync::OnceLock,
};

#[allow(dead_code)]
pub trait F32x4:
    Sized
    + Add<Output = Self>
    + Sub<Output = Self>
    + Mul<Output = Self>
    + Div<Output = Self>
    + AddAssign
    + SubAssign
    + MulAssign
    + DivAssign
    + Copy
    + Debug
{
    unsafe fn load(src: &[f32]) -> Self;

    unsafe fn load4(src: &[f32; 4]) -> Self {
        Self::load(src.as_slice())
    }

    unsafe fn load1(src: &f32) -> Self;

    fn store(self, dst: &mut [f32]);
    fn store1(self, dst: &mut f32);

    unsafe fn zero() -> Self {
        Self::set1(0.0)
    }

    unsafe fn set1(src: f32) -> Self;

    fn mul_add(self, a: Self, b: Self) -> Self;
    fn mul_sub(self, a: Self, b: Self) -> Self;
    fn neg_mul_add(self, a: Self, b: Self) -> Self;
    fn neg_mul_sub(self, a: Self, b: Self) -> Self;
    fn swizzle(self, x: i32, y: i32, z: i32, w: i32) -> Self;
    fn insert<const INDEX: i32>(self, value: f32) -> Self;
}

#[cfg(target_arch = "x86_64")]
pub mod x86_64 {
    use std::arch::x86_64::{
        __m128, _mm_add_ps, _mm_broadcast_ss, _mm_castps_si128, _mm_castsi128_ps,
_mm_div_ps,
```

```
        _mm_fmadd_ps, _mm_fmsub_ps, _mm_fnmadd_ps, _mm_fnmsub_ps, _mm_insert_epi32,
_mm_loadu_ps,
        _mm_mul_ps, _mm_permutevar_ps, _mm_set1_ps, _mm_set_epi32, _mm_set_epi8,
_mm_shuffle_epi8,
        _mm_store_ss, _mm_storeu_ps, _mm_sub_ps,
    };
    use std::{
        fmt::Debug,
        ops::{Add, AddAssign, Div, DivAssign, Mul, MulAssign, Sub, SubAssign},
    };

    use super::F32x4;

    #[derive(Clone, Copy, Debug)]
    #[repr(transparent)]
    pub struct IntelF32x4<const USE_AVX2: bool>(__m128);

    pub type AvxF32x4 = IntelF32x4<true>;
    pub type SseF32x4 = IntelF32x4<false>;

    const fn _mm_shuffle(x: i32, y: i32, z: i32, w: i32) -> i32 {
        (x << 0) | (y << 2) | (z << 4) | (w << 6)
    }

    impl<const USE_AVX2: bool> From<__m128> for IntelF32x4<USE_AVX2> {
        #[inline(always)]
        fn from(src: __m128) -> Self {
            IntelF32x4(src)
        }
    }

    impl<const USE_AVX2: bool> From<IntelF32x4<USE_AVX2>> for __m128 {
        #[inline(always)]
        fn from(src: IntelF32x4<USE_AVX2>) -> Self {
            src.0
        }
    }

    impl<const USE_AVX2: bool> Add for IntelF32x4<USE_AVX2> {
        type Output = Self;

        #[inline(always)]
        fn add(self, rhs: Self) -> Self::Output {
            unsafe { _mm_add_ps(self.into(), rhs.into()).into() }
        }
    }

    impl<const USE_AVX2: bool> Sub for IntelF32x4<USE_AVX2> {
        type Output = Self;
```

```
    #[inline(always)]
    fn sub(self, rhs: Self) -> Self::Output {
        unsafe { _mm_sub_ps(self.into(), rhs.into()).into() }
    }
}

impl<const USE_AVX2: bool> Mul for IntelF32x4<USE_AVX2> {
    type Output = Self;

    #[inline(always)]
    fn mul(self, rhs: Self) -> Self::Output {
        unsafe { _mm_mul_ps(self.into(), rhs.into()).into() }
    }
}

impl<const USE_AVX2: bool> Div for IntelF32x4<USE_AVX2> {
    type Output = Self;

    #[inline(always)]
    fn div(self, rhs: Self) -> Self::Output {
        unsafe { _mm_div_ps(self.into(), rhs.into()).into() }
    }
}

impl<const USE_AVX2: bool> AddAssign for IntelF32x4<USE_AVX2> {
    #[inline(always)]
    fn add_assign(&mut self, rhs: Self) {
        *self = *self + rhs;
    }
}

impl<const USE_AVX2: bool> SubAssign for IntelF32x4<USE_AVX2> {
    #[inline(always)]
    fn sub_assign(&mut self, rhs: Self) {
        *self = *self - rhs;
    }
}

impl<const USE_AVX2: bool> MulAssign for IntelF32x4<USE_AVX2> {
    #[inline(always)]
    fn mul_assign(&mut self, rhs: Self) {
        *self = *self * rhs;
    }
}

impl<const USE_AVX2: bool> DivAssign for IntelF32x4<USE_AVX2> {
    #[inline(always)]
    fn div_assign(&mut self, rhs: Self) {
```

```
        *self = *self / rhs;
    }
}

impl<const USE_AVX2: bool> F32x4 for IntelF32x4<USE_AVX2> {
    #[inline(always)]
    unsafe fn load(src: &[f32]) -> Self {
        unsafe { _mm_loadu_ps(src[0..4].as_ptr()).into() }
    }

    #[inline(always)]
    unsafe fn load1(src: &f32) -> Self {
        if USE_AVX2 {
            unsafe { _mm_broadcast_ss(src).into() }
        } else {
            unsafe { _mm_set1_ps(*src).into() }
        }
    }

    #[inline(always)]
    fn store(self, dst: &mut [f32]) {
        unsafe { _mm_storeu_ps(dst[0..4].as_mut_ptr(), self.into()) }
    }

    #[inline(always)]
    fn store1(self, dst: &mut f32) {
        unsafe { _mm_store_ss(dst, self.into()) }
    }

    #[inline(always)]
    unsafe fn set1(src: f32) -> Self {
        unsafe { _mm_set1_ps(src).into() }
    }

    #[inline(always)]
    fn mul_add(self, a: Self, b: Self) -> Self {
        if USE_AVX2 {
            unsafe { _mm_fmadd_ps(self.into(), a.into(), b.into()).into() }
        } else {
            (self * a) + b
        }
    }

    #[inline(always)]
    fn mul_sub(self, a: Self, b: Self) -> Self {
        if USE_AVX2 {
            unsafe { _mm_fmsub_ps(self.into(), a.into(), b.into()).into() }
        } else {
            (self * a) - b
```

```
            }
        }

        #[inline(always)]
        fn neg_mul_add(self, a: Self, b: Self) -> Self {
            if USE_AVX2 {
                unsafe { _mm_fnmadd_ps(self.into(), a.into(), b.into()).into() }
            } else {
                b - (self * a)
            }
        }

        #[inline(always)]
        fn neg_mul_sub(self, a: Self, b: Self) -> Self {
            if USE_AVX2 {
                unsafe { _mm_fnmsub_ps(self.into(), a.into(), b.into()).into() }
            } else {
                (unsafe { Self::set1(0.0) } - (self * a)) - b
            }
        }

        #[inline(always)]
        fn swizzle(self, x: i32, y: i32, z: i32, w: i32) -> Self {
            assert!((x | y | z | w) & !3 == 0, "Invalid swizzle indices");
            if USE_AVX2 {
                unsafe { _mm_permutevar_ps(self.into(), _mm_set_epi32(w, z, y,
x)).into() }
            } else {
                let x = (x << 2) as i8;
                let y = (y << 2) as i8;
                let z = (z << 2) as i8;
                let w = (w << 2) as i8;

                unsafe {
                    let control_mask = _mm_set_epi8(
                        w + 3,
                        w + 2,
                        w + 1,
                        w + 0,
                        z + 3,
                        z + 2,
                        z + 1,
                        z + 0,
                        y + 3,
                        y + 2,
                        y + 1,
                        y + 0,
                        x + 3,
                        x + 2,
```

```
                        x + 1,
                        x + 0,
                    );
                    _mm_castsi128_ps(_mm_shuffle_epi8(
                        _mm_castps_si128(self.into()),
                        control_mask,
                    ))
                    .into()
                }
            }
        }

        #[inline(always)]
        fn insert<const INDEX: i32>(self, value: f32) -> Self {
            assert!(INDEX & !3 == 0, "Invalid insert index");
            unsafe {
                _mm_castsi128_ps(_mm_insert_epi32::<INDEX>(
                    _mm_castps_si128(self.into()),
                    value.to_bits() as i32,
                ))
                .into()
            }
        }
    }
}

#[cfg(target_arch = "aarch64")]
pub mod aarch64 {
    use std::arch::aarch64::{
        float32x4_t, uint8x16_t, vaddq_f32, vcombine_u8, vcreate_u8, vdivq_f32,
vdupq_n_f32,
        vfmaq_f32, vfmsq_f32, vld1q_dup_f32, vld1q_f32, vmulq_f32, vnegq_f32,
vqtbl1q_u8,
        vreinterpretq_f32_u8, vreinterpretq_u8_f32, vsetq_lane_f32, vst1q_f32,
vst1q_lane_f32,
        vsubq_f32,
    };
    use std::{
        fmt::Debug,
        ops::{Add, AddAssign, Div, DivAssign, Mul, MulAssign, Sub, SubAssign},
    };

    use super::F32x4;

    #[derive(Clone, Copy, Debug)]
    #[repr(transparent)]
    pub struct ArmF32x4(float32x4_t);

    impl From<float32x4_t> for ArmF32x4 {
```

```
    #[inline(always)]
    fn from(src: float32x4_t) -> Self {
        ArmF32x4(src)
    }
}

impl From<ArmF32x4> for float32x4_t {
    #[inline(always)]
    fn from(src: ArmF32x4) -> Self {
        src.0
    }
}

impl Add for ArmF32x4 {
    type Output = Self;

    #[inline(always)]
    fn add(self, rhs: Self) -> Self::Output {
        unsafe { vaddq_f32(self.into(), rhs.into()).into() }
    }
}

impl Sub for ArmF32x4 {
    type Output = Self;

    #[inline(always)]
    fn sub(self, rhs: Self) -> Self::Output {
        unsafe { vsubq_f32(self.into(), rhs.into()).into() }
    }
}

impl Mul for ArmF32x4 {
    type Output = Self;

    #[inline(always)]
    fn mul(self, rhs: Self) -> Self::Output {
        unsafe { vmulq_f32(self.into(), rhs.into()).into() }
    }
}

impl Div for ArmF32x4 {
    type Output = Self;

    #[inline(always)]
    fn div(self, rhs: Self) -> Self::Output {
        unsafe { vdivq_f32(self.into(), rhs.into()).into() }
    }
}
```

```rust
impl AddAssign for ArmF32x4 {
    #[inline(always)]
    fn add_assign(&mut self, rhs: Self) {
        *self = *self + rhs;
    }
}

impl SubAssign for ArmF32x4 {
    #[inline(always)]
    fn sub_assign(&mut self, rhs: Self) {
        *self = *self - rhs;
    }
}

impl MulAssign for ArmF32x4 {
    #[inline(always)]
    fn mul_assign(&mut self, rhs: Self) {
        *self = *self * rhs;
    }
}

impl DivAssign for ArmF32x4 {
    #[inline(always)]
    fn div_assign(&mut self, rhs: Self) {
        *self = *self / rhs;
    }
}

impl F32x4 for ArmF32x4 {
    #[inline(always)]
    unsafe fn load(src: &[f32]) -> Self {
        unsafe { vld1q_f32(src[0..4].as_ptr()).into() }
    }

    #[inline(always)]
    unsafe fn load1(src: &f32) -> Self {
        unsafe { vld1q_dup_f32(src as *const _).into() }
    }

    #[inline(always)]
    fn store(self, dst: &mut [f32]) {
        unsafe { vst1q_f32(dst[0..4].as_mut_ptr(), self.into()) }
    }

    #[inline(always)]
    fn store1(self, dst: &mut f32) {
        unsafe { vst1q_lane_f32::<0>(dst as *mut _, self.into()) };
    }
```

```
#[inline(always)]
unsafe fn set1(src: f32) -> Self {
    unsafe { vdupq_n_f32(src).into() }
}

#[inline(always)]
fn mul_add(self, a: Self, b: Self) -> Self {
    unsafe { vfmaq_f32(b.into(), self.into(), a.into()).into() }
}

#[inline(always)]
fn mul_sub(self, a: Self, b: Self) -> Self {
    unsafe { vnegq_f32(vfmsq_f32(b.into(), self.into(), a.into())).into() }
}

#[inline(always)]
fn neg_mul_add(self, a: Self, b: Self) -> Self {
    unsafe { vfmsq_f32(b.into(), self.into(), a.into()).into() }
}

#[inline(always)]
fn neg_mul_sub(self, a: Self, b: Self) -> Self {
    unsafe { vnegq_f32(vfmaq_f32(b.into(), self.into(), a.into())).into() }
}

#[inline(always)]
fn swizzle(self, x: i32, y: i32, z: i32, w: i32) -> Self {
    assert!((x | y | z | w) & !3 == 0, "Invalid swizzle indices");
    let x = (x << 2) as u64;
    let y = (y << 2) as u64;
    let z = (z << 2) as u64;
    let w = (w << 2) as u64;
    unsafe {
        let indexes: uint8x16_t = vcombine_u8(
            vcreate_u8(
                (x + 0)
                    | (x + 1) << 8
                    | (x + 2) << 16
                    | (x + 3) << 24
                    | (y + 0) << 32
                    | (y + 1) << 40
                    | (y + 2) << 48
                    | (y + 3) << 56,
            ),
            vcreate_u8(
                (z + 0)
                    | (z + 1) << 8
                    | (z + 2) << 16
                    | (z + 3) << 24
```

```
                                | (w + 0) << 32
                                | (w + 0) << 40
                                | (w + 0) << 48
                                | (w + 0) << 56,
                            ),
                        );
                        vreinterpretq_f32_u8(vqtbl1q_u8(vreinterpretq_u8_f32(self.into()),
indexes)).into()
                    }
                }

                #[inline(always)]
                fn insert<const INDEX: i32>(self, value: f32) -> Self {
                    assert!(INDEX & !3 == 0, "Invalid insert index");
                    unsafe { vsetq_lane_f32::<INDEX>(value, self.into()).into() }
                }
            }
        }

        #[derive(Clone, Copy, Debug, PartialEq, Eq)]
        pub enum SupportedSimdType {
            #[cfg(target_arch = "x86_64")]
            Avx2,
            #[cfg(target_arch = "x86_64")]
            Sse41,
            #[cfg(target_arch = "aarch64")]
            Neon,
            None,
        }

        static SUPPORTED_SIMD_TYPE: OnceLock<SupportedSimdType> = OnceLock::new();

        pub fn get_supported_simd_type() -> SupportedSimdType {
            *SUPPORTED_SIMD_TYPE.get_or_init(|| {
                #[cfg(target_arch = "x86_64")]
                {
                    if is_x86_feature_detected!("avx2") && is_x86_feature_detected!("fma") {
                        SupportedSimdType::Avx2
                    } else if is_x86_feature_detected!("sse4.1") {
                        SupportedSimdType::Sse41
                    } else {
                        SupportedSimdType::None
                    }
                }
                #[cfg(target_arch = "aarch64")]
                {
                    SupportedSimdType::Neon
                }
                #[cfg(not(any(target_arch = "x86_64", target_arch = "aarch64")))]
```

```
        {
            SupportedSimdType::None
        }
    })
}

use std::mem::MaybeUninit;

use crate::f32x4::{get_supported_simd_type, F32x4, SupportedSimdType};

pub fn polynomial_multiply(a: &[f32], b: &[f32]) -> Vec<f32> {
    let degree = a.len() + b.len() - 1;

    let mut out = vec![0f32; degree];

    for ai in 0..a.len() {
        for bi in 0..b.len() {
            out[ai + bi] += a[ai] * b[bi];
        }
    }

    out
}

fn trim_zeros(input: &[f32]) -> &[f32] {
    let mut end = input.len() - 1;
    while input[end].abs() == 0.0 {
        end -= 1;
    }
    &input[0..=end]
}

fn each_mut<T, const N: usize>(arr: &mut [T; N]) -> [&mut T; N] {
    let mut out = unsafe { MaybeUninit::<[MaybeUninit<&mut T>;
N]>::uninit().assume_init() };
    for (src, dst) in arr.iter_mut().zip(&mut out) {
        dst.write(src);
    }

    unsafe { (&mut out as *mut _ as *mut [&mut T; N]).read() }
}

#[derive(Debug, Clone)]
#[non_exhaustive]
pub struct TransferFunction {
    pub num: Vec<f32>,
    /// Coefficients for the denominator polynomial.
    pub den: Vec<f32>,
}
```

```rust
impl TransferFunction {
    pub fn new<I, J>(num: I, den: J) -> Self
    where
        I: IntoIterator<Item = f32>,
        J: IntoIterator<Item = f32>,
    {
        let mut num = num.into_iter().collect::<Vec<f32>>();
        let mut den = den.into_iter().collect::<Vec<f32>>();
        if num.len() > den.len() {
            panic!("Numerator length exceeds denominator length.");
        }

        if den.len() == 2 || den.len() == 3 {
            den.resize(4, 0.0);
        }
        num.resize(den.len(), 0.0);

        TransferFunction { num, den }
    }

    pub fn cascade_self(&self, n: usize) -> Self {
        let mut filt = self.clone();
        for _ in 1..n {
            filt = &filt * self;
        }
        filt
    }

    /// Whether processing the rows in chunks or one-by-one is faster.
    pub fn should_use_row_chunks(&self) -> bool {
        // Row chunks are ~25% slower than processing each row one-by-one with the
scalar implementation.
        // We should only process rows in chunks with the SIMD implementation.
        self.den.len() == 4 && get_supported_simd_type() != SupportedSimdType::None
    }

    /// Return initial conditions for the filter that results in a given steady-state
value (e.g. "start" the filter as
    /// if every previous sample was the given value).
    fn initial_condition(&self, value: f32) -> Vec<f32> {
        // Adapted from scipy
        //
https://github.com/scipy/scipy/blob/da82ac849a4ccade2d954a0998067e6aa706dd70/scipy/sig
nal/_signaltools.py#L3609-L3742

        let filter_len = usize::max(self.num.len(), self.den.len());
        // The last element here will always be 0--in the loop below, we intentionally
do not initialize the last
```

```
        // element of zi.
        let mut zi = vec![0f32; filter_len];
        if value.abs() == 0.0 {
            return zi;
        }

        let first_nonzero_coeff = self
            .den
            .iter()
            .find_map(|coeff| {
                if coeff.abs() != 0.0 {
                    Some(*coeff)
                } else {
                    None
                }
            })
            .expect("There must be at least one nonzero coefficient in the
denominator.");

        let norm_num = self
            .num
            .iter()
            .map(|item| *item / first_nonzero_coeff)
            .collect::<Vec<f32>>();
        let norm_den = self
            .den
            .iter()
            .map(|item| *item / first_nonzero_coeff)
            .collect::<Vec<f32>>();

        let mut b_sum = 0.0;
        for i in 1..filter_len {
            let num_i = norm_num.get(i).unwrap_or(&0.0);
            let den_i = norm_den.get(i).unwrap_or(&0.0);
            b_sum += num_i - den_i * norm_num[0];
        }

        zi[0] = b_sum / norm_den.iter().sum::<f32>();
        let mut a_sum = 1.0;
        let mut c_sum = 0.0;
        for i in 1..filter_len - 1 {
            let num_i = norm_num.get(i).unwrap_or(&0.0);
            let den_i = norm_den.get(i).unwrap_or(&0.0);
            a_sum += den_i;
            c_sum += num_i - den_i * norm_num[0];
            zi[i] = (a_sum * zi[0] - c_sum) * value;
        }
        zi[0] *= value;
```

```
        zi
    }

    #[inline(always)]
    fn filter_sample(
        filter_len: usize,
        num: &[f32],
        den: &[f32],
        z: &mut [f32],
        sample: f32,
        scale: f32,
    ) -> f32 {
        // This function gets auto-vectorized by the compiler. The following attempts
to optimize it have backfired:
        // - Special-casing a function for when there's only one nonzero coefficient
in the numerator: slower.
        // - Using a smallvec to store all the coefficients: slower.
        let filt_sample = z[0] + (num[0] * sample);
        for i in 0..filter_len - 1 {
            z[i] = z[i + 1] + (num[i + 1] * sample) - (den[i + 1] * filt_sample);
        }
        (filt_sample - sample) * scale + sample
    }

    /// Filter a signal, reading from one slice and writing into another.
    /// # Arguments
    /// - `src` - The slice containing the signal to be filtered.
    /// - `dst` - The slice to place the filtered signal into. This must be the same
size as `src`.
    /// - `initial` - The initial steady-state value of the filter.
    /// - `scale` - Scale the effect of the filter on the output by this amount.
    /// - `delay` - Offset the filter output backwards (to the left) by this amount.
    pub fn filter_signal_into(
        &self,
        src: &[f32],
        dst: &mut [f32],
        initial: f32,
        scale: f32,
        delay: usize,
    ) {
        if dst.len() != src.len() {
            panic!(
                "Source slice is {} samples but destination is {} samples",
                src.len(),
                src.len()
            );
        }

        let filter_len = usize::max(self.num.len(), self.den.len());
```

```
        let mut z = self.initial_condition(initial);

        for i in 0..(src.len() + delay) {
            let sample = unsafe { src.get_unchecked(i.min(src.len() - 1)) };
            let filt_sample =
                Self::filter_sample(filter_len, &self.num, &self.den, &mut z, *sample,
scale);
            if i >= delay {
                dst[i - delay] = filt_sample;
            }
        }
    }
    #[inline(always)]
    fn filter_signal_in_place_impl<const ROWS: usize>(
        signal: &mut [&mut [f32]; ROWS],
        num: &[f32],
        den: &[f32],
        z: [&mut [f32]; ROWS],
        scale: f32,
        delay: usize,
    ) {
        let filter_len = num.len();
        for row_idx in 0..ROWS {
            let signal = &mut signal[row_idx];
            for i in 0..(signal.len() + delay) {
                let sample = unsafe { signal.get_unchecked(i.min(signal.len() - 1)) };
                let filt_sample =
                    Self::filter_sample(filter_len, num, den, z[row_idx], *sample,
scale);
                if i >= delay {
                    signal[i - delay] = filt_sample;
                }
            }
        }
    }

    #[inline(never)]
    fn filter_signal_in_place_fixed_size<const SIZE: usize, const ROWS: usize>(
        &self,
        signal: &mut [&mut [f32]; ROWS],
        initial: [f32; ROWS],
        scale: f32,
        delay: usize,
    ) {
        let mut z_rows = [[0f32; SIZE]; ROWS];
        z_rows.iter_mut().zip(initial).for_each(|(z, initial)| {
            *z = self.initial_condition(initial).try_into().unwrap();
        });
```

```
        let z_rows_ref = each_mut::<[f32; SIZE], ROWS>(&mut z_rows).map(|z|
z.as_mut_slice());

        Self::filter_signal_in_place_impl::<ROWS>(
            signal, &self.num, &self.den, z_rows_ref, scale, delay,
        );
    }

    #[inline(always)]
    unsafe fn filter_signal_in_place_fixed_size_simdx4<S: F32x4, const ROWS: usize>(
        &self,
        signal: &mut [&mut [f32]; ROWS],
        initial: [f32; ROWS],
        scale: f32,
        delay: usize,
    ) {
        let mut z_rows = [[0f32; 4]; ROWS];
        z_rows.iter_mut().zip(initial).for_each(|(z, initial)| {
            *z = self.initial_condition(initial).try_into().unwrap();
        });
        let mut z = z_rows;

        let mut num: [f32; 4] = [0f32; 4];
        num.copy_from_slice(&self.num);

        let mut den: [f32; 4] = [0f32; 4];
        den.copy_from_slice(&self.den);

        let num = S::load(&num);
        let den = S::load(&den);
        let scale_b = S::set1(scale);
        let width = signal[0].len();

        for i in 0..(width + delay) {
            for j in 0..ROWS {
                let mut zmm = S::load4(&z[j]);
                // Either the loop bound extending past items.len() or the min() call
seems to prevent the optimizer from
                // determining that we're in-bounds here. Since i.min(items.len() - 1)
never exceeds items.len() - 1 by
                // definition, this is safe.
                let sample = S::load1(signal[j].get_unchecked(i.min(width - 1)));
                let filt_sample = num.mul_add(sample, zmm).swizzle(0, 0, 0, 0);

                // Add the sample * the numerator, subtract the filtered sample * the
denominator
                zmm = num.mul_add(sample, zmm);
                zmm = den.neg_mul_add(filt_sample, zmm);
```

```
            // Shift it all over
            zmm = zmm.swizzle(1, 2, 3, 0);

            // Zero out the last element
            zmm = zmm.insert::<3>(0.0);

            zmm.store(&mut z[j]);

            if i >= delay {
                let samp_diff = filt_sample - sample;
                let final_samp = samp_diff.mul_add(scale_b, sample);
                final_samp.store1(signal[j].get_unchecked_mut(i - delay));
            }
        }
    }
}

#[cfg(target_arch = "x86_64")]
#[target_feature(enable = "sse4.1")]
/// Process the signal using SSE4.1 intrinsics.
unsafe fn filter_signal_dispatch_sse41<const ROWS: usize>(
    &self,
    signal: &mut [&mut [f32]; ROWS],
    initial: [f32; ROWS],
    scale: f32,
    delay: usize,
) {
    use crate::f32x4::x86_64::SseF32x4;
    unsafe {
        self.filter_signal_in_place_fixed_size_simdx4::<SseF32x4, ROWS>(
            signal, initial, scale, delay,
        );
    }
}

#[cfg(target_arch = "x86_64")]
#[target_feature(enable = "avx2", enable = "fma")]
/// Process the signal using AVX2 intrinsics.
unsafe fn filter_signal_dispatch_avx2<const ROWS: usize>(
    &self,
    signal: &mut [&mut [f32]; ROWS],
    initial: [f32; ROWS],
    scale: f32,
    delay: usize,
) {
    use crate::f32x4::x86_64::AvxF32x4;
    unsafe {
        self.filter_signal_in_place_fixed_size_simdx4::<AvxF32x4, ROWS>(
            signal, initial, scale, delay,
```

```
            );
        }
    }

    #[cfg(target_arch = "aarch64")]
    #[target_feature(enable = "neon")]
    /// Process the signal using NEON intrinsics.
    unsafe fn filter_signal_dispatch_neon<const ROWS: usize>(
        &self,
        signal: &mut [&mut [f32]; ROWS],
        initial: [f32; ROWS],
        scale: f32,
        delay: usize,
    ) {
        use crate::f32x4::aarch64::ArmF32x4;
        unsafe {
            self.filter_signal_in_place_fixed_size_simdx4::<ArmF32x4, ROWS>(
                signal, initial, scale, delay,
            );
        }
    }

    pub fn filter_signal_in_place<const ROWS: usize>(
        &self,
        signal: &mut [&mut [f32]; ROWS],
        initial: [f32; ROWS],
        scale: f32,
        delay: usize,
    ) {
        let filter_len = usize::max(self.num.len(), self.den.len());

        if filter_len == 4 {
            #[cfg(target_arch = "x86_64")]
            match get_supported_simd_type() {
                SupportedSimdType::Sse41 => {
                    unsafe {
                        self.filter_signal_dispatch_sse41::<ROWS>(signal, initial,
scale, delay);
                    }
                    return;
                }
                SupportedSimdType::Avx2 => {
                    unsafe {
                        self.filter_signal_dispatch_avx2::<ROWS>(signal, initial,
scale, delay);
                    }
                    return;
                }
                _ => {}
```

```
            }

            #[cfg(target_arch = "aarch64")]
            if get_supported_simd_type() == SupportedSimdType::Neon {
                unsafe {
                    self.filter_signal_dispatch_neon::<ROWS>(signal, initial, scale,
delay);
                }
                return;
            }

            self.filter_signal_in_place_fixed_size::<4, ROWS>(signal, initial, scale,
delay);
        } else {
            match filter_len {
                // Specialize fixed-size implementations for filter sizes 1-8
                1 => {
                    self.filter_signal_in_place_fixed_size::<1, ROWS>(signal, initial,
scale, delay)
                }
                2 => {
                    self.filter_signal_in_place_fixed_size::<2, ROWS>(signal, initial,
scale, delay)
                }
                3 => {
                    self.filter_signal_in_place_fixed_size::<3, ROWS>(signal, initial,
scale, delay)
                }
                // 4 is covered in the branch above
                5 => {
                    self.filter_signal_in_place_fixed_size::<5, ROWS>(signal, initial,
scale, delay)
                }
                6 => {
                    self.filter_signal_in_place_fixed_size::<6, ROWS>(signal, initial,
scale, delay)
                }
                7 => {
                    self.filter_signal_in_place_fixed_size::<7, ROWS>(signal, initial,
scale, delay)
                }
                8 => {
                    self.filter_signal_in_place_fixed_size::<8, ROWS>(signal, initial,
scale, delay)
                }
                _ => {
                    let mut z: [Vec<f32>; ROWS] = initial
                        .into_iter()
                        .map(|init| self.initial_condition(init))
```

```
                        .collect::<Vec<Vec<_>>>()
                        .try_into()
                        .unwrap();
                    let z: [&mut [f32]; ROWS] = z
                        .iter_mut()
                        .map(|z| z.as_mut_slice())
                        .collect::<Vec<_>>()
                        .try_into()
                        .unwrap();
                    Self::filter_signal_in_place_impl::<ROWS>(
                        signal, &self.num, &self.den, z, scale, delay,
                    );
                }
            }
        }
    }
}

impl std::ops::Mul<&TransferFunction> for &TransferFunction {
    type Output = TransferFunction;

    fn mul(self, rhs: &TransferFunction) -> Self::Output {
        TransferFunction::new(
            polynomial_multiply(trim_zeros(&self.num), trim_zeros(&rhs.num)),
            polynomial_multiply(trim_zeros(&self.den), trim_zeros(&rhs.den)),
        )
    }
}
```

Lib.rs

```
mod f32x4;
mod filter;
pub mod ntsc;
mod random;
pub mod settings;
mod shift;
pub mod yiq_fielding;

#[macro_use]
extern crate num_derive;

pub use num_traits::cast::{FromPrimitive, ToPrimitive};
```

ntsrc.rs

```
use std::convert::identity;
use std::{collections::VecDeque, ops::RangeInclusive};

use core::f32::consts::PI;
use rand::{Rng, RngCore, SeedableRng};
use rand_xoshiro::Xoshiro256PlusPlus;
use rayon::prelude::*;
use simdnoise::{NoiseBuilder, Settings as NoiseSettings, SimplexSettings};

use crate::{
    filter::TransferFunction,
    random::{Geometric, Seeder},
    shift::{shift_row, shift_row_to, BoundaryHandling},
    yiq_fielding::{BlitInfo, PixelFormat, YiqField, YiqOwned, YiqView},
};

pub use crate::settings::standard::*;

const NTSC_RATE: f32 = (315000000.00 / 88.0) * 4.0;

pub fn make_lowpass(cutoff: f32, rate: f32) -> TransferFunction {
    let time_interval = 1.0 / rate;
    let tau = (cutoff * 2.0 * PI).recip();
    let alpha = time_interval / (tau + time_interval);

    TransferFunction::new(vec![alpha], vec![1.0, -(1.0 - alpha)])
}

pub fn make_lowpass_triple(cutoff: f32, rate: f32) -> TransferFunction {
    make_lowpass(cutoff, rate).cascade_self(3)
}

/// Construct a lowpass filter of the filter type given in the settings.
fn make_lowpass_for_type(cutoff: f32, rate: f32, filter_type: FilterType) ->
TransferFunction {
    match filter_type {
        FilterType::ConstantK => make_lowpass_triple(cutoff, rate),
        FilterType::Butterworth => make_butterworth_filter(cutoff, rate),
    }
}

/// Create an IIR notch filter.
pub fn make_notch_filter(freq: f32, quality: f32) -> TransferFunction {
    // Adapted from scipy and simplified
    //
    if !(0.0..=1.0).contains(&freq) {
        panic!("Frequency outside valid range");
    }
```

```
    let bandwidth = (freq / quality) * PI;
    let freq = freq * PI;

    let beta = (bandwidth * 0.5).tan();

    let gain = (1.0 + beta).recip();

    let num = vec![gain, -2.0 * freq.cos() * gain, gain];
    let den = vec![1.0, -2.0 * freq.cos() * gain, 2.0 * gain - 1.0];

    TransferFunction::new(num, den)
}

/// Create a 2nd-order Butterworth filter.
pub fn make_butterworth_filter(cutoff: f32, rate: f32) -> TransferFunction {
    let coeffs = biquad::Coefficients::<f32>::from_params(
        biquad::Type::LowPass,
        biquad::Hertz::<f32>::from_hz(rate).unwrap(),
        biquad::Hertz::<f32>::from_hz(cutoff.min(rate * 0.5)).unwrap(),
        biquad::Q_BUTTERWORTH_F32,
    )
    .unwrap();
    TransferFunction::new(
        [coeffs.b0, coeffs.b1, coeffs.b2],
        [1.0, coeffs.a1, coeffs.a2],
    )
}

#[allow(dead_code)]
enum InitialCondition {
    /// Convenience value--just use 0.
    Zero,
    /// Set the initial filter condition to a constant.
    Constant(f32),
    /// Set the initial filter condition to that of the first sample to be filtered.
    FirstSample,
}

fn filter_plane(
    plane: &mut [f32],
    width: usize,
    filter: &TransferFunction,
    initial: InitialCondition,
    scale: f32,
    delay: usize,
) {
    if filter.should_use_row_chunks() {
        filter_plane_with_rows::<8>(plane, width, filter, initial, scale, delay)
    } else {
```

```
        filter_plane_with_rows::<1>(plane, width, filter, initial, scale, delay)
    }
}

fn filter_plane_with_rows<const ROWS: usize>(
    plane: &mut [f32],
    width: usize,
    filter: &TransferFunction,
    initial: InitialCondition,
    scale: f32,
    delay: usize,
) {
    let mut row_chunks = plane.par_chunks_exact_mut(width * ROWS);

    // Process the remainder of the rows. `for_each` consumes the iterator, so we have
to at least call `take_remainder`
    // before that. We might as well do the processing while we're at it.
    row_chunks
        .take_remainder()
        .par_chunks_exact_mut(width)
        .for_each(|row| {
            let initial = match initial {
                InitialCondition::Zero => 0.0,
                InitialCondition::Constant(c) => c,
                InitialCondition::FirstSample => row[0],
            };
            filter.filter_signal_in_place::<1>(&mut [row], [initial], scale, delay)
        });

    // Process the row in chunks, each `ROWS` rows tall.
    row_chunks.for_each(|rows| {
        let mut row_chunks: Vec<&mut [f32]> = rows.chunks_exact_mut(width).collect();
        let rows: &mut [&mut [f32]; ROWS] =
row_chunks.as_mut_slice().try_into().unwrap();

        let mut initial_rows: [f32; ROWS] = [0f32; ROWS];
        for i in 0..ROWS {
            initial_rows[i] = match initial {
                InitialCondition::Zero => 0.0,
                InitialCondition::Constant(c) => c,
                InitialCondition::FirstSample => rows[i][0],
            };
        }

        filter.filter_signal_in_place::<ROWS>(rows, initial_rows, scale, delay);
    });
}
```

```
/// Common settings/info that many passes need to use. Passed to each individual
effect function as needed.
struct CommonInfo {
    /// Random seed.
    seed: u64,
    /// Current frame index.
    frame_num: usize,
    /// The "bandwidth scale" setting, used mainly for IIR filters.
    bandwidth_scale: f32,
}

/// Apply a lowpass filter to the luminance (Y) plane before the Y, I, and Q planes
are modulated together. This should
/// reduce color fringing/rainbow artifacts, which may or may not be what you want.
Note that this is *not* the "Luma
/// smear" setting--that's its own function (`luma_smear`).
fn luma_filter(frame: &mut YiqView, filter_mode: LumaLowpass) {
    match filter_mode {
        LumaLowpass::None => {}
        LumaLowpass::Box => {
            frame.y.par_chunks_mut(frame.dimensions.0).for_each(|y| {
                let mut delay = VecDeque::<f32>::with_capacity(4);
                delay.push_back(16.0 / 255.0);
                delay.push_back(16.0 / 255.0);
                delay.push_back(y[0]);
                delay.push_back(y[1]);
                let mut sum: f32 = delay.iter().sum();
                let width = y.len();

                for index in 0..width {
                    // Box-blur the signal.
                    let c = y[usize::min(index + 2, width - 1)];
                    sum -= delay.pop_front().unwrap();
                    delay.push_back(c);
                    sum += c;
                    y[index] = sum * 0.25;
                }
            });
        }
        LumaLowpass::Notch => filter_plane(
            frame.y,
            frame.dimensions.0,
            &make_notch_filter(0.5, 2.0),
            InitialCondition::FirstSample,
            1.0,
            0,
        ),
    }
}
```

```rust
/// Apply a lowpass filter to the input chroma, emulating broadcast NTSC's bandwidth
cutoffs.
/// (Well, almost--Wikipedia (https://en.wikipedia.org/wiki/YIQ) puts the Q bandwidth
at 0.4 MHz, not 0.6. Although
/// that statement seems unsourced and I can't find any info on it...)
fn composite_chroma_lowpass(frame: &mut YiqView, info: &CommonInfo, filter_type:
FilterType) {
    let i_filter = make_lowpass_for_type(1300000.0, NTSC_RATE * info.bandwidth_scale,
filter_type);
    let q_filter = make_lowpass_for_type(600000.0, NTSC_RATE * info.bandwidth_scale,
filter_type);

    let width = frame.dimensions.0;

    filter_plane(frame.i, width, &i_filter, InitialCondition::Zero, 1.0, 2);
    filter_plane(frame.q, width, &q_filter, InitialCondition::Zero, 1.0, 4);
}

/// Apply a less intense lowpass filter to the input chroma.
fn composite_chroma_lowpass_lite(frame: &mut YiqView, info: &CommonInfo, filter_type:
FilterType) {
    let filter = make_lowpass_for_type(2600000.0, NTSC_RATE * info.bandwidth_scale,
filter_type);

    let width = frame.dimensions.0;

    filter_plane(frame.i, width, &filter, InitialCondition::Zero, 1.0, 1);
    filter_plane(frame.q, width, &filter, InitialCondition::Zero, 1.0, 1);
}

/// Calculate the chroma subcarrier phase for a given row/field.
fn chroma_phase_shift(
    scanline_phase_shift: PhaseShift,
    offset: i32,
    frame_num: usize,
    line_num: usize,
) -> usize {
    (match scanline_phase_shift {
        PhaseShift::Degrees90 | PhaseShift::Degrees270 => {
            (frame_num as i32 + offset + ((line_num as i32) >> 1)) & 3
        }
        PhaseShift::Degrees180 => (((frame_num + line_num) & 2) as i32 + offset) & 3,
        PhaseShift::Degrees0 => 0,
    } & 3) as usize
}

const I_MULT: [f32; 4] = [1.0, 0.0, -1.0, 0.0];
const Q_MULT: [f32; 4] = [0.0, 1.0, 0.0, -1.0];
```

```rust
/// Modulate a single line of the chrominance signal into the luminance signal.
fn chroma_into_luma_line(y: &mut [f32], i: &mut [f32], q: &mut [f32], xi: usize) {
    y.iter_mut()
        .zip(i.iter_mut().zip(q))
        .enumerate()
        .for_each(|(index, (y, (i, q)))| {
            let phase = (index + (xi & 3)) & 3;
            *y += *i * I_MULT[phase] + *q * Q_MULT[phase];
            // *i = 0.0;
            // *q = 0.0;
        });
}

/// Modulate the chrominance signal (I and Q planes) into the Y (luminance) plane.
/// TODO: Make the chroma carrier's frequency/sample rate configurable.
fn chroma_into_luma(
    yiq: &mut YiqView,
    info: &CommonInfo,
    phase_shift: PhaseShift,
    phase_offset: i32,
) {
    let width = yiq.dimensions.0;

    let y_lines = yiq.y.par_chunks_mut(width);
    let i_lines = yiq.i.par_chunks_mut(width);
    let q_lines = yiq.q.par_chunks_mut(width);

    y_lines
        .zip(i_lines.zip(q_lines))
        .enumerate()
        .for_each(|(index, (y, (i, q)))| {
            let xi = chroma_phase_shift(phase_shift, phase_offset, info.frame_num,
index * 2);

            chroma_into_luma_line(y, i, q, xi);
        });
}

/// Demodulate the chrominance (I and Q) signals from a combined NTSC signal, looking
at a single source pixel at the
/// given index and writing into the destination I and Q plane pixels as well as their
immediate neighbors.
/// TODO: Accumulating the signal this way seems to add a data dependency. However, I
believe doing this in a
/// parallelizable way would require *two* scratch buffers.
#[inline(always)]
fn demodulate_chroma(chroma: f32, index: usize, xi: usize, i: &mut [f32], q: &mut
[f32]) {
```

```
    let width = i.len();

    let offset = (index + (xi & 3)) & 3;

    let i_modulated = -(chroma * I_MULT[offset]);
    let q_modulated = -(chroma * Q_MULT[offset]);

    // TODO: ntscQT seems to mess this up, giving chroma a "jagged" look reminiscent
of the "dot crawl" artifact.
    // Is that worth trying to replicate, or should it just be left like this?
    if index < width - 1 {
        i[index + 1] = i_modulated * 0.5;
        q[index + 1] = q_modulated * 0.5;
    }
    i[index] += i_modulated;
    q[index] += q_modulated;
    if index > 0 {
        i[index - 1] += i_modulated * 0.5;
        q[index - 1] += q_modulated * 0.5;
    }
}

/// Demodulate the chrominance signal using a box filter to separate it out.
fn luma_into_chroma_line_box(
    y: &mut [f32],
    i: &mut [f32],
    q: &mut [f32],
    scratch: &mut [f32],
    xi: usize,
) {
    let width = y.len();
    for index in 0..width {
        let c = y[usize::min(index + 2, width - 1)];
        let area = [
            y.get(index.wrapping_sub(1))
                .cloned()
                .unwrap_or(16.0 / 255.0),
            y[index],
            y[(index + 1).min(width - 1)],
            y[(index + 2).min(width - 1)],
        ];
        scratch[index] = area.iter().sum::<f32>() * 0.25;
        let chroma = c - scratch[index];
        demodulate_chroma(chroma, index, xi, i, q);
    }
    y.copy_from_slice(scratch);
}

/// Demodulate the chroma signal from the Y (luma) plane back into the I and Q planes.
```

```
/// TODO: Make the chroma carrier's frequency/sample rate configurable.
fn luma_into_chroma(
    yiq: &mut YiqView,
    info: &CommonInfo,
    filter_mode: ChromaDemodulationFilter,
    phase_shift: PhaseShift,
    phase_offset: i32,
) {
    let width = yiq.dimensions.0;

    match filter_mode {
        ChromaDemodulationFilter::Box => {
            let y_lines = yiq.y.par_chunks_mut(width);
            let i_lines = yiq.i.par_chunks_mut(width);
            let q_lines = yiq.q.par_chunks_mut(width);
            let scratch_lines = yiq.scratch.par_chunks_mut(width);

            y_lines
                .zip(i_lines.zip(q_lines.zip(scratch_lines)))
                .enumerate()
                .for_each(|(index, (y, (i, (q, scratch))))| {
                    let xi =
                        chroma_phase_shift(phase_shift, phase_offset, info.frame_num,
index * 2);

                    luma_into_chroma_line_box(y, i, q, scratch, xi);
                });
        }
        ChromaDemodulationFilter::Notch => {
            // Apply a notch filter to the signal to remove the high-frequency chroma
carrier, and store it in the
            // scratch buffer. We can then get *just* the chroma by subtracting the
filtered signal from the original.
            let scratch = &mut yiq.scratch;
            let filter: TransferFunction = make_notch_filter(0.5, 2.0);
            scratch.copy_from_slice(yiq.y);
            filter_plane(yiq.y, width, &filter, InitialCondition::Zero, 1.0, 0);

            let y_lines = yiq.y.par_chunks_mut(width);
            let i_lines = yiq.i.par_chunks_mut(width);
            let q_lines = yiq.q.par_chunks_mut(width);
            let scratch_lines = scratch.par_chunks_mut(width);

            y_lines
                .zip(i_lines.zip(q_lines.zip(scratch_lines)))
                .enumerate()
                .for_each(|(index, (y, (i, (q, scratch))))| {
                    let xi =
```

```
                        chroma_phase_shift(phase_shift, phase_offset, info.frame_num,
index * 2);

                    for index in 0..width {
                        let chroma = y[index] - scratch[index];
                        demodulate_chroma(chroma, index, xi, i, q);
                    }
                });
        }
        ChromaDemodulationFilter::OneLineComb => {
            let delay = &mut yiq.scratch;
            // "Reflect" line 2 to line 0, so that the chroma is properly demodulated
for line 0.
            // A comb filter requires the phase of the chroma carrier to alternate per
line, so simply repeating line 1
            // wouldn't work.
            delay[0..width].copy_from_slice(&yiq.y[width..width * 2]);
            delay[width..].copy_from_slice(&yiq.y[0..yiq.y.len() - width]);

            let y_lines = yiq.y.par_chunks_mut(width);
            let i_lines = yiq.i.par_chunks_mut(width);
            let q_lines = yiq.q.par_chunks_mut(width);
            let delay_lines = delay.par_chunks_mut(width);
            y_lines
                .zip(i_lines.zip(q_lines.zip(delay_lines)))
                .enumerate()
                .for_each(|(line_index, (y, (i, (q, delay))))| {
                    for index in 0..width {
                        let blended = (y[index] + delay[index]) * 0.5;
                        let chroma = blended - y[index];
                        y[index] = blended;
                        let xi = chroma_phase_shift(
                            phase_shift,
                            phase_offset,
                            info.frame_num,
                            line_index * 2,
                        );
                        demodulate_chroma(chroma, index, xi, i, q);
                    }
                });
        }
        ChromaDemodulationFilter::TwoLineComb => {
            let height = yiq.num_rows();
            let modulated = &mut yiq.scratch;
            modulated.copy_from_slice(yiq.y);

            let y_lines = yiq.y.par_chunks_mut(width);
            let i_lines = yiq.i.par_chunks_mut(width);
            let q_lines = yiq.q.par_chunks_mut(width);
```

```
            y_lines
                .zip(i_lines.zip(q_lines))
                .enumerate()
                .for_each(|(line_index, (y, (i, q)))| {
                    // For the first line, both prev_line and next_line point to the
second line. This effecively makes
                    // it a one-line comb filter for that line. See the comment above
in the one-line comb filter for
                    // why we do this.
                    let prev_index = if line_index == 0 { 1 } else { line_index - 1 };

                    // Similar for the last line.
                    let next_index = if line_index == height - 1 {
                        height - 2
                    } else {
                        line_index + 1
                    };

                    let prev_line = &modulated[prev_index * width..(prev_index + 1) *
width];
                    let cur_line = &modulated[line_index * width..(line_index + 1) *
width];
                    let next_line = &modulated[next_index * width..(next_index + 1) *
width];

                    for sample_index in 0..width {
                        let cur_sample = cur_line[sample_index];
                        let blended = (cur_sample * 0.5)
                            + (prev_line[sample_index] * 0.25)
                            + (next_line[sample_index] * 0.25);
                        let chroma = blended - cur_sample;
                        y[sample_index] = blended;

                        let xi = chroma_phase_shift(
                            phase_shift,
                            phase_offset,
                            info.frame_num,
                            line_index * 2,
                        );
                        demodulate_chroma(chroma, sample_index, xi, i, q);
                    }
                });
        }
    };
}

/// Blur the luminance plane using a lowpass filter.
fn luma_smear(yiq: &mut YiqView, info: &CommonInfo, amount: f32) {
    let lowpass = make_lowpass(f32::exp2(-4.0 * amount) * 0.25, info.bandwidth_scale);
```

```
    filter_plane(
        yiq.y,
        yiq.dimensions.0,
        &lowpass,
        InitialCondition::Zero,
        1.0,
        0,
    );
}

/// We use a seeded RNG to generate random noise deterministically, but we don't want
every pass which uses noise to use
/// the *same* noise. Each pass gets its own random seed which is mixed into the RNG.
mod noise_seeds {
    pub const VIDEO_COMPOSITE: u64 = 0;
    pub const HEAD_SWITCHING: u64 = 2;
    pub const TRACKING_NOISE: u64 = 3;
    pub const VIDEO_CHROMA_PHASE: u64 = 4;
    pub const EDGE_WAVE: u64 = 5;
    pub const SNOW: u64 = 6;
    pub const CHROMA_LOSS: u64 = 7;
    pub const HEAD_SWITCHING_MID_LINE_JITTER: u64 = 8;

    pub const VIDEO_CHROMA_I: u64 = 1;
    pub const VIDEO_CHROMA_Q: u64 = 9;
    pub const VIDEO_LUMA: u64 = 10;
}

/// Helper function to apply gradient noise to a single row of a single plane.
fn video_noise_line(
    row: &mut [f32],
    scratch: &mut [f32],
    seeder: &Seeder,
    index: usize,
    frequency: f32,
    intensity: f32,
    detail: u32,
) {
    let width = row.len();
    let mut rng = Xoshiro256PlusPlus::seed_from_u64(seeder.clone().mix(index as
u64).finalize());
    let noise_seed = rng.next_u32();
    let offset = rng.gen::<f32>() * width as f32;

    NoiseBuilder::fbm_1d_offset(offset, width)
        .with_seed(noise_seed as i32)
        .with_freq(frequency)
        .with_octaves(detail.clamp(1, 5) as u8)
```

```
        // Yes, they got the lacunarity backwards by making it apply to frequency
instead of scale.
        // 2.0 *halves* the scale each time because it doubles the frequency.
        .with_lacunarity(2.0)
        .generate_into(scratch);

    row.iter_mut().enumerate().for_each(|(x, pixel)| {
        *pixel += scratch[x] * 0.25 * intensity;
    });
}

/// Add gradient noise to an NTSC-encoded (composite) signal.
fn composite_noise(yiq: &mut YiqView, info: &CommonInfo, noise_settings:
&FbmNoiseSettings) {
    let width = yiq.dimensions.0;
    let seeder = Seeder::new(info.seed)
        .mix(noise_seeds::VIDEO_COMPOSITE)
        .mix(info.frame_num);

    yiq.y
        .par_chunks_mut(width)
        .zip(yiq.scratch.par_chunks_exact_mut(width))
        .enumerate()
        .for_each(|(index, (row, scratch))| {
            video_noise_line(
                row,
                scratch,
                &seeder,
                index,
                noise_settings.frequency / info.bandwidth_scale,
                noise_settings.intensity,
                noise_settings.detail,
            );
        });
}

/// Add gradient noise to a single plane of a de-modulated signal.
fn plane_noise(
    plane: &mut [f32],
    scratch: &mut [f32],
    width: usize,
    info: &CommonInfo,
    settings: &FbmNoiseSettings,
    noise_seed: u64,
) {
    let seeder = Seeder::new(info.seed).mix(noise_seed).mix(info.frame_num);

    plane
        .par_chunks_mut(width)
```

```
        .zip(scratch.par_chunks_mut(width))
        .enumerate()
        .for_each(|(index, (row, scratch))| {
            video_noise_line(
                row,
                scratch,
                &seeder,
                index,
                settings.frequency / info.bandwidth_scale,
                settings.intensity,
                settings.detail,
            );
        });
}

/// Emulate timing error in the chrominance carrier for a single line.
fn chroma_phase_offset_line(i: &mut [f32], q: &mut [f32], offset: f32) {
    // Phase shift angle in radians. Mapped so that an intensity of 1.0 is a phase
shift ranging from a full
    // rotation to the left, to a full rotation to the right.
    let phase_shift = offset * PI * 2.0;
    let (sin_angle, cos_angle) = phase_shift.sin_cos();

    for (i, q) in i.iter_mut().zip(q.iter_mut()) {
        // Treat (i, q) as a 2D vector and rotate it by the phase shift amount.
        let rotated_i = (*i * cos_angle) - (*q * sin_angle);
        let rotated_q = (*i * sin_angle) + (*q * cos_angle);

        *i = rotated_i;
        *q = rotated_q;
    }
}

/// Emulate timing error in the chrominance carrier, which results in a hue shift.
fn chroma_phase_error(yiq: &mut YiqView, intensity: f32) {
    let width = yiq.dimensions.0;

    yiq.i
        .par_chunks_mut(width)
        .zip(yiq.q.par_chunks_mut(width))
        .for_each(|(i, q)| {
            chroma_phase_offset_line(i, q, intensity);
        });
}

/// Add per-scanline chroma phase error.
fn chroma_phase_noise(yiq: &mut YiqView, info: &CommonInfo, intensity: f32) {
    let width = yiq.dimensions.0;
    let seeder = Seeder::new(info.seed)
```

```
        .mix(noise_seeds::VIDEO_CHROMA_PHASE)
        .mix(info.frame_num);

    yiq.i
        .par_chunks_mut(width)
        .zip(yiq.q.par_chunks_mut(width))
        .enumerate()
        .for_each(|(index, (i, q))| {
            // Phase shift angle in radians. Mapped so that an intensity of 1.0 is a
phase shift ranging from a full
            // rotation to the left, to a full rotation to the right.
            let phase_shift = (seeder.clone().mix(index).finalize::<f32>() - 0.5) *
2.0 * intensity;

            chroma_phase_offset_line(i, q, phase_shift);
        });
}

/// Emulate VHS head-switching at the bottom of the image.
fn head_switching(
    yiq: &mut YiqView,
    info: &CommonInfo,
    num_rows: usize,
    offset: usize,
    shift: f32,
    mid_line: Option<&HeadSwitchingMidLineSettings>,
) {
    if offset > num_rows {
        return;
    }
    let num_affected_rows = num_rows - offset;

    let width = yiq.dimensions.0;
    let height = yiq.num_rows();
    // Handle cases where the number of affected rows exceeds the number of actual
rows in the image
    let start_row = height.max(num_affected_rows) - num_affected_rows;
    let affected_rows = &mut yiq.y[start_row * width..];
    let cut_off_rows = if num_affected_rows > height {
        num_affected_rows - height
    } else {
        0
    };

    let seeder = Seeder::new(info.seed)
        .mix(noise_seeds::HEAD_SWITCHING)
        .mix(info.frame_num);

    affected_rows
```

```
        .par_chunks_mut(width)
        .enumerate()
        .for_each(|(index, row)| {
            let index = num_affected_rows - (index + cut_off_rows);
            let row_shift = shift * ((index + offset) as f32 / num_rows as
f32).powf(1.5);
            let noisy_shift = (row_shift +
(seeder.clone().mix(index).finalize::<f32>() - 0.5))
                * info.bandwidth_scale;

            // because if-let chains are unstable :(
            if index == num_affected_rows && mid_line.is_some() {
                let mid_line = mid_line.unwrap();
                // Shift the entire row, but only copy back a portion of it.
                let mut tmp_row = vec![0.0; width];
                shift_row_to(
                    row,
                    &mut tmp_row,
                    noisy_shift,
                    BoundaryHandling::Constant(0.0),
                );

                let seeder = Seeder::new(info.seed)
                    .mix(noise_seeds::HEAD_SWITCHING_MID_LINE_JITTER)
                    .mix(info.frame_num);

                // Average two random numbers to bias the result towards the middle
                let jitter_rand = (seeder.clone().mix(0).finalize::<f32>()
                    + seeder.clone().mix(1).finalize::<f32>())
                    * 0.5;
                let jitter = (jitter_rand - 0.5) * mid_line.jitter;

                let copy_start = (width as f32 * (mid_line.position + jitter)) as
usize;
                if copy_start > width {
                    return;
                }
                row[copy_start..].copy_from_slice(&tmp_row[copy_start..]);

                // Add a transient where the head switch is supposed to start
                let transient_intensity = (seeder.clone().mix(0).finalize::<f32>() +
0.5) * 0.5;
                let transient_len = 16.0 * info.bandwidth_scale;

                for i in copy_start..(copy_start + transient_len.ceil() as
usize).min(width) {
                    let x = (i - copy_start) as f32;
                    row[i] += (1.0 - (x / transient_len)).powi(3) *
transient_intensity;
```

```
                }
            } else {
                shift_row(row, noisy_shift, BoundaryHandling::Constant(0.0));
            }
        });
}

/// Helper function for generating "snow"/transient speckles.
fn row_speckles(
    row: &mut [f32],
    rng: &mut Xoshiro256PlusPlus,
    intensity: f32,
    anisotropy: f32,
    bandwidth_scale: f32,
) {
    let intensity = intensity as f64;
    let anisotropy = anisotropy as f64;
    const TRANSIENT_LEN_RANGE: RangeInclusive<f32> = 8.0..=64.0;

    // Anisotropy controls how much the snow appears "clumped" within given lines vs.
appearing independently across
    // lines.
    //
    // Transition smoothly from a flat function that always returns `intensity`, to a
step function that returns
    // 1.0 with a probability of `intensity` and 0.0 with a probability of `1.0 -
intensity`. In-between states
    // look like S-curves with increasing sharpness.
    // As a bonus, the integral of this function over (0, 1) as we transition from 0%
to 100% anisotropy is *almost*
    // constant, meaning there's approximately the same amount of snow each time.
    let logistic_factor = ((rng.gen::<f64>() - intensity)
        / (intensity * (1.0 - intensity) * (1.0 - anisotropy)))
        .exp();
    // Intensity of the "snow" for this specific line
    let mut line_snow_intensity =
        anisotropy / (1.0 + logistic_factor) + intensity * (1.0 - anisotropy);

    // At maximum intensity (1.0), the line just gets completely whited out. 0.125
intensity looks more reasonable.
    line_snow_intensity *= 0.125;
    line_snow_intensity = line_snow_intensity.clamp(0.0, 1.0);
    if line_snow_intensity <= 0.0 {
        return;
    }

    // Turn each pixel into "snow" with probability snow_intensity * intensity_scale
    // We can simulate the distance between each "snow" pixel with a geometric
distribution which avoids having to
```

```
    // loop over every pixel:
    // https://en.wikipedia.org/wiki/Geometric_distribution
    let dist = Geometric::new(line_snow_intensity);
    // Start leftwards of the visible region to simulate transients that may have
started before it. This avoids
    // transients being sparser towards the leftmost edge.
    let mut pixel_idx = (-TRANSIENT_LEN_RANGE.end()).floor() as isize;
    loop {
        pixel_idx += rng.sample(&dist).min(isize::MAX as usize) as isize;
        if pixel_idx >= row.len() as isize {
            break;
        }

        let transient_len: f32 = rng.gen_range(TRANSIENT_LEN_RANGE) * bandwidth_scale;
        let transient_freq = rng.gen_range(transient_len * 3.0..=transient_len * 5.0);
        let pixel_idx_end = pixel_idx + transient_len.ceil() as isize;

        // Each transient gets its own RNG to determine the intensity of each pixel
within it.
        // This is to prevent the length of each transient from affecting the random
state of the subsequent
        // transient, which can cause the snow to "jitter" when changing the
"bandwidth scale" setting.
        rng.jump();
        let mut transient_rng = rng.clone();

        for i in pixel_idx.clamp(0, row.len() as isize)..pixel_idx_end.clamp(0,
row.len() as isize)
        {
            let x = (i - pixel_idx) as f32;
            // Simulate transient with sin(pi*x / 4) * (1 - x/len)^2
            row[i as usize] += ((x * PI) / transient_freq).cos()
                * (1.0 - x / transient_len).powi(2)
                * transient_rng.gen_range(-1.0..2.0);
        }

        // Make sure we advance the pixel index each time. Our geometric distribution
gives us the time between
        // successive events, which can be 0 for very high probabilities.
        pixel_idx += 1;
    }
}

/// Emulate VHS tracking error/noise.
/// TODO: this is inaccurate and doesn't let you control the position of the noise.
/// I need to revamp this at some point.
fn tracking_noise(
    yiq: &mut YiqView,
    info: &CommonInfo,
```

```
    num_rows: usize,
    wave_intensity: f32,
    snow_intensity: f32,
    snow_anisotropy: f32,
    noise_intensity: f32,
) {
    let width = yiq.dimensions.0;
    let height = yiq.num_rows();

    let mut seeder = Seeder::new(info.seed)
        .mix(noise_seeds::TRACKING_NOISE)
        .mix(info.frame_num);
    let noise_seed = seeder.clone().mix(0).finalize::<i32>();
    let offset = seeder.clone().mix(1).finalize::<f32>() * yiq.num_rows() as f32;
    seeder = seeder.mix(2);
    let shift_noise = NoiseBuilder::gradient_1d_offset(offset, num_rows)
        .with_seed(noise_seed)
        .with_freq(0.5)
        .generate()
        .0;

    // Handle cases where the number of affected rows exceeds the number of actual
rows in the image
    let start_row = height.max(num_rows) - num_rows;
    let cut_off_rows = if num_rows > height {
        num_rows - height
    } else {
        0
    };
    let affected_rows = &mut yiq.y[start_row * width..];
    let scratch_rows = &mut yiq.scratch[start_row * width..];

    affected_rows
        .par_chunks_mut(width)
        .zip(scratch_rows.par_chunks_mut(width))
        .enumerate()
        .for_each(|(index, (row, scratch))| {
            let index = index + cut_off_rows;
            // This iterates from the top down. Increase the intensity as we approach
the bottom of the picture.
            let intensity_scale = index as f32 / num_rows as f32;
            shift_row(
                row,
                shift_noise[index] * intensity_scale * wave_intensity * 0.25 *
info.bandwidth_scale,
                BoundaryHandling::Constant(0.0),
            );

            video_noise_line(
```

```
                row,
                scratch,
                &seeder,
                index,
                0.25 / info.bandwidth_scale,
                intensity_scale.powi(2) * noise_intensity * 4.0,
                1,
            );

            row_speckles(
                row,
                &mut
Xoshiro256PlusPlus::seed_from_u64(seeder.clone().mix(index).finalize()),
                snow_intensity * intensity_scale.powi(2),
                snow_anisotropy,
                info.bandwidth_scale,
            );
        });
}

/// Add random bits of "snow" to an NTSC-encoded signal.
fn snow(yiq: &mut YiqView, info: &CommonInfo, intensity: f32, anisotropy: f32) {
    let seeder = Seeder::new(info.seed)
        .mix(noise_seeds::SNOW)
        .mix(info.frame_num);

    yiq.y
        .par_chunks_mut(yiq.dimensions.0)
        .enumerate()
        .for_each(|(index, row)| {
            let line_seed = seeder.clone().mix(index);

            row_speckles(
                row,
                &mut Xoshiro256PlusPlus::seed_from_u64(line_seed.finalize()),
                intensity,
                anisotropy,
                info.bandwidth_scale,
            );
        });
}

/// Offset the chrominance (I and Q) planes horizontally and/or vertically.
/// Note how the horizontal shift is a float (the signal is continuous), but the
vertical shift is an int (each scanline
/// is discrete).
fn chroma_delay(yiq: &mut YiqView, info: &CommonInfo, offset: (f32, isize)) {
    let horiz_shift = offset.0 * info.bandwidth_scale;
    let copy_or_shift = |src: &mut [f32], dst: &mut [f32]| {
```

```
        if offset.0.abs() == 0.0 {
            dst.copy_from_slice(src);
        } else {
            shift_row_to(src, dst, horiz_shift, BoundaryHandling::Constant(0.0));
        }
    };

    let width = yiq.dimensions.0;
    let height = yiq.num_rows();

    match offset.1.cmp(&0) {
        std::cmp::Ordering::Less => {
            let offset = (-offset.1) as usize;
            // Starting from the top, copy (or write a horizontally-shifted copy of)
each row upwards.
            for dst_row_idx in 0..height {
                let (i_dst_part, i_src_part) = yiq.i.split_at_mut((dst_row_idx + 1) *
width);
                let (q_dst_part, q_src_part) = yiq.q.split_at_mut((dst_row_idx + 1) *
width);

                let dst_row_range = dst_row_idx * width..(dst_row_idx + 1) * width;
                let dst_i = &mut i_dst_part[dst_row_range.clone()];
                let dst_q = &mut q_dst_part[dst_row_range.clone()];

                if dst_row_idx >= height.max(offset) - offset {
                    dst_i.fill(0.0);
                    dst_q.fill(0.0);
                } else {
                    let src_row_range = (offset - 1) * width..offset * width;
                    let src_i = &mut i_src_part[src_row_range.clone()];
                    let src_q = &mut q_src_part[src_row_range.clone()];

                    copy_or_shift(src_i, dst_i);
                    copy_or_shift(src_q, dst_q);
                }
            }
        }
        std::cmp::Ordering::Equal => {
            // Only a horizontal shift is necessary. We can do this in-place easily.
            yiq.i
                .par_chunks_mut(width)
                .zip(yiq.q.par_chunks_mut(width))
                .for_each(|(i, q)| {
                    shift_row(i, horiz_shift, BoundaryHandling::Constant(0.0));
                    shift_row(q, horiz_shift, BoundaryHandling::Constant(0.0));
                });
        }
        std::cmp::Ordering::Greater => {
```

```
            // Some finagling is required to shift vertically. This branch shifts the
chroma planes downwards.
            let offset = offset.1 as usize;
            // Starting from the bottom, copy (or write a horizontally-shifted copy
of) each row downwards.
            for dst_row_idx in (0..height).rev() {
                let (i_src_part, i_dst_part) = yiq.i.split_at_mut(dst_row_idx *
width);
                let (q_src_part, q_dst_part) = yiq.q.split_at_mut(dst_row_idx *
width);

                let dst_row_range = 0..width;
                let dst_i = &mut i_dst_part[dst_row_range.clone()];
                let dst_q = &mut q_dst_part[dst_row_range.clone()];

                if dst_row_idx < offset {
                    dst_i.fill(0.0);
                    dst_q.fill(0.0);
                } else {
                    let src_row_idx = dst_row_idx - offset;
                    let src_row_range = src_row_idx * width..(src_row_idx + 1) *
width;
                    let src_i = &mut i_src_part[src_row_range.clone()];
                    let src_q = &mut q_src_part[src_row_range.clone()];

                    copy_or_shift(src_i, dst_i);
                    copy_or_shift(src_q, dst_q);
                }
            }
        }
    };
}

/// Emulate VHS waviness / horizontal shift noise.
fn vhs_edge_wave(yiq: &mut YiqView, info: &CommonInfo, settings: &VHSEdgeWaveSettings)
{
    let width = yiq.dimensions.0;
    let height = yiq.num_rows();

    let seeder = Seeder::new(info.seed).mix(noise_seeds::EDGE_WAVE);
    let noise_seed: i32 = seeder.clone().mix(0).finalize();
    let offset = seeder.mix(1).finalize::<f32>() * yiq.num_rows() as f32;
    let noise = &mut yiq.scratch[..height];
    NoiseBuilder::fbm_2d_offset(offset, height, info.frame_num as f32 *
settings.speed, 1)
        .with_seed(noise_seed)
        .with_freq(settings.frequency)
        .with_octaves(settings.detail.clamp(1, 5) as u8)
```

```
        // Yes, they got the lacunarity backwards by making it apply to frequency
instead of scale.
        // 2.0 *halves* the scale each time because it doubles the frequency.
        .with_lacunarity(2.0)
        .with_gain(std::f32::consts::FRAC_1_SQRT_2)
        .generate_into(noise);

    for plane in [&mut yiq.y, &mut yiq.i, &mut yiq.q] {
        plane
            .par_chunks_mut(width)
            .enumerate()
            .for_each(|(index, row)| {
                let shift =
                    (noise[index] / 0.022) * settings.intensity * 0.5 *
info.bandwidth_scale;
                shift_row(row, shift, BoundaryHandling::Extend);
            })
    }
}

/// Drop out the chrominance signal from random lines.
fn chroma_loss(yiq: &mut YiqView, info: &CommonInfo, intensity: f32) {
    let width = yiq.dimensions.0;
    let height = yiq.num_rows();

    let seed = Seeder::new(info.seed)
        .mix(noise_seeds::CHROMA_LOSS)
        .mix(info.frame_num)
        .finalize();

    let mut rng = Xoshiro256PlusPlus::seed_from_u64(seed);
    // We blank out each row with a probability of `intensity` (0 to 1). Instead of
going over each row and checking
    // whether to blank out the chroma, use a geometric distribution to simulate that
process and tell us which rows
    // to blank.
    let dist = Geometric::new(intensity as f64);

    let mut row_idx = 0usize;
    loop {
        row_idx += rng.sample(&dist);
        if row_idx >= height {
            break;
        }

        let row_range = row_idx * width..(row_idx + 1) * width;
        yiq.i[row_range.clone()].fill(0.0);
        yiq.q[row_range.clone()].fill(0.0);
        row_idx += 1;
```

```
    }
}

/// Vertically blend each chroma scanline with the one above it, as VHS does.
fn chroma_vert_blend(yiq: &mut YiqView) {
    let width = yiq.dimensions.0;
    let mut delay_i = vec![0f32; width];
    let mut delay_q = vec![0f32; width];

    yiq.i
        .chunks_mut(width)
        .zip(yiq.q.chunks_mut(width))
        .for_each(|(i_row, q_row)| {
            // TODO: check if this is faster than interleaved (I think the cache
locality is better this way)
            i_row.iter_mut().enumerate().for_each(|(index, i)| {
                let c_i = *i;
                *i = (delay_i[index] + c_i) * 0.5;
                delay_i[index] = c_i;
            });
            q_row.iter_mut().enumerate().for_each(|(index, q)| {
                let c_q = *q;
                *q = (delay_q[index] + c_q) * 0.5;
                delay_q[index] = c_q;
            });
        });
}

impl NtscEffect {
    fn apply_effect_to_yiq_field(&self, yiq: &mut YiqView, frame_num: usize) {
        let width = yiq.dimensions.0;

        let seed = self.random_seed as u32 as u64;

        let info = CommonInfo {
            seed,
            frame_num,
            bandwidth_scale: self.bandwidth_scale,
        };

        luma_filter(yiq, self.input_luma_filter);

        match self.chroma_lowpass_in {
            ChromaLowpass::Full => {
                composite_chroma_lowpass(yiq, &info, self.filter_type);
            }
            ChromaLowpass::Light => {
                composite_chroma_lowpass_lite(yiq, &info, self.filter_type);
            }
```

```
    ChromaLowpass::None => {}
};

chroma_into_luma(
    yiq,
    &info,
    self.video_scanline_phase_shift,
    self.video_scanline_phase_shift_offset,
);

if self.composite_preemphasis != 0.0 {
    let preemphasis_filter = make_lowpass(
        (315000000.0 / 88.0 / 2.0) * self.bandwidth_scale,
        NTSC_RATE * self.bandwidth_scale,
    );
    filter_plane(
        yiq.y,
        width,
        &preemphasis_filter,
        InitialCondition::Zero,
        -self.composite_preemphasis,
        0,
    );
}

if let Some(noise) = &self.composite_noise {
    composite_noise(yiq, &info, noise);
}

if self.snow_intensity > 0.0 && self.bandwidth_scale > 0.0 {
    snow(yiq, &info, self.snow_intensity * 0.01, self.snow_anisotropy);
}

if let Some(HeadSwitchingSettings {
    height,
    offset,
    horiz_shift,
    mid_line,
}) = &self.head_switching
{
    head_switching(
        yiq,
        &info,
        *height as usize,
        *offset as usize,
        *horiz_shift,
        mid_line.as_ref(),
    );
}
```

```
if let Some(TrackingNoiseSettings {
    height,
    wave_intensity,
    snow_intensity,
    snow_anisotropy,
    noise_intensity,
}) = self.tracking_noise
{
    tracking_noise(
        yiq,
        &info,
        height as usize,
        wave_intensity,
        snow_intensity,
        snow_anisotropy,
        noise_intensity,
    );
}

luma_into_chroma(
    yiq,
    &info,
    self.chroma_demodulation,
    self.video_scanline_phase_shift,
    self.video_scanline_phase_shift_offset,
);

if self.luma_smear > 0.0 {
    luma_smear(yiq, &info, self.luma_smear);
}

if let Some(ringing) = &self.ringing {
    let notch_filter = make_notch_filter(
        (ringing.frequency / self.bandwidth_scale).clamp(0.0, 1.0),
        ringing.power,
    );
    filter_plane(
        yiq.y,
        width,
        &notch_filter,
        InitialCondition::FirstSample,
        ringing.intensity,
        1,
    );
}

if let Some(luma_noise_settings) = &self.luma_noise {
    plane_noise(
```

```
            yiq.y,
            yiq.scratch,
            yiq.dimensions.0,
            &info,
            luma_noise_settings,
            noise_seeds::VIDEO_LUMA,
        );
    }

    if let Some(chroma_noise_settings) = &self.chroma_noise {
        plane_noise(
            yiq.i,
            yiq.scratch,
            yiq.dimensions.0,
            &info,
            chroma_noise_settings,
            noise_seeds::VIDEO_CHROMA_I,
        );
        plane_noise(
            yiq.q,
            yiq.scratch,
            yiq.dimensions.0,
            &info,
            chroma_noise_settings,
            noise_seeds::VIDEO_CHROMA_Q,
        );
    }

    if self.chroma_phase_error > 0.0 {
        chroma_phase_error(yiq, self.chroma_phase_error);
    }

    if self.chroma_phase_noise_intensity > 0.0 {
        chroma_phase_noise(yiq, &info, self.chroma_phase_noise_intensity);
    }

    if self.chroma_delay_horizontal != 0.0 || self.chroma_delay_vertical != 0 {
        chroma_delay(
            yiq,
            &info,
            (
                self.chroma_delay_horizontal,
                self.chroma_delay_vertical as isize,
            ),
        );
    }

    if let Some(vhs_settings) = &self.vhs_settings {
        if let Some(edge_wave) = &vhs_settings.edge_wave {
```

```
                if edge_wave.intensity > 0.0 {
                    vhs_edge_wave(yiq, &info, edge_wave);
                }
            }

            if let Some(VHSTapeParams {
                luma_cut,
                chroma_cut,
                chroma_delay,
            }) = vhs_settings.tape_speed.filter_params()
            {
                // TODO: add an option to control whether there should be a line on
the left from the filter starting
                // at 0. it's present in both the original C++ code and Python port
but probably not an actual VHS
                // TODO: use a better filter! this effect's output looks way more
smear-y than real VHS
                let luma_filter = make_lowpass_for_type(
                    luma_cut,
                    NTSC_RATE * self.bandwidth_scale,
                    self.filter_type,
                );
                let chroma_filter = make_lowpass_for_type(
                    chroma_cut,
                    NTSC_RATE * self.bandwidth_scale,
                    self.filter_type,
                );
                filter_plane(yiq.y, width, &luma_filter, InitialCondition::Zero, 1.0,
0);
                filter_plane(
                    yiq.i,
                    width,
                    &chroma_filter,
                    InitialCondition::Zero,
                    1.0,
                    chroma_delay,
                );
                filter_plane(
                    yiq.q,
                    width,
                    &chroma_filter,
                    InitialCondition::Zero,
                    1.0,
                    chroma_delay,
                );
                let luma_filter_single = make_lowpass(luma_cut, NTSC_RATE *
self.bandwidth_scale);
                filter_plane(
                    yiq.y,
```

```
                width,
                &luma_filter_single,
                InitialCondition::Zero,
                -1.6,
                0,
            );
        }

        if vhs_settings.chroma_loss > 0.0 {
            chroma_loss(yiq, &info, vhs_settings.chroma_loss);
        }

        if let Some(sharpen) = &vhs_settings.sharpen {
            if let Some(VHSTapeParams { luma_cut, .. }) =
                vhs_settings.tape_speed.filter_params()
            {
                let frequency_extra_multiplier = match self.filter_type {
                    FilterType::ConstantK => 4.0,
                    FilterType::Butterworth => 1.0,
                };
                let luma_sharpen_filter = make_lowpass_for_type(
                    luma_cut * frequency_extra_multiplier * sharpen.frequency,
                    NTSC_RATE * self.bandwidth_scale,
                    self.filter_type,
                );
                );*/
                filter_plane(
                    yiq.y,
                    width,
                    &luma_sharpen_filter,
                    InitialCondition::Zero,
                    -sharpen.intensity * 2.0 * sharpen.frequency,
                    0,

    if self.chroma_vert_blend {
        chroma_vert_blend(yiq);
    }

    match self.chroma_lowpass_out {
        ChromaLowpass::Full => {
            composite_chroma_lowpass(yiq, &info, self.filter_type);
        }
        ChromaLowpass::Light => {
            composite_chroma_lowpass_lite(yiq, &info, self.filter_type);
        }
        ChromaLowpass::None => {}
    };
}
```

```
    pub fn apply_effect_to_yiq(&self, yiq: &mut YiqView, frame_num: usize) {
        let pool = rayon::ThreadPoolBuilder::new()
            .stack_size(2 * 1024 * 1024)
            .build()
            .unwrap();
        pool.scope(|_| match yiq.field {
            YiqField::Upper | YiqField::Lower | YiqField::Both => {
                self.apply_effect_to_yiq_field(yiq, frame_num);
            }
            YiqField::InterleavedUpper | YiqField::InterleavedLower => {
                // "Interleaved" basically means we apply the effect to one set of
                let (mut yiq_upper, mut yiq_lower, frame_num_upper, frame_num_lower) =
                    match yiq.field {
                        YiqField::InterleavedUpper => {
                            let num_upper_rows =
YiqField::Upper.num_actual_image_rows(yiq.dimensions.1);
                            let (upper, lower) = yiq.split_at_row(num_upper_rows);
                            (upper, lower, frame_num * 2, frame_num * 2 + 1)
                        }
                        YiqField::InterleavedLower => {
                            let num_lower_rows =
YiqField::Lower.num_actual_image_rows(yiq.dimensions.1);
                            let (lower, upper) = yiq.split_at_row(num_lower_rows);
                            (upper, lower, frame_num * 2 + 1, frame_num * 2)
                        }
                        _ => unreachable!(),
                    };

                if let Some(yiq_upper) = yiq_upper.as_mut() {
                    yiq_upper.field = YiqField::Upper;
                    self.apply_effect_to_yiq_field(yiq_upper, frame_num_upper);
                }

                if let Some(yiq_lower) = yiq_lower.as_mut() {
                    yiq_lower.field = YiqField::Lower;
                    self.apply_effect_to_yiq_field(yiq_lower, frame_num_lower);
                }
            }
        })
    }

    pub fn apply_effect_to_buffer<S: PixelFormat>(
        &self,
        dimensions: (usize, usize),
        input_frame: &mut [S::DataFormat],
        frame_num: usize,
    ) {
        let field = self.use_field.to_yiq_field(frame_num);
        let row_bytes = dimensions.0 * S::pixel_bytes();
```

```
        let mut yiq = YiqOwned::from_strided_buffer::<S>(
            input_frame,
            row_bytes,
            dimensions.0,
            dimensions.1,
            field,
        );
        let mut view = YiqView::from(&mut yiq);
        self.apply_effect_to_yiq(&mut view, frame_num);
        view.write_to_strided_buffer::<S, _>(
            input_frame,
            BlitInfo::from_full_frame(dimensions.0, dimensions.1, row_bytes),
            crate::yiq_fielding::DeinterlaceMode::Bob,
            identity,
        );
    }
}

#[derive(Clone, Copy, Debug, PartialEq)]
pub enum BoundaryHandling {
    /// Repeat the boundary pixel over and over.
    Extend,
    /// Use a specific constant for the boundary.
    Constant(f32),
}

fn shift_row_initial_conditions(
    row: &[f32],
    shift: f32,
    boundary_handling: BoundaryHandling,
) -> (isize, f32, f32, f32) {
    // Floor the shift (conversions round towards zero)
    let shift_int = shift as isize - if shift < 0.0 { 1 } else { 0 };

    let width = row.len();
    let boundary_value = match boundary_handling {
        BoundaryHandling::Extend => {
            if shift_int >= 0 {
                row[0]
            } else {
                row[width - 1]
            }
        }
        BoundaryHandling::Constant(value) => value,
    };
    let shift_frac = if shift < 0.0 {
        1.0 - shift.fract().abs()
    } else {
        shift.fract()
```

```
    };

    let prev = if shift_int >= width as isize || shift_int < -(width as isize) {
        boundary_value
    } else if shift_int >= 0 {
        row[(width - shift_int as usize) - 1]
    } else {
        row[-shift_int as usize - 1]
    };

    (shift_int, shift_frac, boundary_value, prev)
}

/// Shift a row by a non-integer amount using linear interpolation.
pub fn shift_row(row: &mut [f32], shift: f32, boundary_handling: BoundaryHandling) {
    let width = row.len();
    let (shift_int, shift_frac, boundary_value, mut prev) =
        shift_row_initial_conditions(row, shift, boundary_handling);

    if shift_int >= 0 {
        // Shift forwards; iterate the list backwards
        let offset = shift_int as usize + 1;
        for i in (0..width).rev() {
            let old_value = if i >= offset {
                row[i - offset]
            } else {
                boundary_value
            };
            row[i] = (prev * (1.0 - shift_frac)) + (old_value * shift_frac);
            prev = old_value;
        }
    } else {
        // Shift backwards; iterate the list forwards
        let offset = (-shift_int) as usize;
        for i in 0..width {
            let old_value = if i + offset < width {
                row[i + offset]
            } else {
                boundary_value
            };
            row[i] = (prev * shift_frac) + (old_value * (1.0 - shift_frac));
            prev = old_value;
        }
    }
}

/// Shift a row by a non-integer amount using linear interpolation.
pub fn shift_row_to(src: &[f32], dst: &mut [f32], shift: f32, boundary_handling:
BoundaryHandling) {
```

```
    let width = src.len();
    let (shift_int, shift_frac, boundary_value, mut prev) =
        shift_row_initial_conditions(src, shift, boundary_handling);
    if shift_int >= 0 {
        // Shift forwards; iterate the list backwards
        let offset = shift_int as usize + 1;
        for i in (0..width).rev() {
            let old_value = if i >= offset {
                src[i - offset]
            } else {
                boundary_value
            };
            dst[i] = (prev * (1.0 - shift_frac)) + (old_value * shift_frac);
            prev = old_value;
        }
    } else {
        // Shift backwards; iterate the list forwards
        let offset = (-shift_int) as usize;
        for i in 0..width {
            let old_value = if i + offset < width {
                src[i + offset]
            } else {
                boundary_value
            };
            dst[i] = (prev * shift_frac) + (old_value * (1.0 - shift_frac));
            prev = old_value;
        }
    }
}

#[cfg(test)]
mod tests {
    use super::*;

    const TEST_DATA: &[f32] = &[1.0, 2.5, -0.7, 0.0, 0.0, 2.2, 0.3];

    fn assert_almost_eq(a: &[f32], b: &[f32]) {
        let all_almost_equal = a.iter().zip(b).all(|(a, b)| (a - b).abs() <= 0.01);
        assert!(all_almost_equal, "{a:?} is almost equal to {b:?}");
    }

    fn test_case(shift: f32, boundary_handling: BoundaryHandling, expected: &[f32]) {
        let mut shifted = TEST_DATA.to_vec();
        shift_row(&mut shifted, shift, boundary_handling);
        assert_almost_eq(&shifted, expected);

        let mut shifted = TEST_DATA.to_vec();
        shift_row_to(TEST_DATA, &mut shifted, shift, boundary_handling);
        assert_almost_eq(&shifted, expected);
```

```
}

#[test]
fn test_shift_pos_1() {
    test_case(
        0.5,
        BoundaryHandling::Extend,
        &[1.0, 1.75, 0.9, -0.35, 0.0, 1.1, 1.25],
    );
}

#[test]
fn test_shift_pos_2() {
    test_case(
        5.5,
        BoundaryHandling::Extend,
        &[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.75],
    );
}

#[test]
fn test_shift_neg_1() {
    test_case(
        -0.01,
        BoundaryHandling::Extend,
        &[1.015, 2.468, -0.693, 0.0, 0.02199998, 2.181, 0.3],
    );
}

#[test]
fn test_shift_neg_2() {
    test_case(
        -1.01,
        BoundaryHandling::Extend,
        &[2.468, -0.693, 0.0, 0.02199998, 2.181, 0.3, 0.3],
    );
}

#[test]
fn test_shift_neg_full() {
    test_case(
        -6.0,
        BoundaryHandling::Extend,
        &[0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3],
    );
}

#[test]
fn test_shift_neg_full_ext() {
```

```
        test_case(
            -7.0,
            BoundaryHandling::Extend,
            &[0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3],
        );
    }

    #[test]
    fn test_shift_pos_full() {
        test_case(
            6.0,
            BoundaryHandling::Extend,
            &[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0],
        );
    }

    #[test]
    fn test_shift_pos_full_ext() {
        test_case(
            7.0,
            BoundaryHandling::Extend,
            &[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0],
        );
    }
}
```

ДОДАТОК Г.

Додаток Г – Ілюстративна частина

РОЗРОБКА МЕТОДУ ТА ПРОГРАМНОГО ЗАСОБУ ДЛЯ СТВОРЕННЯ ВІНТАЖНИХ ЕФЕКТІВ VHS-ПЛІВКИ У ЦИФРОВИХ ВІДЕО

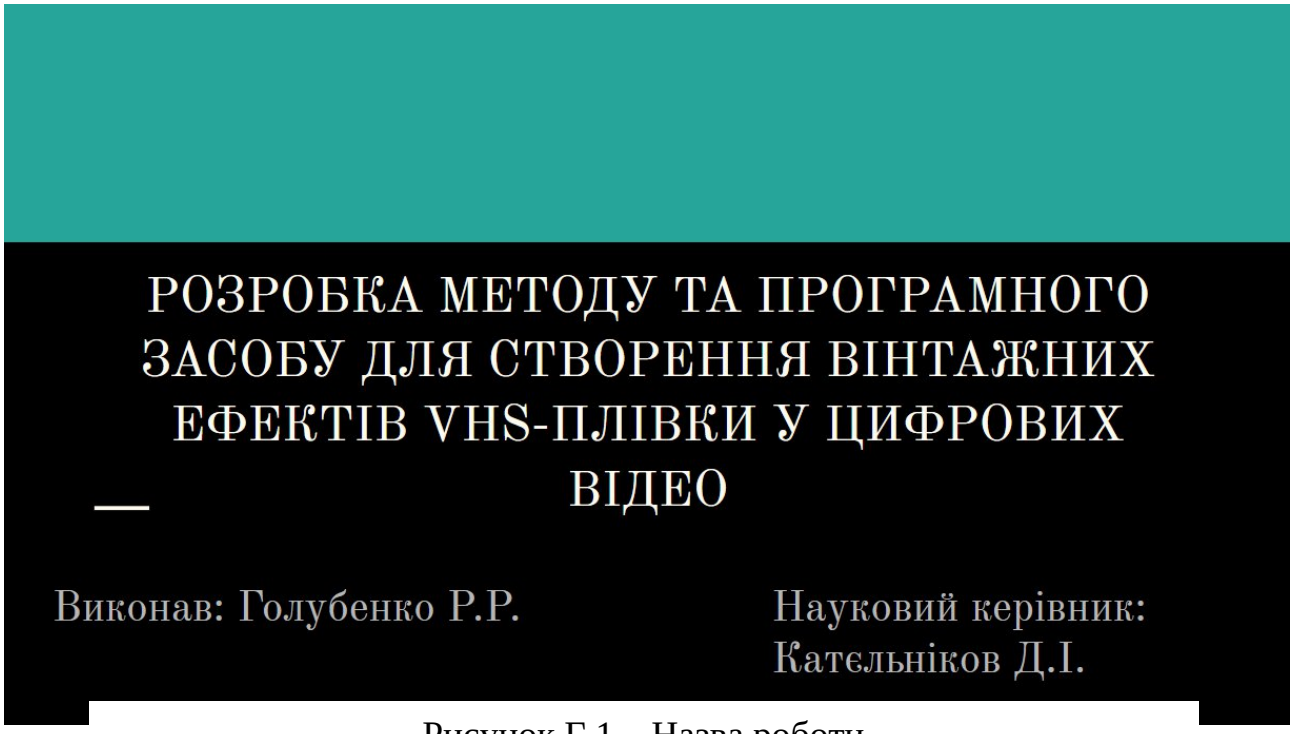

Рисунок Г.1 – Назва роботи

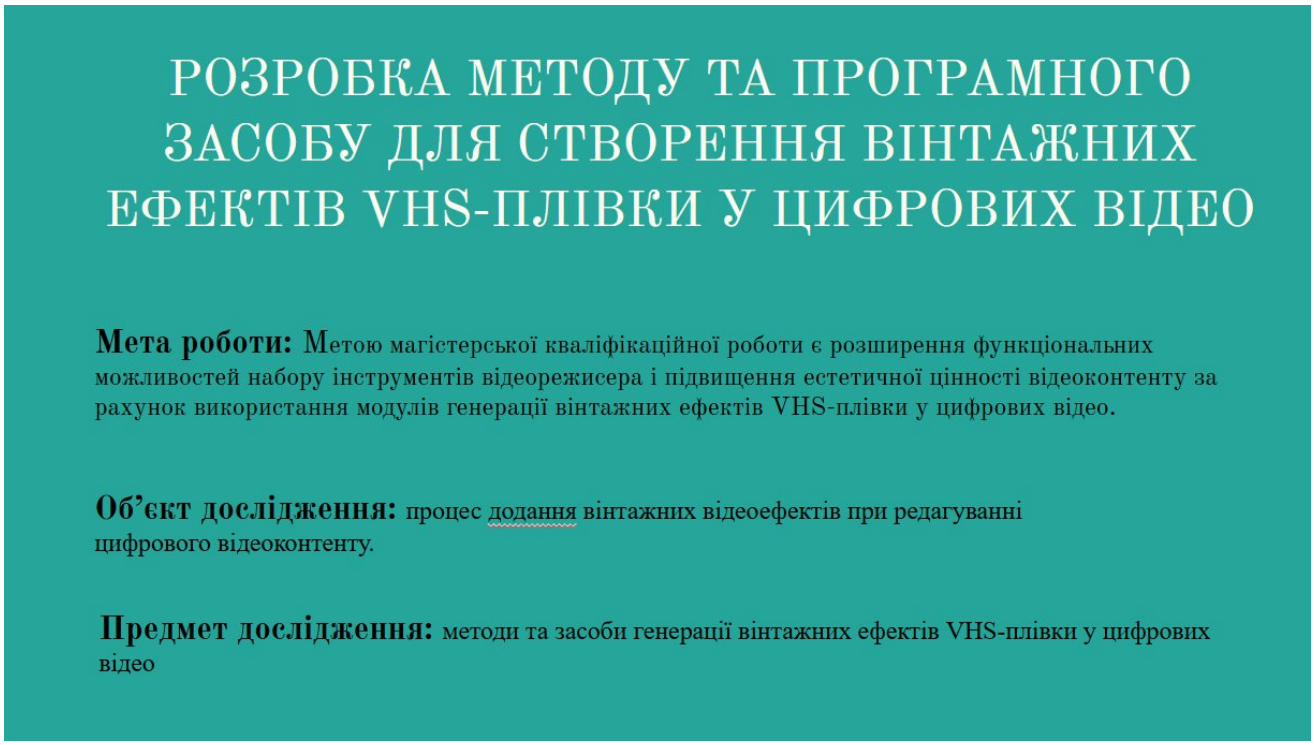

Рисунок Г.2 – Мета та дослідження

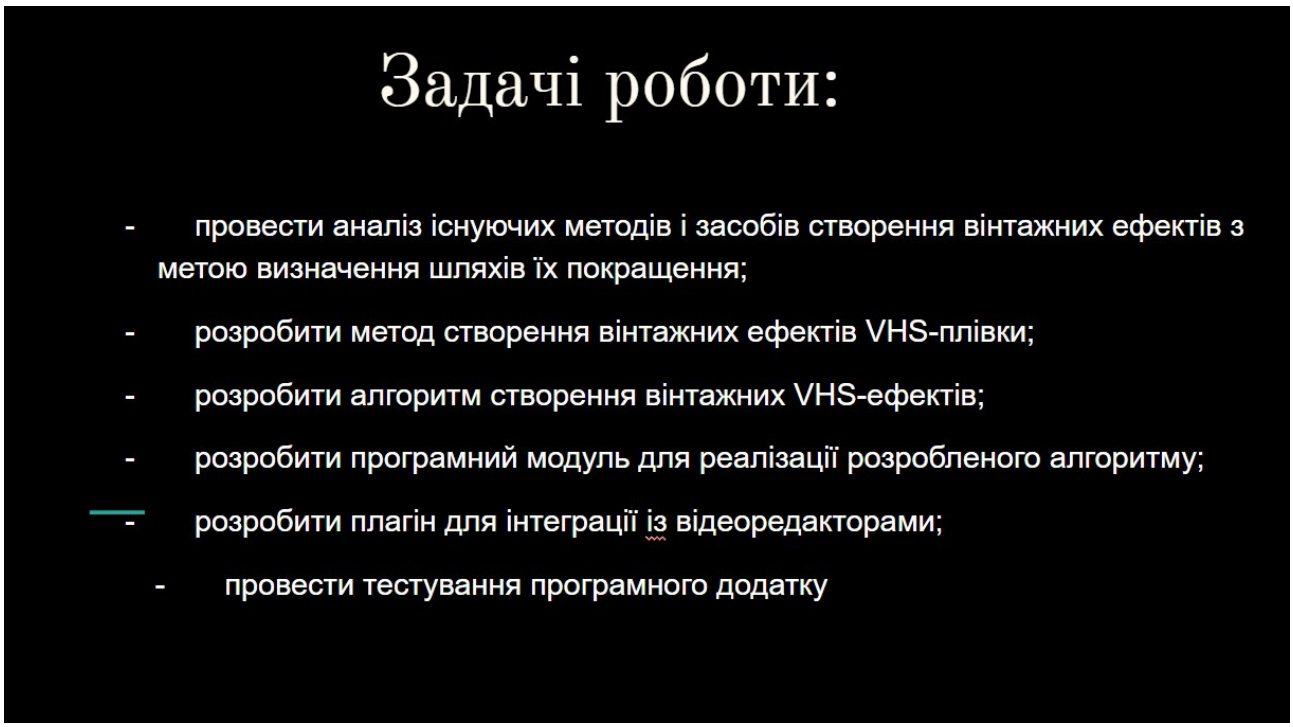

Рисунок Г.3 – Задачі Роботи

Рисунок Г.4 – Актуальність розробки

Рисунок Г.5 – Аналоги

Рисунок Г.6 - Розробка методів і засобів реалізації симуляції аналогового шуму та артефактів

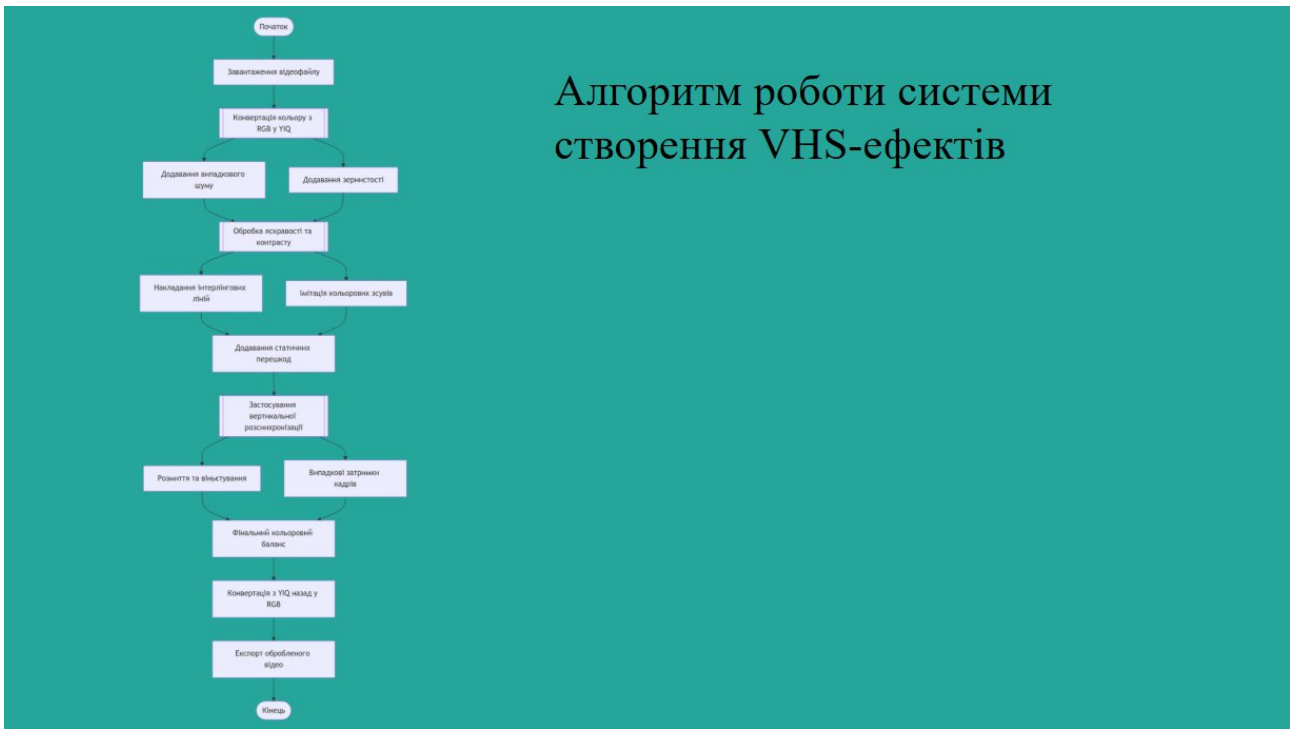

Рисунок Г.7 - Алгоритм роботи системи створення VHS-ефектів

Рисунок Г.8 - Блок-схема ініціалізації плагіну

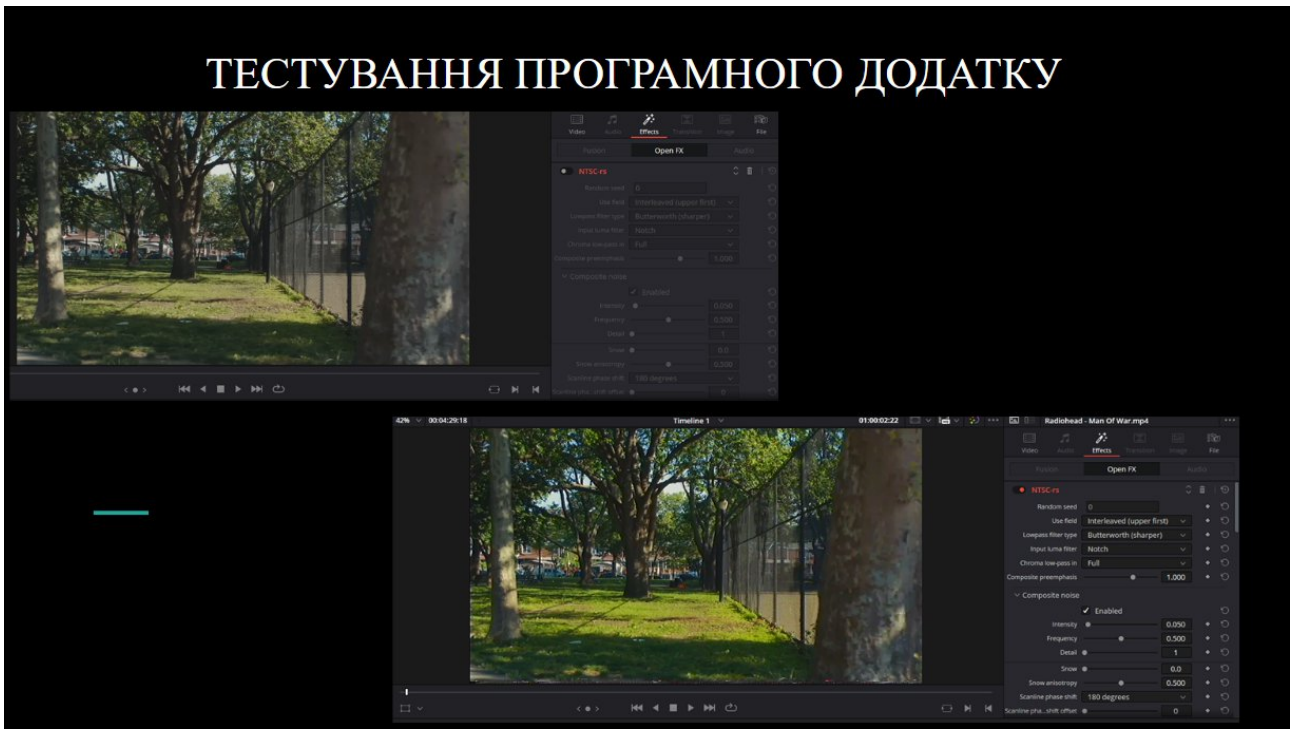

Рисунок Г.9 – Тестування програмного додатку

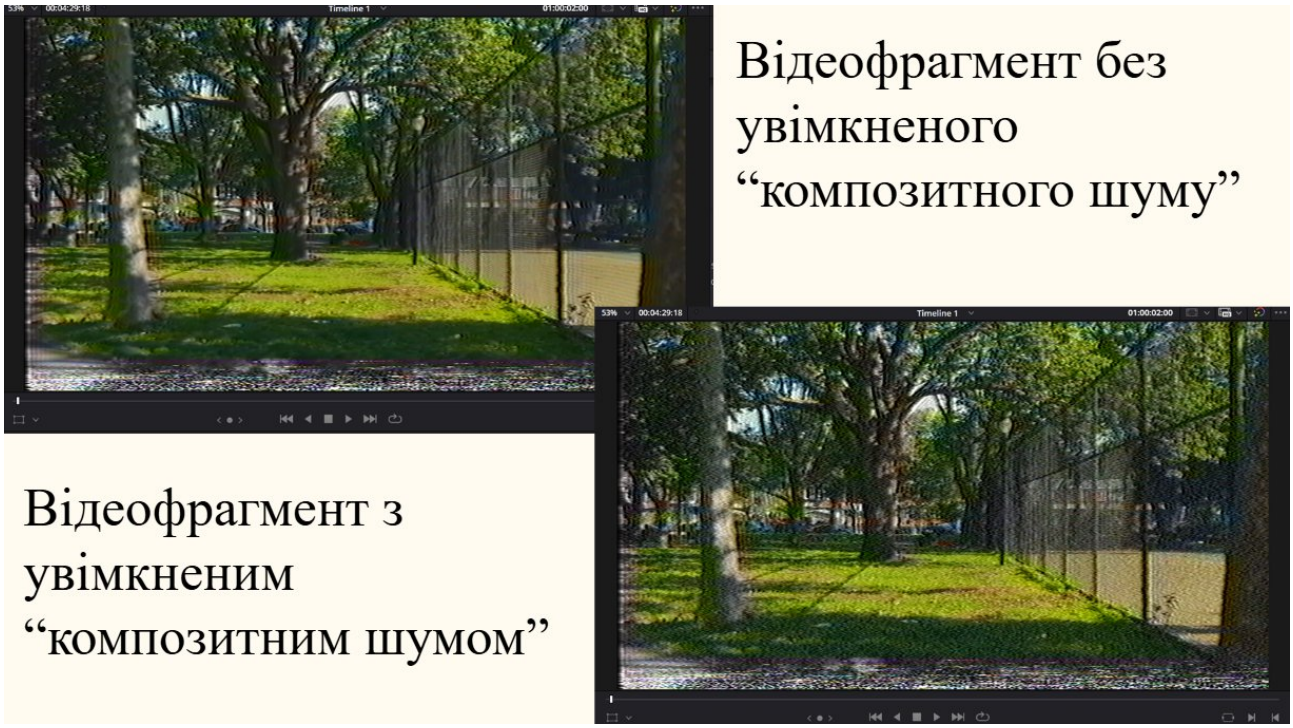

Рисунок Г.10 – Порівняння ефектів

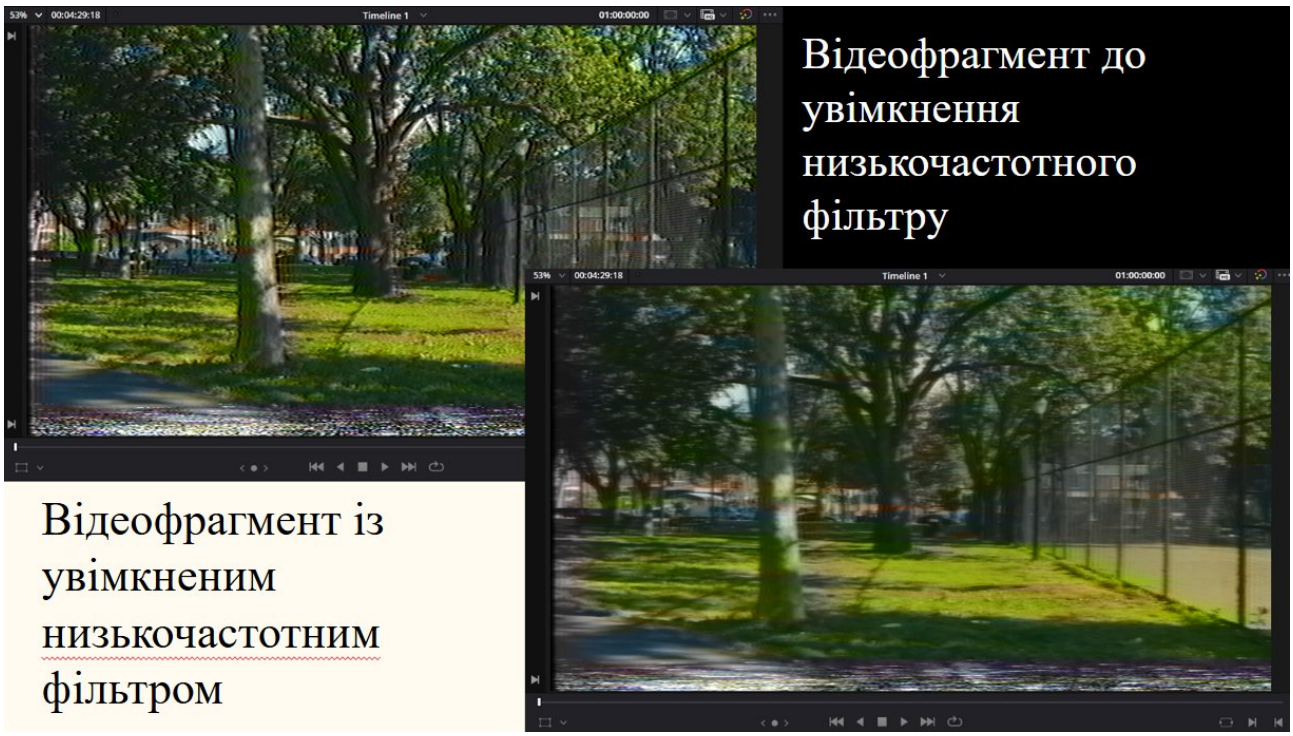

Рисунок Г.11 – Порівняння низкьочастоних фільтрів

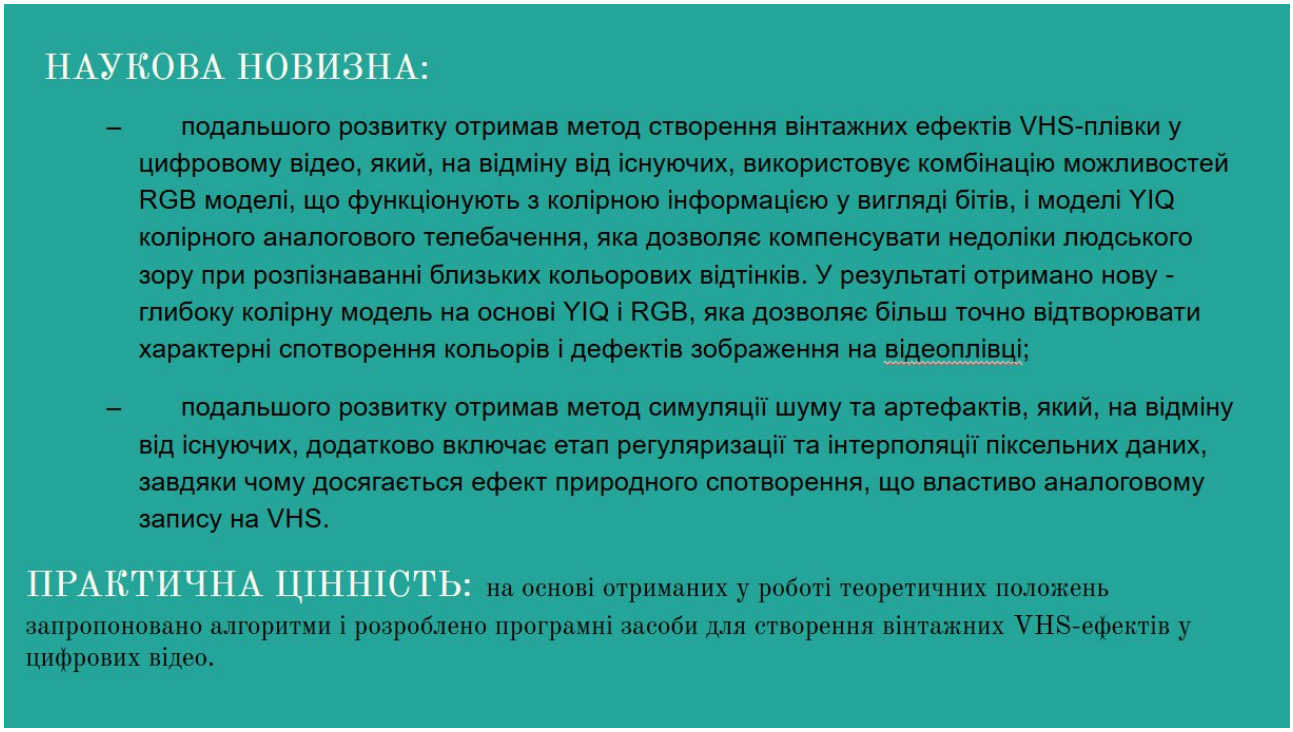

Рисунок Г.12 – Наукова новизна та практична цінність

Висновки

Під час виконання кваліфікаційної роботи було:

- розроблено методи та засоби для створення цифрових ефектів, що імітують візуальні властивості відеокасет формату VHS.
- реалізацію виконано мовою Rust із використанням бібліотеки OpenFX для інтеграції плагіна з популярними відеоредакторами.
- створено алгоритми для моделювання колірних перетворень, шумів, стрічкових спотворень та інших характерних артефактів VHS.
- розроблено плагін який забезпечує імітацію ефектів хроматичної аберації, скануючих ліній, зернистості та інших дефектів.
- проведено тестування програмного продукту, розроблено інструкцію користувача.

Рисунок Г.13 – Висновки

Публікації

Результати дослідження опубліковано в наступній науковій роботі:

Р.Р. Голубенко, Д.І. Катєльніков. Розробка Методу І Програмного Засобу для Створення Вінтажних Ефектів VHS-плівки У Цифрових Відео. Матеріали з конференції комп'ютерних ігор 26-27 вересня 2024 р. – Одеса. 2024. С. 248-249.

Рисунок Г.14 - Публікації