**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ТЕЛЕКОМУНІКАЦІЙ
Кафедра Мобільних та відеоінформаційних технологій

# КВАЛІФІКАЦІЙНА РОБОТА

## на тему: «Методи стиснення потокових відеоданих на основі сучасних відеокодеків»

на здобуття освітнього ступеня магістра
зі спеціальності 172 Телекомунікації та радіотехніка
освітньо-професійної програми Апаратура радіозв'язку, радіомовлення і телебачення

*Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело*

(підпис) Юрій ЛИЦЕНКО (Ім'я, ПРІЗВИЩЕ)

Виконав: здобувач вищої освіти
групи АРДМ-62
Юрій ЛИЦЕНКО

Керівник: Віталій СТРЕЛЬНІКОВ

Рецензент: Ганна Гринкевич

Київ – 2023

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
**Навчально-науковий інститут телекомунікацій**

Кафедра Мобільних та відеоінформаційних технологій
Ступінь вищої освіти Магістр
Спеціальність 172 Телекомунікації та радіотехніка
Освітньо-професійна програма Апаратура радіозв'язку, радіомовлення і телебачення

**ЗАТВЕРДЖУЮ**
Завідувачка кафедрою МВТ
_____ Наталія РУДЕНКО
«  » _____ 2023 року

# ЗАВДАННЯ
# НА КВАЛІФІКАЦІЙНУ РОБОТУ

Лиценку Юрію Вадимовичу

1. Тема кваліфікаційної роботи: «Методи стиснення потокових відеоданих на основі сучасних відеокодеків»,
керівник кваліфікаційної роботи Віталій СТРЕЛЬНІКОВ, PhD,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від 19.10.2023 р. №145.

2. Строк подання кваліфікаційної роботи 25.12.2023 р.

3. Вихідні дані до кваліфікаційної роботи:
   1. Кодування відеоданих.
   2. 4k-відео.
   3. SDK VIDEO CODEC.
   4. Науково-технічна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   1. Стан проблеми кодування відео.
   2. Архітектура та використання відеокарт.
   3. Аналіз та розробка SDK video codec.
   4. Реалізація програми розширення SDK video codec.
   5. Тестування розширення SDK video codec.

5. Перелік ілюстративного матеріалу: презентація.

6. Дата видачі завдання _____

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Підбір науково-технічної літератури | 19.10.23 | Викон. |
| 2. | Стан проблеми кодування відео | 31.10.23 | Викон. |
| 3. | Архітектура та використання відеокарт | 15.11.23 | Викон. |
| 4. | Аналіз та розробка SDK video codec | 29.11.23 | Викон. |
| 5. | Реалізація програми розширення SDK video codec. Тестування розширення SDK video codec | 07.12.23 | Викон. |
| 6. | Висновки, вступ, реферат | 11.12.23 | Викон. |
| 7. | Розробка презентації | 18.12.23 | Викон. |

**Здобувач вищої освіти** _____ Юрій ЛИЦЕНКО
*(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

**Керівник кваліфікаційної роботи** _____ Віталій СТРЕЛЬНІКОВ
*(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

# РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 80 стор., 26 рис., 14 табл., 28 дж.

*Мета роботи* – розробка власного програмного забезпечення кодування 4k-відео в реальному часі на основі SDK Video Codec.

*Об'єкт дослідження* - процес кодування відеоданих.

*Предмет дослідження* – методи кодування 4k-відео в реальному часі.

*Короткий зміст роботи*: Магістерська робота присвячена питанням кодування відео 4K у режимі реального часу. Містить основний опис найбільш використовуваних відеокодеків H.264 та H.265. Показано принципи роботи графічних карт та їх програмованих блоків. Проаналізовано рішення вільно доступного SDK Video Codec. Розроблено програму, яка може кодувати відео з декількох камер 4K в режимі реального часу.

Практична частина роботи реалізована на мові програмування C ++. Тут описані найважливіші частини реалізації програми. Описано розподіл процесів на кілька потоків за допомогою технології Visual Studio 2019.

Важлива частина опису реалізації присвячена процесу кодування. Додаток було успішно реалізовано, і, крім роботи, був реалізований графічний інтерфейс для запуску отриманих додатків та контролю навантаження кодування в системі.

Проведено тестування ефективності та якості роботи розробленого ПЗ кодування відео з декількох камер 4K в режимі реального часу.

*КЛЮЧОВІ СЛОВА:* КОДУВАННЯ ВІДЕО, КОДЕР, NVENC, H.264, HEVC, CUDA, DIRECTX, SDK VIDEO CODEC

## ABSTRACT

Text part of the master's qualification work: 80 pages, 26 pictures, 14 table, 28 sources.

*The purpose of the work* - development of own real-time 4k video encoding software based on SDK Video Codec.

*Object of research* – video data encoding process.

*Subject of research* – real-time 4k video encoding methods.

*Summary of the work:* The master's thesis is devoted to the issues of real-time 4K video coding. Contains a basic description of the most used H.264 and H.265 video codecs. The principles of operation of graphics cards and their programmable blocks are shown. The solution of the freely available SDK Video Codec is analyzed. A program has been developed that can encode video from multiple 4K cameras in real time.

The practical part of the work is implemented in the programming language C ++. The most important parts of the program implementation are described here. Describes how to split processes into multiple threads using Visual Studio 2019 technology.

An important part of the implementation description is devoted to the coding process. The application was successfully implemented and, in addition to the work, a GUI was implemented to run the resulting applications and control the coding load on the system.

Testing of the effectiveness and quality of the developed video encoding software from several 4K cameras in real time was conducted.



*KEYWORDS:* VIDEO ENCODING, ENCODER, NVENC, H.264, HEVC, CUDA, DIRECTX, SDK VIDEO CODEC

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
**Навчально-науковий інститут телекомунікацій**

# ПОДАННЯ
# ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
# ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
**на здобуття освітнього ступеня магістра**

Направляється здобувач Лиценко Ю.В. до захисту кваліфікаційної роботи
*(прізвище та ініціали)*

за спеціальністю 172 Телекомунікації та радіотехніка
освітньо-професійної програми Апаратура радіозв'язку, радіомовлення і телебачення

на тему: Методи стиснення потокових відеоданих на основі сучасних відеокодеків

Кваліфікаційна робота і рецензія додаються.

Директор ННІТ _____ Владислав КРАВЧЕНКО
*(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

**Висновок керівника кваліфікаційної роботи**

Здобувач Лиценко Юрій Вадимович показав гарну теоретичну та інженерну підготовку, уміння володіти технологіями нового покоління, вміло працювати з програмно-апаратними засобами мобільного зв'язку, довідковою і науково-технічною літературою. Працюючи над завданнями, які доручались керівником, проявив ініціативність, сумлінність та хист до інженерної роботи.

Все це дозволяє оцінити виконану кваліфікаційну роботу Лиценка Юрія Вадимовича на оцінку «відмінно» та присвоїти йому кваліфікацію «Магістр з телекомунікацій та радіотехніки».

Керівник кваліфікаційної роботи _____ Віталій СТРЕЛЬНІКОВ
*(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

" " _____ 2023 року

**Висновок кафедри про кваліфікаційну роботу**

Кваліфікаційна робота розглянута. Здобувач Лиценко Ю.В.
*(прізвище та ініціали)*

допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувачка кафедри
Мобільних та відеоінформаційних технологій _____ Наталія РУДЕНКО
*(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

" " _____ 2023 року

**ВІДГУК РЕЦЕНЗЕНТА**

на кваліфікаційну магістерську роботу

здобувача вищої освіти Лиценка Юрія Вадимовича

на тему: «Методи стиснення потокових відеоданих на основі сучасних відеокодеків»

**Актуальність:**

Кодування відео дозволяє ефективно зменшувати потік, необхідний для передачі відео по каналах радіомовлення, зменшувати простір, необхідний для зберігання даних на носії. Все це особливо актуально для кодування відео 4k-якості.

Тому, дослідження, проектування та тестування запропонованих рішень, є актуальними і своєчасними.

**Позитивні сторони:**

Робота дозволяє досить повно оцінити загальну характеристику, сутність та структуру поставленої проблеми. Дається чітке визначення мети, об'єкту та предмету дослідження.

Робота викладена науковою мовою, логічно й послідовно відбиває мету та поставлені в роботі завдання. Пояснювальна записка відповідає стандартам до її оформлення.

**Недоліки:**

1. В роботі доцільно було б проводити порівняльний аналіз з більшою кількістю існуючих технологій стиснення.
2. В роботі не показано чи використовує кодек H.264 для кодування аудіосигналів сторонні кодеки.

**Висновок:** кваліфікаційна робота заслуговує оцінку «відмінно», а здобувач Лиценко Юрій Вадимович заслуговує присвоєння кваліфікації: «Магістр з телекомунікацій та радіотехніки».

**Рецензент:** к.т.н., доцент _____ Г. Гринкевич

*(науковий ступінь, вчене звання)* *(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

**ЗМІСТ**

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ ..... 9
ВСТУП ..... 10
1 СТАН ПРОБЛЕМИ КОДУВАННЯ ВІДЕО ..... 12
1.1 Роль стандартів відеокодування в галузі телекомунікацій ..... 12
1.2 Стандарт кодування H.264 ..... 14
1.3 Стандарт кодування H.265 ..... 20
2 АРХІТЕКТУРА ТА ВИКОРИСТАННЯ ВІДЕОКАРТ ..... 27
2.1 Апаратне забезпечення GPU ..... 27
2.2 Графічний процесор GPGPU ..... 29
2.3 Обчислювальна платформа CUDA ..... 30
2.4 SDK для відеокодеків Nvidia ..... 31
3 АНАЛІЗ ТА РОЗРОБКА SDK VIDEO CODEC ..... 34
3.1 Загальний принцип Video Codec SDK ..... 34
3.2 AppEncCuda ..... 36
3.3 AppEncD3D11 ..... 39
3.4 AppEncGL ..... 41
3.5 Пропозиція щодо розширення SDK для Video Codec ..... 43
4 РЕАЛІЗАЦІЯ ПРОГРАМИ РОЗШИРЕННЯ SDK VIDEO CODEC ..... 44
4.1 Мова програмування та використовувані інструменти ..... 44
4.2 Поділ процесів ..... 44
4.3 Завантаження зображень із камер ..... 47
4.4 Процес кодування ..... 49
4.5 Впровадження графічного інтерфейсу ..... 52
5 ТЕСТУВАННЯ РОЗШИРЕННЯ SDK VIDEO CODEC ..... 57
5.1 Порівняння результатів аналізу окремих проектів ..... 57
5.2 Тести продуктивності ..... 61
5.3 Тести якості ..... 69
5.4 Обговорення результатів ..... 70
ВИСНОВКИ ..... 74
ПЕРЕЛІК ПОСИЛАНЬ ..... 76
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ ..... 79

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

VCEG – Video Coding Experts Group – Група експертів з кодування відео

MPEG – Moving Picture Experts Group – Група експертів з рухомих зображень

CTU - Coding Tree Unit – дерево кодування

JCT-VC - Joint Collaborative Team on Video Coding – Спільна група з кодування відео

CTU - Coding Tree Unit

CTB – Coding Tree Block – Блок кодування дерева

CB – Coding Blocks – Блок кодування

PU - Prediction Unit - одиниця передбачення

TU – Transform Unit - одиниця перетворення

TB – Transform Block – Блок транформації

CU – Coding Unit

PB – Prediction Block – Блок попередження

# ВСТУП

З розвитком нових технологій також зростає попит на кодування відео з максимально можливим стисненням і одночасно з найменшими втратами якості. Вони також підпорядковуються вимогам, встановленим для кодування та декодування фільмів із високою роздільною здатністю, які не можна передавати інакше, ніж у стислому стані. Це необхідно, щоб кодування і декодування не вимагало занадто багато обчислювальної потужності, і тому був створений чіп NVENC.

Ця робота зосереджена на технології чіпів NVENC та кодуванні за стандартами H.264 та H.265. У теоретичній частині описуються принципи кодування як наметового дардо, так і відмінності між ними. Далі обговорюються відеокарти та чіп NVENC та SDK Video Codec. Метою даної роботи є аналіз вже реалізованого Video Codec SDK, який використовує чіп NVENC, та реалізація власного простого кодера для кодування відеопослідовності високої чіткості в режимі реального часу.

У першому розділі увага приділяється кодуванню відео. Тут обговорюються відеокодеки та кольорові простори, які є важливим компонентом кодування відео. Згодом пояснюється, чому були створені стандарти, що стосуються кодування відео, і для чого вони використовуються. Основна частина – це детальний аналіз стандартів H.264 та H.265, де описується їх принцип та найважливіші методи.

У другому розділі йдеться про використання відеокарт. Представлена історія та архітектура відеокарт. У цьому розділі також зосереджено увагу на програмованих частинах графічної карти. Однією з них є CUDA, яка використовує відповідні передачі з Video Codec SDK для кодування фільмів. У наступному розділі описані принципи роботи Video Codec SDK від Nvidia. Також пояснюється роль чіпа NVENC та його попередника NVCUVID.

У третьому розділі йдеться про аналіз доступного SDK Video Codec. Він описує загальний принцип роботи кодування у вже реалізованому та

немодифікованому проекті. Крім того, вони аналізуються окремо для проектів з використанням кодування CUDA, DirectX і OpenGL. Важливі функції та методи кодера висвітлені та детально описані. В кінці глави - пропозиція щодо реалізації простого кодера.

У четвертому розділі описується здійснення простого кодера з допомогою Video Codec SDK. Ці інструменти, які були використані для реалізації. Крім того, розділ розділений на кілька розділів, що описують підхід до реалізації та уривки із використаних кодексів. Розділи розділені обговорюваними принципами процесів на кілька пунктів, отримання зображень з камери, а потім процес кодування відео.

Для тестів продуктивності величини продуктивності системи вимірювали як функцію параметри кодування. Тут вимірювали час оцінки частини коду за допомогою різних проектів, а також середній керований кадр при підключенні більшої кількості камер. Для кваліативних тестів вимірювали коефіцієнт втрати на стиск. Ці тести були проведені на декількох конфігураціях апаратного забезпечення.

**Мета роботи.** Розробка програмного забезпечення кодування 4k-відео в реальному часі на основі SDK Video Codec.

**Об'єктом дослідження** є процес кодування відеоданих.

**Предметом дослідження** є методи кодування 4k-відео в реальному часі.

**Методи досліджень.** Для досягнення поставлених цілей в магістерській роботі використано: методи теорії інформації, кодування та передавання сигналів; методи комп'ютерного моделювання; математичної статистики.

# 1 СТАН ПРОБЛЕМИ КОДУВАННЯ ВІДЕО

## 1.1 Роль стандартів відеокодування в галузі телекомунікацій

Кодування відео є особливо важливим через стиснення даних до найдрібніших втрати якості. У цьому розділі загалом обговорюються відеокодеки, кольорові простори та роль стандартів. Крім того, докладно описано два стандарти, що використовуються в цій магістерської роботі.

Відеокодек – це пристрій або програмне забезпечення, яке використовується для стиснення або декомпресії цифрового відео. Слово кодек складається з двох слів: кодер і декодер. Кодер опосередковує процес стиснення, а декодер – реконструкцію вихідної послідовності відео. Існують кодеки, що містять обидва компоненти, але є також кодеки, які дозволяють лише одну з вищезазначених функцій [1].

Сьогодні програми для цифрового відео залежать від кольоропередачі, тому важливо визначити, як програма буде обробляти цю інформацію. Зображення потребує лише один байт на яскравості дисплея, кольорові мрії mek потрібно один піксель принаймні три. Метод подання інформації про колір або яскравість визначає вибраний колірний простір. Існує кілька кольорів для Шторму, але для цієї роботи були використані пробіли RGB та YCrCb, які обговорюються нижче.

Зорова система людини сприймає колір краще, ніж яскравість. RGB є три компоненти, що зберігаються в тій же роздільній здатності і він вільний від YCrCb пропонує ефективні презентації кольору розділовий яскравості кольору. Цей простір складається з трьох компонентів. Y – компонент освітленості, який можна розрахувати з простору RGB за середньозваженими значеннями R, G та B згідно з формулою 1.1

$$Y = k_r R + k_r G + k_r B, \tag{1.1}$$

де k - вагові коефіцієнти.

Інформація про колір представлена у вигляді різниці кольорів, де кожен

зсув кольору є результатом різниці між складовою яскравості та відповідним кольором 1.2.

$$Cr = R - Y$$
$$Cb = B - Y \quad (1.2)$$
$$Cg = G - Y$$

Повне представлення кольору повідомляється і про один компонент яскравості Y, і про три кольорові зсуви Cr, Cg та Cb, які виражають різницю між яскравістю та інтенсивністю спондируючої кольорової складової. Отже, для повного вираження потрібні чотири компоненти, але це більше, ніж для RGB. Однак сума Cr, Cb та Cg є константою, тому потрібно знати лише два кольорові зсуви, і останній обчислюється. Безумовною перевагою над простором RGB є те, що Cr і Cb можуть виражатися з нижчою роздільною здатністю, ніж Y, оскільки, як зазначалося вище, висяча система людини чутливіша до яскравості, ніж до кольору. Таким чином зменшується пам'ять, яка використовується для зберігання інформації про колір без втрати якості [2].

*RGB*

У колірному просторі RGB кольоровість представлена трьома значеннями. Кожному зі значень присвоюються кольори червоний, зелений та синій. Отриманий колір створюється процесом адитивного розмноження. Це найпростіший спосіб створити будь-який колір. Фільтр захоплення зображення RGB включає композиції червоного, зеленого та синього кольорів. Пікселі зображення мають три години музики для кожного кольорового компонента відповідно до інтенсивності кольору в пікселі. Отриманий колір створюється поєднанням усіх трьох компонентів разом. Приклад аддитивного відтворення, що використовується у відео, наведено на рисунку 1.1 [3].

За всю історію було винайдено та опубліковано багато різних методів кодування відео. Кожен із них пропонує різні інноваційні підходи до цієї проблеми, але не всі з них можуть бути стандартизовані. Стандартизовані мають багато переваг:

• Вони забезпечують співпрацю між кодерами та декодерами.

• Визначаються стандарти для створення платформ, які використовуються, та забезпечується сумісність із відеокодеками, аудіокодеками та транспортними протоколами.

• Вони дуже добре задокументовані, і алгоритми, які вони містять, мають чітко визначені правила ліцензування для використання.

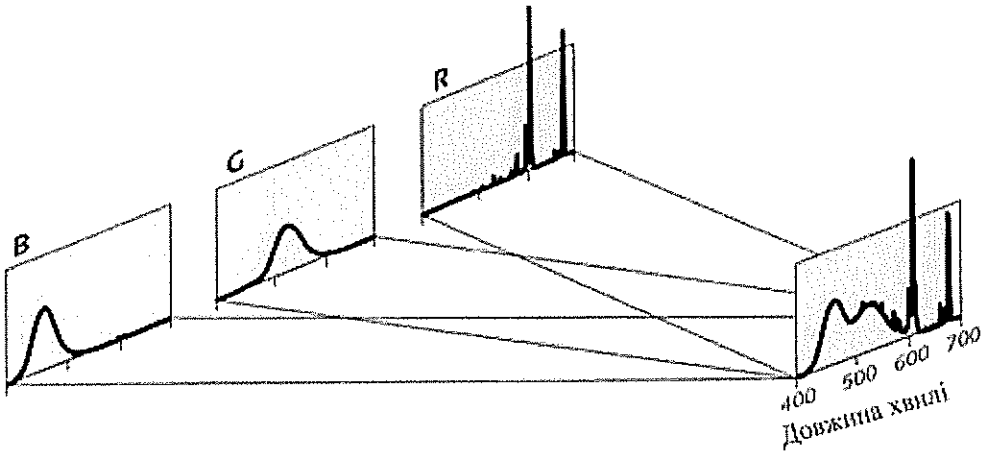



Рис. 1.1. Процес відтворення адитивних матеріалів, що використовується у відео. Кожен компонент R, G і B має незалежний і прямий шлях до отриманого зображення. Спектр отриманого зображення - це сума спектрів кожного з компонентів [3]

## 1.2 Стандарт кодування H.264

H.264 – один із новіших стандартів, що використовується для стиснення цифрового відео. Він також відомий як MPEG-4, частина 10: Розширене кодування відео. Вони почали генеруючу струмінь Організацію ITU-T Video Coding Experts Group (VCEG) і ISO / IEC Moving Picture Experts Group (MPEG). Ці організації розпочали співпрацю у 2000 році та об'єдналися в одну групу, яка називається Спільна спільна група з кодування відео (JCT-VC).

Stand ard був розроблений для використання в широкому діапазоні програм, включаючи IP та бездротові мережі та цифровий кінотеатр. Він був створений у

2003 році на основі попереднього стандарту MPEG-2. Він має багато переваг перед своїм попередником. Найбільшим є краща якість стиснення при тому ж бітрейті. Інші позитивні сторони включають гнучкість стиснення, різні розміри макроблоків, передачу та зберігання відео. Схему кодера можна побачити на рис. 1.2 [1].

*Процес кодування та декодування*

Кодер H.264 включає процес прогнозування, перетворюючи окремі блоки та можливе кодування для отримання стисненого бітового потоку. Потім декодер піклується про процеси декодування, перетворення інверсії та реконструкції для отримання вихідної відеопослідовності. До цих процесів дані спочатку діляться на так звані макроблоки, які зазвичай відповідають розміру 16 x 16 відображуваних пікселів. Це не є передумовою, оскільки на відміну від кодеру MPEG-2 H.264 може обмінюватися даними також до інших вимірів. Розподіл показано на рисунку 1.3. Ця функція дозволяє визначає велику гнучкість і велику ефективність у відео кодування [5].

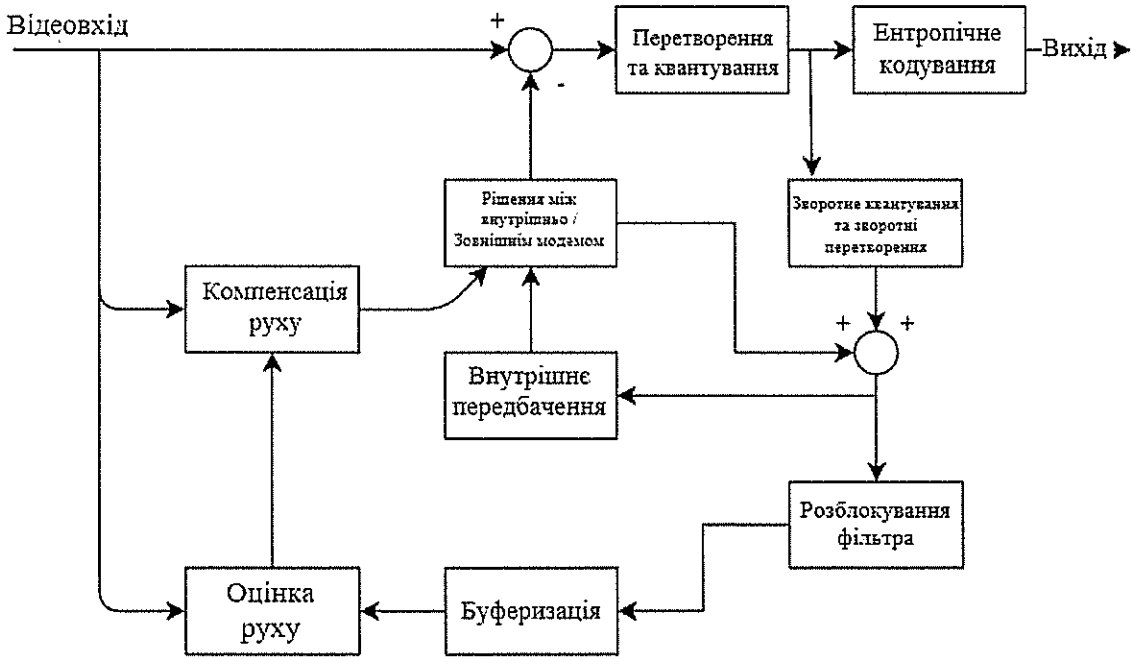



Рис. 1.2 Схема кодера H.264 [4]

Макроблок

Макроблок – це основна одиниця кадру, і при стисненні відео, де

використовується дискретне косинусне перетворення, це не менш важливе поняття. Макроблок зазвичай складається з 16 х 16 пікселів. У варіанті кольорової моделі YCbCr, де Y являє собою компонент яскравості, а Cb і Cr - це кольорові зсуви, макроблок складається з чотирьох блоків Y, одного Cr та одного блоку Cb [6]. Однак він може бути представлений ще однією характеристикою:

• 4: 2: 0 - типовий макроблок, що складається з 4 блоків Y, 1 блоку Cb та 1 блоку Cr

• 4: 2: 2 - макроблок, що складається з 4 блоків Y, 2 блоків Cb і 2 блоків Cr

• 4: 4: 4 - відео з повною пропускною здатністю, містить стільки інформації, скільки у варіанті кольорової моделі RGB. Кожен макроблок містить 4 Y-блоки, 4 Cb-блоки та 4 Cr-блоки.

Таким чином, характеристики (завдяки яскравості та зсуву кольору ) дають змогу зменшити вдвічі кількість даних у поданні моделі 4: 2: 0 порівняно з повною пропускною здатністю моделі 4: 4: 4 або RGB. Приклад усіх згаданих зображень макроблоків наведено на рисунку 1.4 [1].

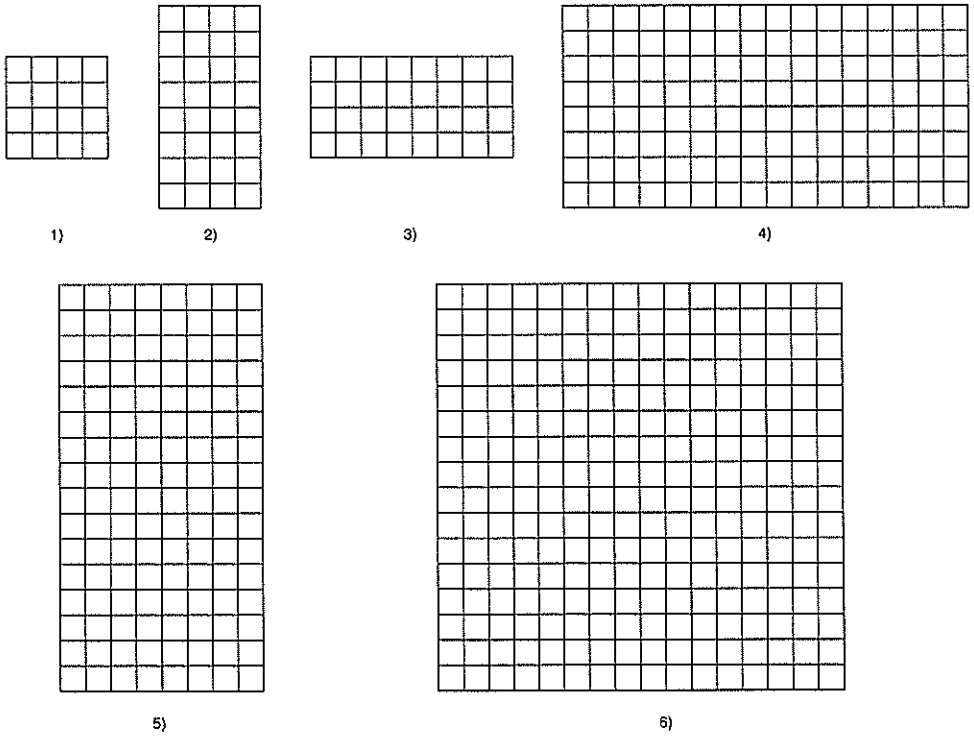



Рис. 1.3 H.264 дозволяє працювати з макроблоками різного розміру. 1) 4 х 4, 2) 4 х 8, 3) 8 х 4, 4) 16 х 8, 5) 8 х 16, 6) 16 х 16, де кожен квадрат представляє один відображений піксель [8]

*Прогнозування*

Поточне зображення, яке обробляється, складається з макроблоків. Мета прогнозування – зменшити надмірність, передбачивши дані, які належатимуть до поточного зображення. Це можна зробити або в поточному кадрі, або на основі вже оброблених Імуноізольованих даних у попередньому кадрі. Перший метод називається внутрішньопередбачуванням, а другий – передбаченням. Кодер передбачає поточний кадр і зберігає дані, які керують цим передбаченням (вектори руху, режим передбачення тощо). Після стиснення ці дані потрібно передати кодеру, щоб можна було відновити зображення. Методи, передбачені стандартом H.264, є більш гнучкими, ніж попередні стандарти. Цілком можливо, для передбачення для макроблоків розміром 16 x 16 або 4 × 4 пікселя. Інтерпретація використовує динамічний розмір блоку в діапазоні від 4 x 4 до 16 x 16 пікселів [7].

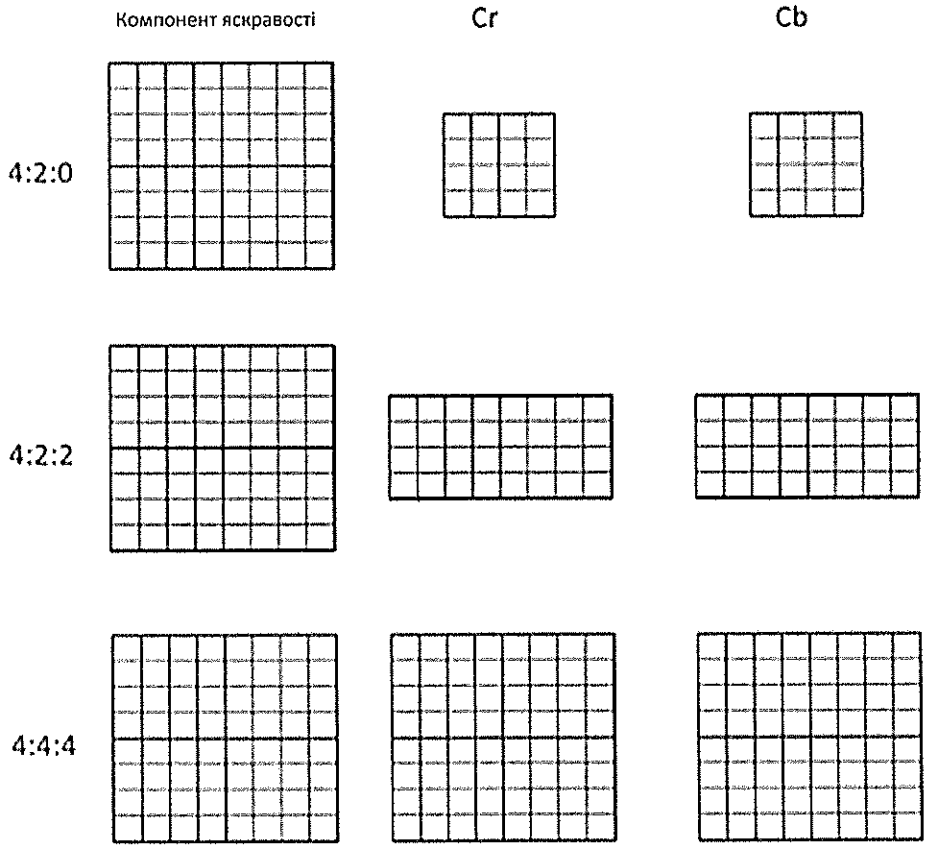



Рис. 1.4. Зображення кольорової моделі YCbCr

Внутрішнє передбачення

Внутрішнє передбачення - це метод, який використовує вже оброблені дані в поточному зображенні та піддає поточному обробленому макроблоку на зображенні на основі подібності. Використовуються 2 методи внутрішньо передбачення.

Перший – це внутрішнє передбачення на маленькому макроблоці, де результат буде точнішим, а розбіжності меншими. Другим можливим методом є внутрішнє передбачення на макроблоку із більшим розміром, наприклад 16 х 16. У цьому випадку прогнозування буде менш ефективним, і отримане зображення матиме недоліки. Однак вибір внутрішнього передбачення для кожного макроблока 4 х 4 повинен бути поданий на декодер, збільшуючи тим самим розмір отриманого файлу [6]. Можна призначити 4 режими внутрішнього передбачення:

• Режим 0 - вертикальний

• Режим 1 - горизонтальний

• Режим 2 - постійний струм (зверху та зліва)

• Режим 3 - площина (відповідно до функції, пов'язаної з попереднім режимом)

Разом можна вибрати одне з дев'яти налаштувань для макроблока 16 х 16 компонента яскравості для кожного 4 х 4 макроблока компонента яскравості. Кожному 4 х 4 макроблоку кольорової складової присвоюється один режим. Ці режими показані на рис. 1.5.

Інтерпрогнозування

Модель інтерпретації працює з поточним зображенням на основі одного або декількох попередніх зображень, які вже були оброблені, див. рис.1.6. Процес прогнозування складається з кількох частин. По-перше, потрібно вибрати область, з якою слід працювати. Потім ви генеруєте інтелектуальний блок, і, нарешті, для цього блоку буде розраховано різницю вихідних областей NAL на зображенні. Ця модель може працювати з макроблоками розміром від 4 х 4 до 16 х 16 пікселів. Як уже зазначалося, він працює з попередніми зображеннями, які зберігаються в

буфері декодованих зображень.

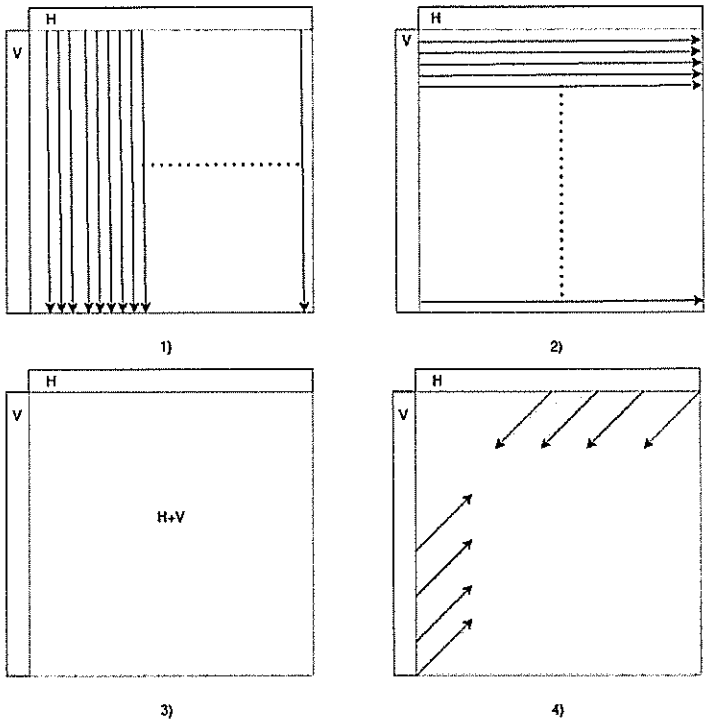



Рис. 1.5. 4 режими, що використовуються для внутрішнього передбачення: 1) Режим 0 - вертикальний, 2) Режим 1 - горизонтальний, 3) Режим 2 - постійний струм (зверху та ліворуч), 4) Режим 3 - площина (відповідно до функції, пов'язаної з попередньою режим) [8]

Зсув між положенням області в поточному кадрі та областю в попередньому кадрі називається вектором руху. Він може вказувати на положення початку контрольної області або її половини або чверті. Положення половини та чверті визначаються інтерполяцією площі попереднього кадру. Кожен вектор руху обчислюється на основі вікових значень руху, що оточують блоки. Блок прогнозування можна обчислити з однієї контрольної області зображення, тоді воно називається макроблоком типу P або B. Якщо це дві області еталонного зображення, воно має тип B і має призначений порядок. Існує також варіант зваженого прогнозування, який розраховував на тимчасову відстань між струмом та посиланнями на нього. Якщо макроблок типу B, то т може бути обчислений без необхідності перерахунку векторів руху. Вони взяті з попередніх зображень [6].

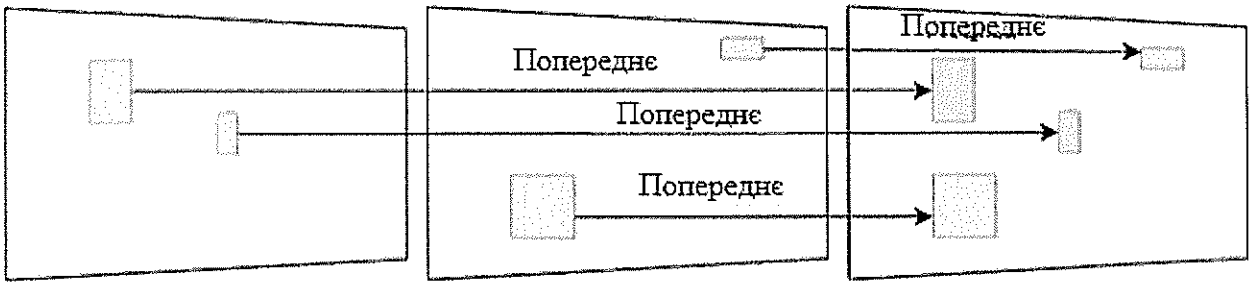



Рис. 1.6. Приклад інтерпретації, яка використовує попередні кадри для створення поточного кадру

*Реконструкційний фільтр*

Фільтр придушення блокової структури застосовується до кожного закодованого макроблока. Його мета - згладити краї після поділу кадру на макроблоки. Фільтр застосовується між зворотним перетворенням та реконструкцією зображення. Відфільтроване зображення використовується для компенсації руху. Цей порядок операцій також збільшує кількість стиснення, оскільки згладжений кадр є більш надійним, ніж не згладжений край. Фільтрування виконується по краях 4 x 4 блоків у макроблоці. Спочатку процес виконується на вертикальних краях блоку яскравості зліва направо, потім на горизонтальних краях зверху вниз.

Далі фільтр застосовується до вертикальних країв кольорового компонента і, нарешті, до горизонтальних країв кольорового компонента [1].

## 1.3 Стандарт кодування H.265

Цей кодек базується на стандартах ITU-T H.264 та його попередниках. Зазвичай ми також можемо зустріти назву HEVC. Як і більшість колишніх Standard One, також був розроблений через необхідність ефективного стиснення для потокових фільмів, комунікації, відеоконференції, носії інформації та телебачення. Стандарт розроблений для використання на широкому діапазоні інтерфейсів і є досить гнучким так що відео можна розглядати як дані, які можна

передавати та зберігати на існуючих та майбутніх пристроях ().

Кодек H.265 був розроблений групою експертів з кодування відео ITU-T (Video Coding Experts Group - VCEG) та групою експертів з рухомих зображень ISO / IEC (Moving Picture Experts Group - MPEG). Його перша версія була випущена в квітні 2013 року вищезазначеними організаціями, які приєднались до спільної групи з кодування відео (Joint Collaborative Team on Video Coding - JCT-VC). З тих пір було випущено ще чотири версії.

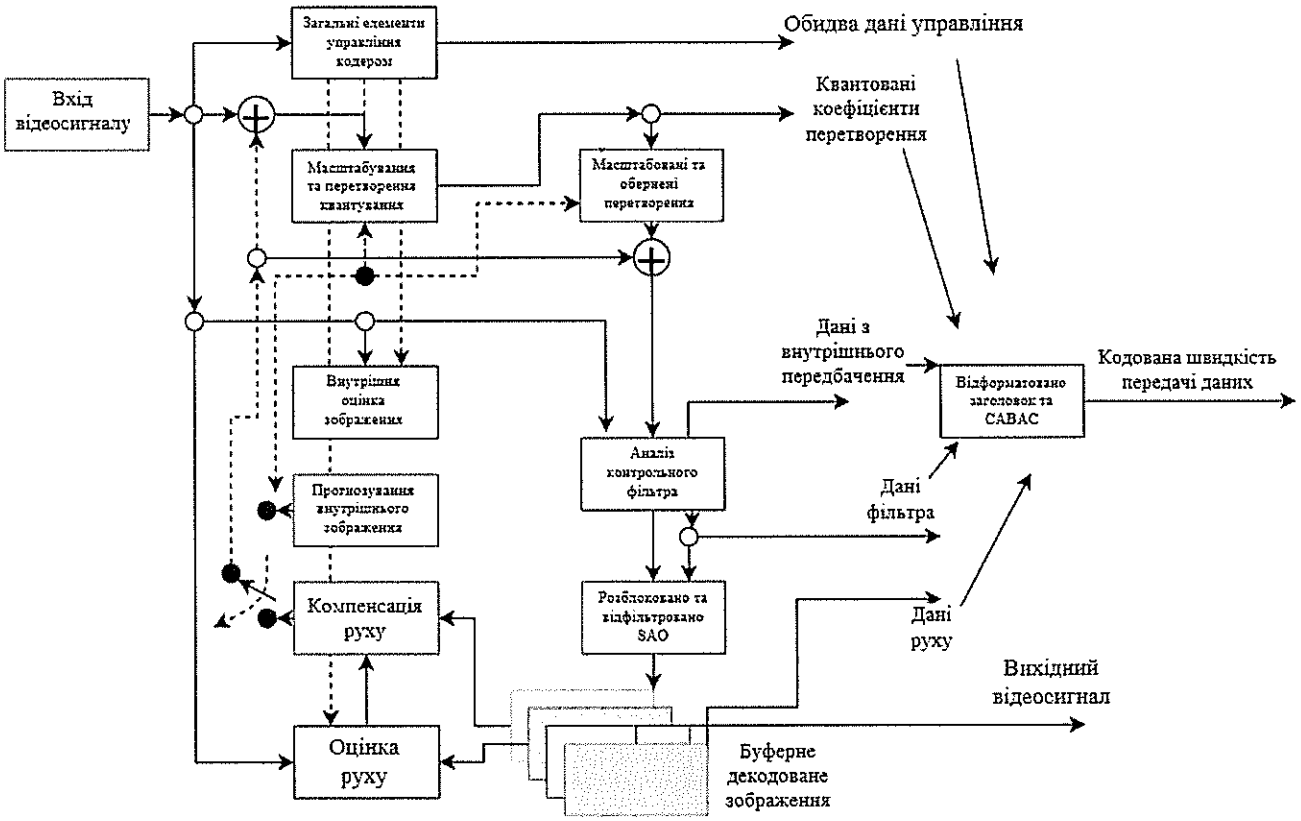



Рис. 1.7. Схема кодера H.265 [9]

*Структура дерева*

HEVC приніс значні зміни у порівнянні зі своїми попередниками, коли він почав використовувати іншу базову структуру дерева блоків. Він базується не на макроблоках, таких як H.264, а на одиницях дерев кодування (CTU - Coding Tree Unit). CTU складається з одного CTB з яскравістю (Coding Tree Block - блок кодування дерева) та двох відповідних кольорових CTB. CTB можна придбати, розміром 16 x 16, 32 x 32 та 64 x 64 пікселів. CTB можуть містити один або

більше СВ (Coding Blocks - кодуючих блоків), які мають квадратну форму і мають мінімальний розмір 8 х 8 для компонента яскравості та 4 х 4 для компонента хромування. Одна яскравість і два синтаксиси сполучення МС містять PU (Prediction Unit - одиниця передбачення) та TU (Transform Unit - одиниця перетворення).

PU, як і CU (Coding Unit), складається з яскравості PB ( Prediction Block - Блок попередження) та кольорової PB, див. рис. 1.8. PU – це синтаксична одиниця, відповідальна за зберігання інформації про внутрішнє передбачення та інтерпретацію. Кожне з передбачень містить свої різні PU. МС може складатися з чотирьох ПУ. TU – це синтаксична одиниця, яка відповідає за зберігання даних перетворень. TU знову складаються з ТВ (Transform Block), який може бути розміром 4 х 4, 8 х 8, 16 х 16 або 32 х 32 пікселів. Поділ СТВ на СВ можна побачити на рис. 1.9 [10].

*Прогнозування*

Внутрішнє передбачення та передбачення базуються на тій самій основі, що і в H.264, з тією різницею, що кількість кутів, які можна використовувати, збільшується при внутрішньому передбаченні.

Внутрішнє передбачення

Стандарт HEVC підтримує 3 основні режими внутрішнього передбачення:

• Постійний струм (прямий компонент)

• Площинний

• Кутовий

На відміну від H.264, де було можливим лише вісім напрямків, кутовий режим внутрішнього передбачення використовує до тридцяти трьох напрямків. Усі можливі напрямки показані на рисунку 1.10. Якщо виконується вертикальне домінантне передбачення, точки зліва від кодованого сегмента називаються основним полем, а точки над кодованим сегментом бічним полем. При горизонтальному домінантному передбаченні основним полем є місце з точками над кодованим сегментом та бічним полем для точок ліворуч від сегмента.

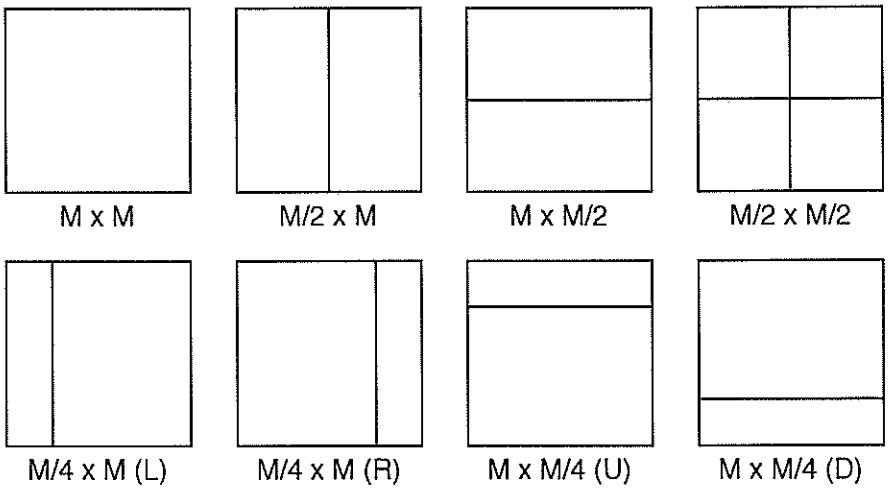



Рис. 1.8. Варіанти поділу блоку кодування на кілька блоків передбачення [10]

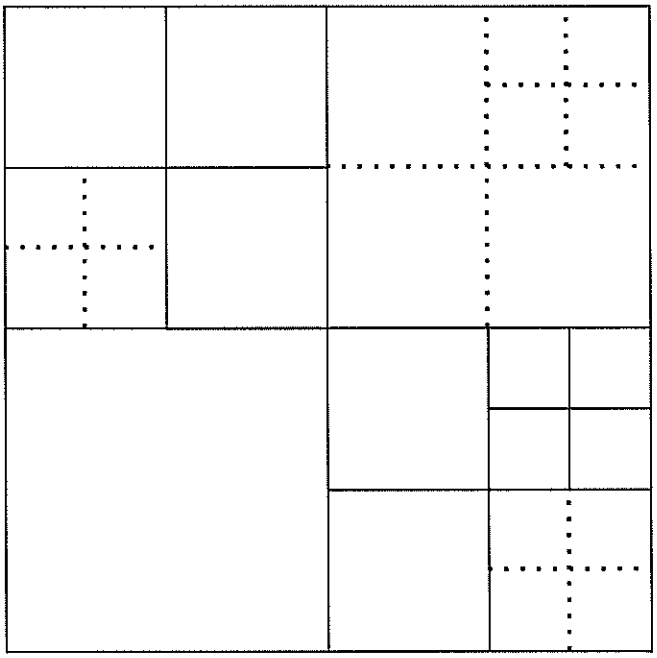


Рис. 1.9. Поділ блоку кодування дерева на блоки кодування та блоки перетворення. Суцільні лінії є межами СВ, а штрихові лінії вказують на ТВ [9]

На додаток до загальної кількості тридцяти трьох режимів внутрішнього передбачення додана можливість використання блоків перетворення, які можуть мати розмір 4 x 4, 8 x 8, 16 x 16 або 32 x 32 пікселів. Якщо це внутрішнє передбачення режиму постійного струму, тоді середнє значення пікселів із сусідніх блоків (верхнього та лівого відносно поточно передбачуваного блоку) обчислюється для прогнозування компонента постійного струму. Планарний

режим є найбільш вимогливим до обчислень з усіх перерахованих режимів, оскільки він використовує двовимірну лінійну інтерполяцію. У кутовому режимі внутрішнього передбачення лінійна інтерполяція точок виконується у напрямку, що визначається кутом. Його величина визначається кутом, який визначає напрямок інтерполяції з вертикальною віссю [9].

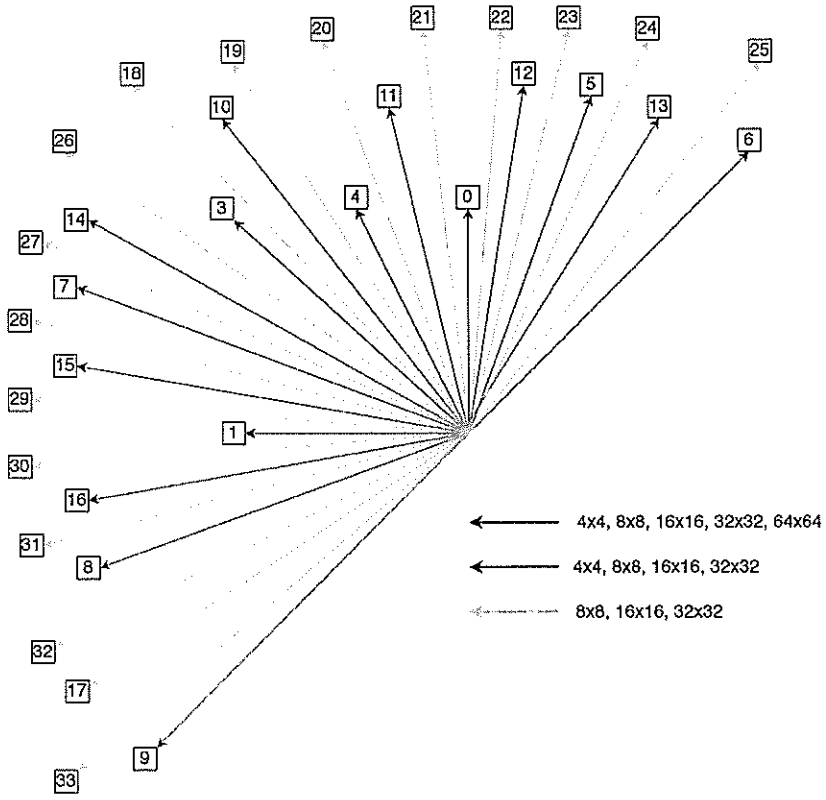



Рис. 1.10. Напрямні режими внутрішнього передбачення [11]

Інтерпрогнозування

Для інтерпретації використовуються два списки посилань L0 та L1. Кожне з них вміщує шістнадцять зображень, де максимум вісім можуть бути унікальними, тому деякі повторюються. Це полегшує процес прогнозування з того самого кадру, але з різними вагами прогнозування. Інтерпретація виконується за допомогою передбачення вектора руху. Він розділений на два режими:

• Злиття

• AMV P2

Для кожної одиниці передбачення кодер вирішує, який із цих режимів

використовувати, і призначає йому відповідний прапор. AMPV використовує дельта-кодування вектора руху і може створювати будь-яке значення вектора руху. HEVC відбирає вектори руху у поля 16 х 16, тобто декодер виділяє простір лише для двох векторів руху L0 та L1 для комірки вектора руху до 16 х 16 пікселів [11].

*Ентропійне кодування*

Ентропічним методом кодування є CABA C3 на рівні CTU. Цей принцип є вошами, подібними до кодування, використовуваного в H.264, але він додав незначні вдосконалення. В цілому, процес ініціалізації був спрощений, і відбувається тільки близько половини перетворень NYC. Існують більш високі швидкості передачі даних, а також можна працювати з конкретною структурою блоку дерева [11].

*Реконструкція фільтра*

HEVC включає можливість використання 2-х фільтрів: вбудований фільтр деблокування (ILD) та зразок адаптивного фільтру зміщення (SAO). Обидва фільтри можна деактивувати. Фільтр ILD подібний до фільтра в H.264, тоді як фільтр SAO є абсолютно новим і застосовується лише після фільтра ILD.

ILD фільтр також називають фільтр для придушення структури блоків. Його завдання - пом'якшити та замаскувати ребра, які були створені під час кодування після поділу на блоки на рівні TU та PU. У H.264 цей фільтр використовувався понад 4 х 4 пікселі, у HEVC застосовується лише понад 8 х 8 зразків. Спочатку всі вертикальні ребра подавляються, а потім застосовується фільтр до всіх горизонтальних країв. Фільтрування виконується по всіх краях до мінімального розміру 8 х 8 пікселів.

Після придушення блокової структури застосовується SAO фільтр. Застосовується лише до зразків, які відповідають усім вимогам. Він застосовується на шарі блоку CTB для кожного пікселя цього блоку. SAO-ltr використовується як для компонентів яскравості, так і для кольоровості. Він

присвоює кожне з чотирьох встановлених значень о кожному блоку CTB, який знаходиться в діапазоні від -7 до 7 для 8-бітової відеопослідовності. Кодер вибирає параметри таким чином, щоб зображення за замовчуванням максимально нагадувало вихідне зображення [11].

*Різниця між H.264 та H.265*

У зв'язку з необхідністю переміщувати відео в Інтернет і транслювати відео, вимоги до ефективності стиснення зростають, щоб зайняти якомога менший розмір відео. Не менш важливою причиною є розширення камер 4K та 8K. Були проведені дослідження, де HEVC зміг зменшити обсяг даних на 52% краще при 480p, а при 4K UHD мав на 64% кращі результати, ніж H.264. Нарешті, отримане відео з однаковим розміром даних має набагато кращу якість візуального зображення [12].

Великою зміною є використання CTU4, який забезпечує можливість роботи з різними розмірами, ніж макроблок H.264. CTU може сформувати блок розміром від 4 x 4 до 64 x 64 пікселів порівняно з H.264, де максимальний розмір становив 16 x 16 пікселів. Інша відмінність полягає у кількості кутових векторів при внутрішньому передбаченні, де HEVC забезпечує до тридцяти трьох напрямків порівняно з вісьма напрямками в H.264. І останнє, але не менш важливе, HEVC придумав фільтр SAO, який допомагає поліпшити реконструкцію зображення, і не був реалізований у H.264.

# 2 АРХІТЕКТУРА ТА ВИКОРИСТАННЯ ВІДЕОКАРТ

## 2.1 Апаратне забезпечення GPU

У цьому розділі розглядається архітектура та використання графічних карт. Він висвітлює основні поняття, такі як GPU та GPGPU з точки зору історії, архітектури та функцій.

GPU – це апаратне забезпечення, основною функцією якого є швидке виконання графічних розрахунків. На сьогоднішній день це невід'ємна частина багатьох пристроїв, таких як вбудовані системи, мобільні телефони, комп'ютери та ігрові пристрої. Їх архітектура розроблена таким чином, що графічний процесор може обробляти великі блоки даних паралельно.

*Історія графічного процесора*

Перші згадки про графічні одиниці були вже в 70-х роках минулого століття, де для безпечового паралельного функціонування графічних та текстових додатків та візуалізації на екрані. З часом їх почали використовувати для візуалізації 2D-об'єктів, а в 1990-х роках вони були пристосовані для візуалізації та обчислення за допомогою 3D-об'єктів. Багато Популярні ції відбулося в 1999 році, коли компанія представила на ринок Nvidia Ge Force 256, який вважається першим офіційним GPU. У 2001 р. Була випущена нова відеокарта, яка мала повністю програмоване обладнання для перетворення вершин та генерації фрагментів [13].

*Архітектура графічного процесора*

Сьогодні найбільшими виробниками графічних адаптерів є Nvidia та ATI. Раніше ці дві компанії випустили найбільшу кількість архітектур. Сучасний графічний процесор складається з глобального планувальника, глобальної пам'яті, кешу текстур, мультипроцесора та блоку текстурування. Архітектура показана на рис. 2.1. Всі розрахунки відбуваються в процесорах, які згруповані в багатопроцесорні. Ці одиниці обчислення оптимізовані для розрахунків SIMD, тобто вони виконують одну операцію з багатьма даними. Цей процес можна легко

розпаралелювати, і дані обробляються на якомога більшій кількості одиниць одночасно в будь-якому порядку.

Причиною використання графічних процесорів є пріоритет обчислювальної потужності. Більшість програм можуть працювати з продуктивністю процесора. Однак деякі програми вимагають більш складних розрахунків для своєї роботи, і звичайний процесор не буде достатнім рішенням.

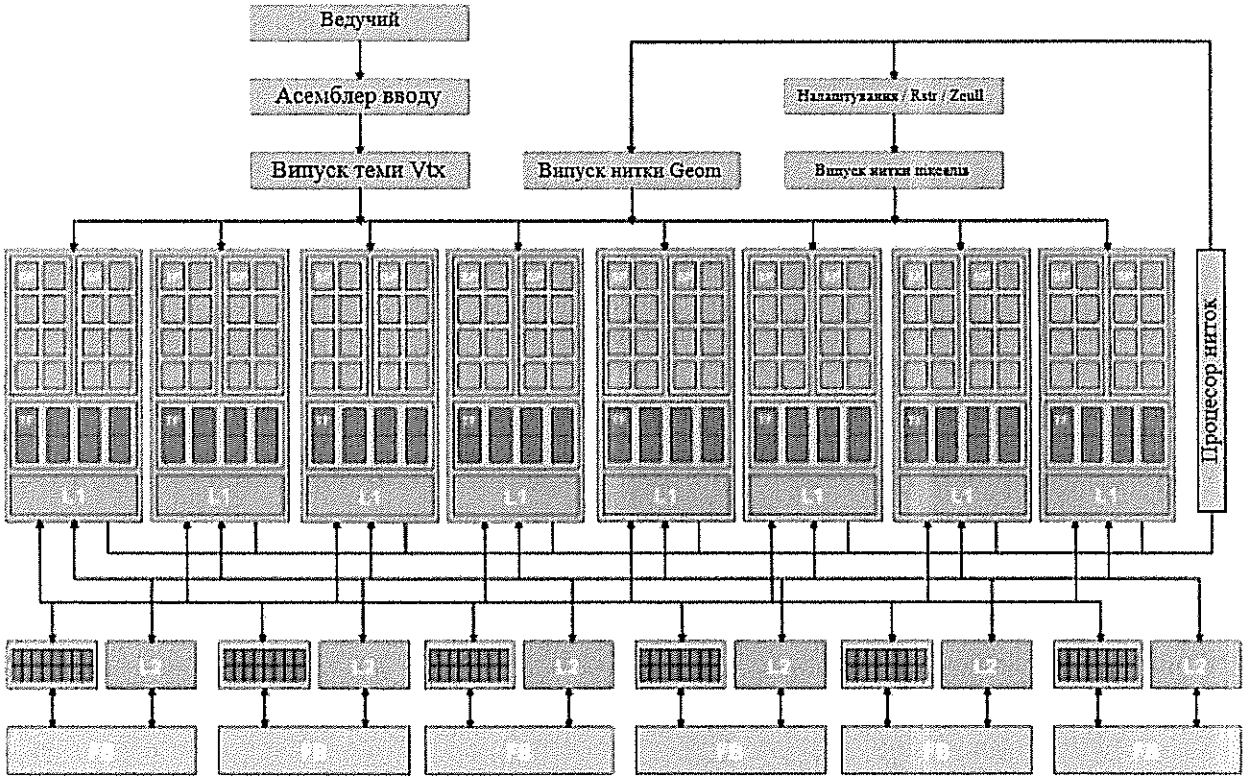



Рис. 2.1. Архітектура сучасного графічного процесора [14]

Для цих програм графічний процесор може багаторазово скоротити обчислювальний час. Порівняння максимально- формального потоку даних через пам'ять RAM між процесором і графічним процесором з часом показано на рис. 2.2 [14].

## 2.2 Графічний процесор GPGPU

GPGPU – це графічний процесор, розроблений як співпроцесор для спеціальних програм. Він тому фірмові найменування, що має програмований блок для розпаралелених обчислень. Історично перші GPGPU почали з'являтися з виходом графічного процесора Geforce 3 від Nvidia. Це принесло новинку - програмований шейдер. На той час апаратні компоненти ще не дозволяли обчислювати складні алгоритми, тому їх використовували переважно для вершинних та піксельних операцій. З приходом Geforce 8 стала програмою цій простіше завдяки підтримці і вдосконалення мови, мову програмування для програмування шейдеров. Сюди входить, наприклад, характеристика поверхонь та предметів. Ця мова програмування використовувалася не тільки для графічних розрахунків. Під час роботи Windows 8.1 з'явилися ядра шейдерів DirectX 10, які визначаються стандартами якісних компонентів [15].

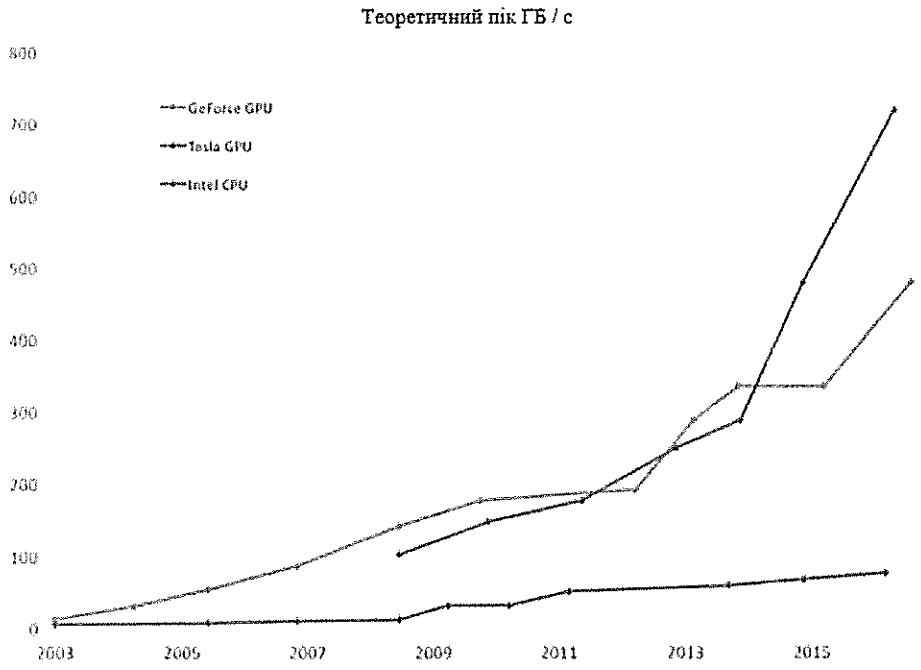



Рис. 2.2. Графік порівнює обсяг даних, що протікають через оперативну пам'ять, розрахований в ГБ / с

Вищі значення означають вищу продуктивність, оскільки не доведеться

чекати звільнення пам'яті. Знову ж таки, є порівняння декількох варіантів GPU та CPU [16].

## 2.3 Обчислювальна платформа CUDA

CUDA була представлена світові в листопаді 2006 року компанією Nvidia. Це обчислювальна платформа загального призначення та програма програмування, яка впливає на обчислювальне ядро GPU. Він був розроблений для виконання складних паралельних обчислень більш ефективно, ніж це було можливо на центральному процесорі. Він підтримує ряд мов, включаючи C, C ++, Fortran або DirectCompute. Він підтримується операційними системами Windows, Linux та MacOs як у 32-розрядної, так і в 64-розрядної версіях. Раніше підтримувались лише деякі графічні адаптери компанії, але згодом підтримка була поширена на всі відеокарти Nvidia [17].

*Модель програмування*

Все залежить від програми, яка працює на мікросхемі графічного адаптера під назвою Kernel. Це функція, яку заявляє програміст. Після виклику він виконується паралельно кількома різними потоками. Волокна - це найменша і в той же час основна одиниця для проведення розрахунків. Їх накладні витрати майже нульові, а одне волокно має дуже низьку продуктивність. Тому ці волокна агрегуються у більші одиниці. Термін деформація означає скупчення волокон, які виконуються паралельно. Якщо деформації виконуються в одному мультипроцесорі, то вони позначаються як блок потоків. Зазвичай він містить від 64 до 512 ниток. Цей блок може передавати деякі дані між своїми деформаціями. Набір блоків потоків, де працює одне ядро, називається сіткою.

Поступово виявляється, що волокна не перераховуються, і воно надсилається на вказаний номер для мультипроцесора. Якщо деформація отримає всю необхідну інформацію, вона буде виконана. Якщо ні, тоді він чекає даних, а тим часом виконується деформація, яка вже отримала необхідні дані. Доставки даних потрібна пам'ять, яка може працювати з вимогами покладених паралельно.

Пам'ять також повинна мати високу пропускну здатність. Завдяки цьому принципу декомпозиції ідея паралельності з робочих розрахунків стає зрозумілою. Для одного ядра потрібні тисячі потоків. Для досягнення максимальної продуктивності, отже, необхідно досягти хорошого співвідношення оброблених блоків до кількості мультипроцесорів та правильного здійснення декомпозиції [18].

### 2.4 SDK для відеокодеків Nvidia

Nvidia Video Codec SDK - це повний набір API, який забезпечує апаратне кодування та декодування Acc Lenii. SDK сумісний з операційною SYS тему Windows, і Linux. Він містить два інтерфейси, NVENCODE і NVDECODE. Графічні процесори Nvidia мають один або кілька декодерів та кодерів на додаток до ядер CUDA. Ефективні для ефективної роботи прискореної продуктивності під час кодування та декодування відео. Ці прискорювачі забезпечують більшу швидкість кодування та декодування, ніж обробка відео в режимі реального часу. Це робить їх особливо придатними для програм перекодування.

Переваги використання SDK Video Codec:

• Гнучкість – динамічна роздільна здатність, вибір потоку даних, вибір CABA C8 або CAVL C, параметри синхронного та асинхронного дзвінків.

• Помилка concealmen – забезпечити плавне відео потоку і звести до мінімуму втрати даних.

• Якість – двофазні режими а для підвищення якості, різноманітність і якість пресетів вище CONUS.

*NVCUVID*

Попередником SDK Video Codec був відеодекодер Nvidia (NVCUVID). Це API, який надає розробникам можливість використовувати апаратні функції для декодування відео в декількох операційних системах. Цей API підтримує відеоформати MPEG-2, VC-1, H.264 та HEVC. Додатки, які використовують цей API, можуть декодувати відео безпосередньо в пам'ять. API може працювати з

CUDA, OpenGL та DirectX. Для намети -стандарту формат висновок використовує відеодоріжки NV12, який згаданий нижче NVENC, використовувані в якості стандартного вводу. Цей API сьогодні більше не підтримується і вважається застарілим [19].

Таблиця. 2.1.

Підтримка NVENC на різних поколіннях графічних процесорів [20]

| GPU | H.264 YUV 4: 2: 0 | | H.264 YUV 4: 4: 4 | | H.264 без втрат | | H.265 YUV 4: 2: 0 | | H.265 YUV 4: 4: 4 | | H.265 без втрат | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Максимальні значення - ^ | Колір | Відзнака | Колір | Відзнака | Колір | Відзнака | Колір | Відзнака | Колір | Відзнака | Колір | Відзнака |
| Кеплер | 8-розрядна | 4096 x 4096 | X | X | X | X | X | X | X | X | X | X |
| Максвелл (Буття 1) | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 | X | X | X | X | X | X |
| Максвелл (2-е покоління) | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 | X | X | X | X |
| Максвелл (3-е покоління) | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 |
| Паскаль | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 | 10-розрядна | 8192 x 8192 | 10-розрядна | 8192 x 8192 | 10-розрядна | 8192 x 8192 |
| Вольта | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 | 10-розрядна | 8192 x 8192 | 10-розрядна | 8192 x 8192 | 10-розрядна | 8192 x 8192 |
| Тьюрінга | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 | 8-розрядна | 4096 x 4096 | 10-розрядна | 8192 x 8192 | 10-розрядна | 8192 x 8192 | 10-розрядна | 8192 x 8192 |

*NVENC*

З виходом першого графічного процесора Kepler покоління - GeForce 600 Nvidia та NVENC. Це апаратний елемент, який забезпечує повністю прискорене кодування відео. NVENC не залежить від продуктивності графіки, таким чином, графічний процесор може повністю придане виконувати інші операції. Наприклад, під час запису з гри використовується NVENC, і графічний процесор може використовувати всю свою здатність для рендерингу самої гри. Огляд підтримуваних відеокарт за поколіннями наведено в таблиці 2.1.

Незалежність графічного процесора – не єдина перевага технології NVENC. Інші включають:

• Низьке споживання енергії з точки зору зменшення передачі даних та фіксованого обладнання.

• Низька відповідь – немає необхідності передавати дані між процесором та графічним процесором.

• Висока продуктивність.

• Більша щільність даних в одному місці.

• Масштабованість.

• Просте програмування для власного використання. Це дозволило розробити більш Jazy ніс для Linux і Windows. До мов програмування належать C, C ++, Fortran тощо.

Велика різниця між відеокартою також NVENC кількість фішок, які містили Худжі. Відповідно, існує обмеження на максимальну кількість потоків, які можна використовувати паралельно паралельно. Але це не умова. Наприклад, покоління GeForce Maxwell та Kepler обмежено 2 потоками, але Quadra вже має вищу максимальну кількість потоків [21].

# 3 АНАЛІЗ ТА РОЗРОБКА SDK VIDEO CODEC

## 3.1 Загальний принцип Video Codec SDK

Метою магістерської роботи є впровадження кодера для стиснення відео 4K в режимі реального часу з використанням NVENC та Video Codec SDK. Для найкращої оптимізації програми та аналізу використання різних API, таких як CUDA, DirectX або OpenGL. Також необхідно встановити обмеження для конкретної апаратної конфігурації та контролювати вплив заданих параметрів на загальний час та навантаження пам'яті щодо результуючого стиснення. У цій главі йдеться про детальне пояснення функціонування принципів в окремих проектах без змін.

Вибрано два проекти AppEncCuda і AppEncD3D11, які є частиною Video Codec SDK 9.1, є вільно доступними прикладами програм. Перший використовує CUDA для кодування, а другий виконує кодування DirectX. Обидва клієнтські додатки використовують виклики функцій із API NVENCODE, наданих nvEncodeAPI.dll для Windows та libnvidia-encode.so для Linux. Ці бібліотеки встановлюються як частина драйвера відеокарти. Третій проект AppEncOpenGL був лише проаналізований, але реалізація на нього не вплинула, оскільки його не можна скомпілювати в середовищі Windows, а лише в операційній системі Linux. Що стосується проекту перекладу для DirectX, який неможливо виконати в темі операційної системи Linux, автор розширив Hodl як пріоритет цього проекту над OpenGL. Загальна процедура того, як програма виконує процес кодування, описана на наступних етапах [22]:

1. Ініціалізація кодера.
2. Встановлення параметрів кодування.
3. Розподіл вхідного та вихідного буфера.
4. Копіювання кадрів у вхідний буфер і прочитування вихідного потоку бітів.
5. Завершення частини кодування.

6. Перерозподіл вхідного та вихідного буфера.

6. Перерозподіл вхідного та вихідного буфера Перед ініціалізацією API перевіряється, чи є версія драйвера відеокарти достатньою для запущеної версії SDK Video Codec. Коли ініціалізується кодер, викликається метод SetInitParams класу NvEncoderInitParam, який з'ясовує всі параметри, задані для кодування. Ті, які не були заповнені під час запуску програми, мають значення за замовчуванням. Потім функції клієнта IsCodecH264 та IsCodecHEVC викликаються клієнтом, щоб дізнатись кількість GUID1, які представляють необхідні кодеки для кодування відеопослідовності. Відповідно, він виділяє буфер для збереження інформації про підтримувані GUID. Наступним кроком є перевірка, чи можна використовувати обраний пресет і чи підтримується він графічним адаптером. Попередньо встановлена настройка - це загальний вибір того, що очікується від кодування, а потім задає ряд параметрів для досягнення найкращого результату для обраного пресету. Інформація про можливі пресети та їх налаштування наведена в таблиці 3.1. Ця таблиця призначена лише для довідки, оскільки не кожен графічний процесор підтримує всі попередньо встановлені налаштування.

Наступним кроком є вказання профілю кодування для конкретного випадку. Наприклад, кодування відео для відтворення на iPhone має інший профіль, ніж відео на диску Blue-Ray. Потім визначається формат вхідного файлу. Щоб ініціалізувати сам кодер, потрібно викликати функцію CreateEncoder з дійсною конфігурацією кодера.

Після ініціалізації сеансу клієнт може розпочати виділення пам'яті для вхідних та вихідних буферів. Якщо потрібно кодувати кілька потоків одночасно, потрібно виділити однакову кількість вхідних та вихідних буферів.

Вхідні файли обробляються і викликається функція ConvertRGBToNV12 (лише DirectX, Cuda може мати кілька форматів вхідних файлів), яка перетворює вихідне зображення BGR у NV12. В основному, мова йде про вирівнювання пікселів за форматом NV12 та запис у вхідний буфер. Потім зображення передається для кодування (описується окремо в кожному проекті). Нарешті,

буфер вивільнюється, результат перераховується і цільовий файл закривається.

Таблиця. 3.1

Інформація про можливі передустановки, їх налаштування та використання

| Попередньо встановлені | Налаштування | Зосередження на додатках |
|---|---|---|
| ВИСОКА ЯКІСТЬ | B кадрів, CABAC, 8x8 Transform, все інтро режимів, все взаємопов'язані режими, VBR RC, GopLength 30 | перекодування з високим бітрейтом |
| ВИСОКА ПРОДУКТИВНІСТЬ | Без кадрів B, CAVLC, P16x16, Внутрішні режими 16x16 та 4x4, VBR, GopLength 30 | багаторазове перекодування |
| НИЗЬКА ЛАТЕНТИЧНІСТЬ HQ | Без B кадрів, CABAC, все інтро режимів, все взаємопов'язаних режими, одного кадру VBV 2 PASS, нескінченний GOP | хмарні ігри, miracast, відеоконференції |
| НИЗЬКА ЛАТЕНТИЧНІСТЬ HP | Без B кадрів, CABAC, все інтро режими, все взаємопов'язаних режими, одного кадру VBV 2 PASS, нескінченна GOP, менший діапазон пошуку по порівнянні з HQ | хмарні ігри, miracast |

### 3.2 AppEncCuda

Програми незабаром перевірять правильність вхідних аргументів, а потім ініціалізують параметр, можна скористатися NvEncoderInitParam (). Всі основні параметри повинні передаватися за допомогою перемикачів. Інші розширені налаштування, такі як попередньо встановлений вибір або параметр B-кадру, задаються за допомогою розширених параметрів, які аналізатор читає як маркери. ЩоБ API використовував CUDA для стиснення, параметр

NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::deviceType потрібно змінити на NV_ENC_DEVICE_TYPE_CUDA. Для ініціалізації необхідно створити, знайти та перевірити, чи можна використовувати пристрій CUDA та чи присутня правильна версія драйверів. CUDA Pou living доступна як в Linux, так і в Windows 7 та пізніших версіях. Якщо налаштування CUDA є дійсним, можна створити контекст CUDA t, викликаючи метод cuCtxCreate з аргументами [23]:

• CUcontext * pctx – вказівник на контекст CUDA;

• непідписані прапори int – позначка того, як CUDA повинна обробляти багатопотоковий доступ;

• CUdevice dev – знайдений на даний момент пристрій CUDA, який буде використовувати програма. Метод створення контексту реалізований в API CUDA.

Після встановлення всіх параметрів та ініціалізації контексту викликається порожня функція EncodeCuda з такими аргументами:

• int nWidth - ширина кадру;

• int nHeight - висота рами;

• NV_ENC_BUFFER_FORMAT eFormat - фільми вхідного формату (треки NV12, YV12, IYUV, YUV444, YUV, ARGB тощо);

• NvEncoderInitParam encodeCLIOptions - інформація про вхідні параметри програми:

• CUcontext cuContext - контекст CUDA;

• std :: ifstream & fpIn - вказівник на вхідний файл;

• s td :: ofstream & fpOut - вказівник на вихідний файл.

Тепер сам кодер InitializeEncoder (EncoderClass & pEnc, NvEncoderInitParam encodeCLIOptions, NV_ENC_BUFFER_FORMAT eFormat) тепер ініціалізований і конфігурація кодера встановлена. Якщо в аргументах не було зазначено одного пункту, програма Menta Основні параметри для пеніса Kompri, такі як вхідні та вихідні файли, а потім найважливіші, які будуть змінені для подальшого тестування, перелічені наступним чином:

• i nput - шлях до вхідного файлу;

• output - шлях до вихідного файлу;

• кодек - можна використовувати або H.264, або HEVC (NV_ENC_H264 або NV_ENC_HEVC);

• розмір - роздільна здатність вхідної відеопослідовності (наприклад, 1280 x 960);

• бітрейт - потік даних;

• fps - кількість кадрів в секунду;

• rcMode - режим управління швидкістю (CONSTQP, VBR, CBR, VBR MINQP, CBR_LOWDELAY_HQ, CBR_HQ, VB_HQ);

• B-кадри - опція B-кадрів;

• QP - тільки тоді, коли режим CONSTQP, являє собою константу, яка визначається квалітет;

• попередньо встановлений - попередньо встановлений вибір, параметри наведені в таблиці 3.1.

Потрібно розподілити вхідні та вихідні буфери перед кодуванням. Кодер після етапу завантаження кадр за кадром вхідних відеопослідовностей з використанням навантажувального кадру NVENCSTATUS (uint8_t yuvInput * [3] HANDLE hInputYUVFile, frmIdx uint32_t, uint32_t width, height uint32_t, uint32_t & numBytesRead), який вставляє в буфер даних кадри. Функція контексту заблокована і приходить до копіювальної на CUDA. Потім сам кадр кодується відповідно до обраних параметрів за допомогою функції NvEncEncodeFrame, яка вже є частиною API NVENC. Згодом закодований кадр записується у раніше вибраний вихідний файл. Діаграма випадків використання показана на рис. 3.1 зліва.

### 3.3 AppEncD3D11

По-перше, визначається, чи можна використовувати DirectX, оскільки він підтримується лише Windows 7 та новішими версіями. Функція ParseCommandLine_AppEncD3D отримує аргументи програми, вибрані перед виконанням. Їх дійсність перевіряється, і функція NvEncoderInitParam ініціалізує параметри. За допомогою методу CreateDXGIFactory1 (REFIID riid, _COM_Outptr_ void ** ppFactory) створюється фабрика DXGI 2, яка реалізує методи та функції для роботи з DirectX. Метод D3D11CreateDevice (params) створює пристрій, який представлятиме графічний адаптер. Потім встановлюються всі необхідні параметри для 2D масиву текстур, такі як ширина, висота, розмір масиву, формат, прапор доступу до процесора тощо [24].

Метод CreateTexture2D (const D3D11_TEXTURE2D_DESC * pDesc, const D3D11_SUBRES- OURCE_DATA * pInitialData, ID3D11Texture2D ** ppTexture2D) створює двовимірний масив текстур із параметрами, заданими на попередньому кроці. Перетворювач, який перетворює зображення з RGB в NV12, скидається, створюючи новий об'єкт класу RGBToNV12ConverterD3D11 (ID3D11Device * pDevice, ID3D11DeviceContext * pContext, int nWidth, int nHeight).

Ініціалізація DirectX та виділення пам'яті для завершення завершено, тому можна викликати функцію void Encode з аргументами:

- ID3D11Device * pDevice - вказівник на пристрій DirectX;
- ID3D11DeviceContext * pContext - вказівник на контекст DirectX;
- RGBToNV12ConverterD3D11 * pConverter - вказівник на функцію перетворення з RGB в NV12;
- int nWidth - ширина кадру;
- int nHeight - висота рами;
- NvEncoderInitParam encodeCLIOptions - дані на зазначеному вхідних параметрів , що кодують;
- bool bForceNv12 - вказує, чи потрібно вводити зображення, що

конвертуються під чан, для монетних доріжок NV12;

• ID3D11Texture2D * pTexSysMem - вказівник на 2D-поле текстури;

• std :: ifstream & fpBgra - вказівник на вхідний файл;

• std :: ofstream & fpOut) - вказівник на вихідний файл.

Тепер сам кодер NvEncoderD3D11 створюється з аргументами:

• ID3D11Device * pD3D11Device - вказівник на пристрій DirectX;

• uint32_t nWidth - ширина кадру;

• uint32_t nHeight - висота кадру;

• NV_ENC_BUFFER_FORMAT eBufferFormat - формат вхідних даних, який може приймати значення NV12 або BGRA в залежності від прапора bForceNv12. Під час ініціалізації викликається функція SetInitParams, яка встановлює всі параметри для кодування. Якщо деякі з них не були вказані під час запуску програми, вони встановлять значення за замовчуванням. Ця функція також виявляє GUID для вказаного пресету та кодека. Якщо він знаходить невідповідність в Уве з по параметрам, програма завершується з повідомленням про помилку «Невірний параметр».

Після успішної ініціалізації кодера починається фаза кодування. На початку функції кодування створюється допоміжний інтелектуальний покажчик std :: unique_ptr <uint8_t []> pHostFrame (новий uint8_t [nSize]); в який завжди буде вставлений кодований кадр. Розумний вказівник має ту перевагу, що пам'ять не потрібно деактивувати вручну, але вказівник контролює, коли він більше не використовується, а потім автоматично деактивується. Це зменшує ймовірність втрати пам'яті. Кодер викликає метод GetNextInputFrame (), який виділяє буферний простір і повертає вказівник на цю пам'ять. Фрейм завантажується за допомогою функції std :: streamsize ReadInputFrame (const NvEncInputFrame * encoderInputFrame, std :: ifstream fpBgra, char * pHostFrame, ID3D11DeviceContext * pContext, RGBToNV12ConverterD3D11 * pCon11S3D11), на попередніх позиціях були записані такі параметри . Функція копіює зображення з вхідного СОУ бору в буфер DirectX.

Після завантаження кадру в буфер кодер викликає метод void EncodeFrame,

в якому він відображає вхідні буфери кодера в поточно закодований кадр і виконує процес кодування. Якщо процес успішний, він вставляє отриманий кадр у пам'ять, на яку вказує параметр vPacket. Останнім етапом кожного циклу є запис кодованого кадру у вихідний файл. Це воно

робить це за допомогою функції запису ostream з аргументами:

• const char * s - вказівник на поле символу;

• streamsize n - кількість символів, які потрібно вставити.

Процес кодування закінчується відсутністю більше кадрів у вхідному файлі, тому кількість очікуваних символів не дорівнює фактично завантаженим символам. Якщо це трапляється, викликається метод кодера EndEncode (), який надсилає прапор закінчення введення і звільняє буфери, зарезервовані кодером. Потім викликається метод кодера DestroyEncoder (), який звільняє буфери, зарезервовані DirectX, деінітизує кодер і закриває вхідні та вихідні файли. Діаграма випадків використання показана на рисунку 3.1 праворуч.

### 3.4 AppEncGL

Як згадувалось у вступі до цього розділу, цьому проекту не приділялося багато уваги завдяки виконуваності лише в операційній системі Linux. Він керувався лише аналітичним проектом з використанням кейсу використання.

Спочатку завантажуються введені користувачем аргументи програми. Слід використовувати функцію GraphicsSetupW indow (contextType), щоб створити новий контекст OpenGL. Дійсність вхідних параметрів перевіряється чи успішно створено контекст OpenGL. Функція void EncodeGL викликається з такими аргументами:

• char * szInFilePath - вказівник на вхідну відеопослідовність;

• char * szO utFilePath - вказівник на вихідний файл, куди будуть записані зображення після кодування;

• int nWidth - ширина кадру;

• int nHeight - висота рами;

• NV_ENC_BUFFER_FORMAT eFormat - тип вхідного файлу, який може приймати значення YUV або NV12;

• NvEncoderInitPa ram * encodeCLIOptions - параметри вхідних програм.

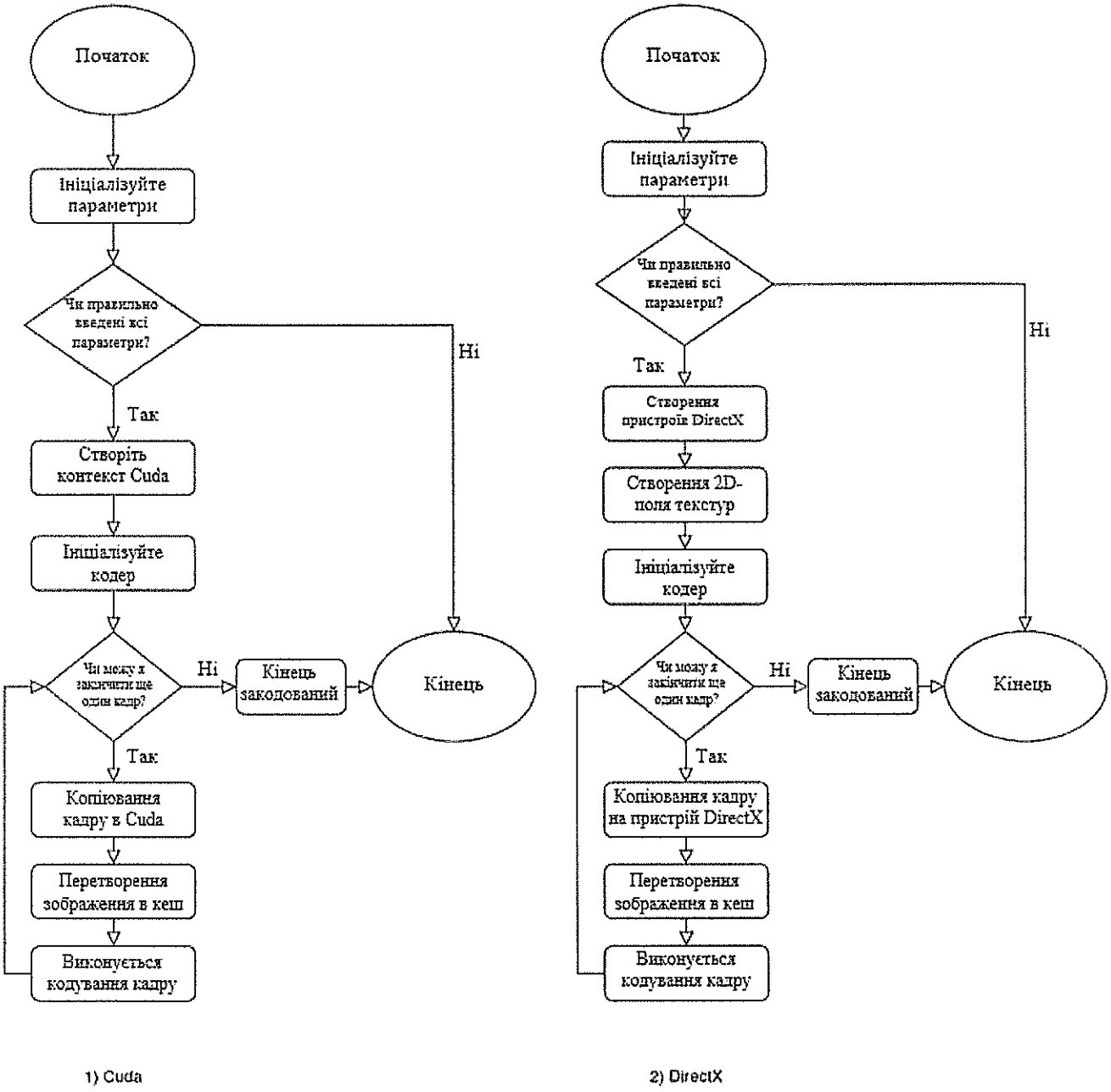



Рис. 3.1. Алгоритм використання проектів AppEncCuda та AppEncD3D11

Функція відкриває вхідний та вихідний файл. Потім створюється новий екземпляр кодера відповідно до шаблону NvEncoderGL. Параметри кодера ініціалізуються і призначаються кодеру за допомогою методу CreateEncoder (initializeParams). Якщо кодер успішно створений та ініціалізований, він починає

фазу кодування, де в циклі NaCl зберігає зображення з вхідного файлу за допомогою методу кодера GetNextInputFrame (). Завантажений кадр кешується в буфері текстури OpenGL і кодується за допомогою методу EncodeFrame (). Потім отриманий фрейм записується у вищезазначений файл stupního Якщо весь кадр ще не завантажений у вхідному фільмі, викликаються методи EndEncode () та DestroyEncoder () для роботи з кодером та всіма виділеними буферами. Останній етап – це закриття вхідного та вихідного файлу та вихід із програми.

### 3.5 Пропозиція щодо розширення SDK для Video Codec

Зараз SDK дозволяє обробляти дані з вхідної відеопослідовності, але метою роботи є завантаження зображень з камер 4K. Для завантаження кадрів з камер необхідно реалізувати функції конфігурації та управління камерами. Для використовуваного пристрою mvBlueFox існує інтерфейс mvImpact Acquire SDK, який надає деякі корисні функції та доступ до цих типів камер [25].

Для того, щоб одночасно обробляти дані з двох або більше камер, процеси завантаження зображення повинні виконуватися паралельно. У випадку з цією роботою необхідно розпаралелювати кодування кадрів. Для роботи з кількома потоками була обрана бібліотека Win 32, яка надає API для багатопоточності. Врешті-решт, буде створено два потоки для однієї камери, яка матиме спільну пам'ять. Для вибраної апаратної конфігурації, доступної автору, максимальна кількість камер, які мікросхема NVENC може кодувати паралельно, дорівнює 2, отже загальна кількість потоків у програмі - 4. Два потоки використовуються для зчитування зображень з камер, а інші два кодують прочитане і збережіть їх у файл. Після заміни відеокарти у конфігурації на графічний процесор, який може одночасно обробляти більше 2 камер, кількість потоків волокон завжди перевищує кількість підключених камер.

# 4 РЕАЛІЗАЦІЯ ПРОГРАМИ РОЗШИРЕННЯ SDK VIDEO CODEC

## 4.1 Мова програмування та використовувані інструменти

У цьому розділі описано власну реалізацію розширення SDK Video Codec. Перша частина стосується поділу процесів на волокна.

CUDA підтримує кілька мов програмування – мову C ++, рекомендовану для даної роботи. Згодом його обрали найбільш підходящим варіантом, оскільки існує низка бібліотек, що підтримують виклики функцій CUDA, і це ідеальне рішення з точки зору подальшої оптимізації кодування.

Для створення GUI було обрано бібліотеку Windows Forms, яка є частиною NET Framework. Він використовується для простого створення середовища користувача за допомогою елементів керування, які використовуються в Windows. У Windows Forms можна використовувати такі мови, як C++ і візуальної Basic, але після того, як проект буде використовуватися тільки C++ оскільки це більш зручно.

Оскільки було обрано IDE MS Visual Studio 2019 IDE був обраний для забезпечення відповідальності на багатьох форумах та плагіна Nsight, що спрощує процес налагодження, профілювання та аналізу [26].

## 4.2 Поділ процесів

Як згадувалося в дизайні реалізації, WinAPI використовується для створення декількох потоків. Можна створити більше потоків одночасно, тому спочатку потрібно з'ясувати кількість підключених камер за допомогою коду рис. 4.1.

```
DeviceManager devMgr;
globvar::devCnt = devMgr.deviceCount();
if (globvar::devCnt == 0) {
    cout << "Пристрій не знайдено !" << endl; getchar();
    return 0;

}
else cout << "Кількість підключених пристроїв: "
<< globvar::devCnt << endl;
```

Рис. 4.1 Код, що показує кількість камер, підключених до комп'ютера

Зараз відомо, скільки потоків можна створити, але ця кількість все ще обмежена важкими конфігураціями комп'ютера, вираними війною. У цьому випадку максимальна кількість опрацьованих камер 2. Тепер створюються відповідні файли купольних номерів .mu.Ci. Ці файли отримали назви за комбінацією слова out, номера камери та кодека, який використовується зараз. Щойно створений файл відкривається для запису, а вказівник зберігається у векторі. Новий потік створюється шляхом виклику функції CreateThread з такими аргументами:

• L PSECURITY_ATTRIBUTES IpThreadAttributes - якщо не NULL, де Фінни , контроль може бути переданий потомству;

• SIZE_T dwStackSize - розмір ініціалізації в байтах, якщо аргумент 0, розмір за замовчуванням призначається новому потоку;

• LPTHREAD_START_ROUTINE l pStartAddress - представляє стандартні волокна ad resu;

• LPVOID lpParameter - вказівник на параметри, що передаються в потік;

• DWORD dwCreationFlags - прапори для створення потоку, наприклад, негайно запустіть або сплять, і їх потрібно запускати вручну;

• LPDWORD lpThread Id - вказівник на змінну, якій належить ідентифікатор потоку.

Функція, яку потрібно запустити в новому потоці, зазвичай призначається lpStartAddress. Він приймає один аргумент PVOID lpParams який пропускає,

навіть якщо формування волокон і мають зворотний тип DWORD (непідписаний довго). Іншою не менш важливою функцією WinAPI є WaitForMultipleObjects, яка чекає, поки всі сформовані волокна основного потоку закінчаться, а потім піде вгору. Якщо після того, як потрібно буде передати деякі параметри функції, необхідно створити величину величини для змінної. Якщо цих змінних більше, створюється клас, об'єкти якого будуть нести всі ці змінні. У випадку з цією роботою є 2 класи. Перший призначений для передачі у формуванні поточних пристроїв обробки (камер) під назвою ThreadParameter, який несе інформацію про камеру та вказівник на сам пристрій. Другий згаданий клас - NVENCTHREAD_PARAMS, який несе параметри кодування. Створення параметрів та нового потоку кодування з функцією очікування повернення потоків показано в коді рис. 4.2.

```
std::vector<NVENCTHREAD_PARAMS*> QuadroParams;

for (int i = 0; i < globvar::devCnt; i++)
{
    QuadroParams.push_back(new NVENCTHREAD_PARAMS (i, eFormat,
    cuContext, nWidth, nHeight, encodeCLIOptions, outputs[i]
    ));
}

  vSize = QuadroParams.size();
  HANDLE* hQuadro = new HANDLE[vSize];

  for (int i = 0; i < vSize; i++)
  {
      DWORD dwQuadroId;
      hQuadro[i] = CreateThread(NULL, 0,
      NvencThread, (LPVOID)QuadroParams[i], 0, &dwQuadroId);
  }

  WaitForMultipleObjects(
  static_cast<DWORD>(vSize), hQuadro, true, INFINITE);
```

Рис. 4.2. Код, що відображає параметри створення та волокно для кодування в залежності від кількості підключених камер

### 4.3 Завантаження зображень із камер

Завантаження зображень із камер виконується за допомогою бібліотеки mvImpact Acquire SDK. Спочатку потрібно з'ясувати кількість підключених камер за допомогою коду 4.1. Створюється диспетчер пристроїв, за допомогою якого буде налаштовано та керовано пристрій. Виклик функції globvar :: buffstack_init (globvar :: devCnt) виділяє простір для змінних глобальної камери для кількості підключених пристроїв. Кожен потік потребує своїх параметрів. Найважливішим параметром є диспетчер пристроїв, який є об'єктом класу DeviceManager. При призначенні параметрів також визначається розмір отриманих кадрів. Це робиться шляхом створення екземпляра ifc GenICam :: ImageFormatControl (devMgr [i]), де аргументом є диспетчер пристроїв в індексі поточно обробленої камери. Цей об'єкт несе інформацію про висоту та ширину кадрів, а також глибину, набори тощо. Необхідна висота та ширина кадру призначаються глобальним змінним і будуть використовуватися в процесі кодування.

Перед створенням потоку для кожної камери викликається функція configureDevice, яка передається менеджеру пристроїв. Призначення цієї функції - встановити необхідні параметри для завантаження зображень. Формат пікселів та наявність альфа-каналу встановлюються зокрема. Після створення потоку за початковою адресою функції WINAPI liveThread DWORD (LPVOID pData), менеджер пристрою перезаписується у змінну з переданого параметра. Спочатку перевіряється, чи пристрій відкритий, якщо ні, робиться спроба відкрити його методом open (). Якщо пристрій не вдається відкрити, відобразиться повідомлення про помилку і програма закінчиться. Потім створюється об'єкт, який несе статистику пристрою та об'єкт FunctionInterface. Цей клас містить усі основні функції для роботи з камерою. Для створення нового функціонального інтерфейсу необхідно передати конструктору вказівник на пристрій. Тепер об'єкти керування пристроєм створені для автоматичного встановлення збору зображення та витримки затвора та для встановлення кількості кадрів в секунду, зазначених у аргументах програми. Запускається функція manualStartAcquisitionIfNeeded (pDev,

fi), яка готує контролер до захоплюйте дані та починайте завантажувати зображення з камери. Після запуску почнуть створюватися запити для зображень, які будуть розміщені в черзі. Цей процес проілюстровано в коді рис. 4.3 [27].

```
requestNr = fi.imageRequestWaitFor(timeout_ms);
if (fi.isRequestNrValid(requestNr)) {
  pRequest = fi.getRequest(requestNr);
  if ((pRequest->isOK()) && ((pRequest->infoFrameNr.read()) > 0)) {
    ++cnt;
    saveImages(pRequest, pDev);
  }
  if (!(pRequest->isOK())) {
    {
      LockedScope lockedScope(globvar::g_critSect_print);
      cout << "Error: " << pRequest->requestResult.readS() << endl;
      getchar();
    }
  }
  if (fi.isRequestNrValid(lastRequestNr)) {
    fi.imageRequestUnlock(lastRequestNr);
  }
  lastRequestNr = requestNr;
  fi.imageRequestSingle();
}
```

Рис. 4.3. Код, що показує завантаження зображень з камери

По-перше, потрібно зробити запит на отримання зображення з камери, це робиться за допомогою функції imageRequestSingle (). Першою вимогою є згадана вище функція тригера. Рядок 1 чекає обробки зображень. Цей метод проходить час, після якого завантаження не вдасться, оскільки запит обробляється занадто довго. Другий рядок перевіряє правильність результату запиту. Згодом перевіряється результат цієї перевірки, якщо все в порядку, результат надсилається для збереження. Якщо зображення не в порядку, відобразиться повідомлення про помилку. Перед вступом останній завантажений кадр потрібно розблокувати, щоб новий кадр можна було заблокувати.

Іншою важливою функцією є void saveImages, аргументами якої є:

• Запит * - вказівник на результат запиту на повернення кадру

• Пристрій * - вказівник на підключений пристрій

Ця функція вставляє зображення в буфер оперативної пам'яті, звідки кодер їх витягує. Цей процес збереження слайдів показаний у спрощеному коді рис. 4.4.

```
for (int i = 0; i < globvar::devCnt; i++)
{
  if (device == globvar::buffstack.camnumb[i])
  {
    size_t alloc_mem = pRequest->imageWidth.read() *
    pRequest->imageHeight.read() * globvar::channels;
    unsigned char* pTexSysBuf = (unsigned char*)malloc(alloc_mem);
    memcpy(pTexSysBuf, pRequest->imageData.read(), alloc_mem);
    globvar::buffstack.tex_stacks[i].push(pTexSysBuf);
  }
}
```

Рис. 4.4. Код відображення зображень в чергу в пам'яті RAM

Програма закінчується після закінчення встановленого користувачем обмеження часу. З точки зору реалізації, завантаження кадру припиняється, якщо кількість уже завантажених кадрів дорівнює добутку вхідної довжини та значення кількості кадрів в секунду. Коли програму зупинено, прапорець потоку встановлюється як завершений і викликається вручну StopSacquisitionIf Needed функція (pDev, fi), яка піклується про зупинку отримання зображень контролером камери.

### 4.4 Процес кодування

Для потоку, що впорядковує кодування, потрібно передати параметри, відповідно до яких створюється екземпляр кодера. Призначення параметрів і створення нового потоку для кодування показано в коді 4.2. Після створення потоку вводиться локальна змінна, яка несе вказівник на параметри, передані

функції. Решта коду функції обгортається блоком try catch. Тепер об'єкт кодера створюється за допомогою виклику конструктора з конкретними аргументами пристрою, розмірами кадру та форматом вхідних даних. Після успішного створення кодера він буде ініціалізований. Під час ініціалізації встановлюються параметри, які користувач ввів на початку програми. Для інших, які не будуть використовуватися для тестування, встановлено значення за замовчуванням. Метод CreateEncoder створює кодер із заданими параметрами. Тепер цикл запущений і триває доти, доки прапорець нитки не буде встановлений як кінець. У цьому циклі запускається інший цикл, який перевіряє розмір черги, на яку в другому потоці записуються прочитані зображення з камери. Якщо це число більше 0, починається кодування. Наступний процес показаний у коді рис. 4.5.

```
if (nFrame == length) {
    pNvencThreadParams->terminateThread();
}
nFrame++;
uint8_t* pTexSysBuf = new uint8_t[nSize]();
CopyFrameToText3chann(globvar::buffstack.tex_stacks
[pNvencThreadParams->count].front(), pTexSysBuf, width, height);
unique_ptr<uint8_t[]> pHostFrame(pTexSysBuf);

const NvEncInputFrame* encoderInputFrame = pEnc->GetNextInputFrame();
// у цьому розділі залежить від обраного проекту
// скопіювати кадр на вибраний пристрій

pEnc->EncodeFrame(vPacket);

for (std::vector<uint8_t>& packet : vPacket)
{ std::endl;
    pNvencThreadParams->fpOut->write(reinterpret_cast<char*>(
    packet.data()), packet.size());
}

auto lastFrame =
globvar::buffstack.tex_stacks[pNvencThreadParams->count].front();
globvar::buffstack.tex_stacks[pNvencThreadParams->count].pop();
delete[] lastFrame;
```

Рис. 4.5. Код, що відображає процес кодування

Якщо кількість закодованих кадрів дорівнює добутку введеної довжини та значення кількості кадрів в секунду, тоді прапор встановлює завершений прапор. Функція CopyFrameToText3chann викликається з аргументами:

• кадр без знака char * - вказівник на вхідний кадр, у цьому випадку перший кадр у черзі збережених кадрів із камери;

• uint8_t * pRGB - вказівник на отриманий кадр;

• int width - ширина кадру;

• int height - висота кадру.

Ця функція піклується про розширення кадру порожнім альфа-каналом, оскільки кодер очікує введення кадру з чотирма каналами. Її тіло видно в коді 4.6. З ESULTS ця функція створює розумний вказівник, який стосується розподілу та вивільнення пам'яті лише після догляду. Метод називається кодером GetNextInputFrame, який переміщується в пам’яті ра розміру кадру і повертає покажчик на наступний слайд. Потім дані кадру необхідно скопіювати у вхідний буфер кодера. Потім викликається метод кодування EncodeFrame, куди передається вказівник, куди дані отриманого кадру копіюються після стиснення. Якщо цей показник є деяким, дані запускаються для pis у вихідному файлі. Нарешті, пам'ять вихідного нестислого зображення з черги повинна бути вивільнена.

Коли кодування завершено, викликається метод кодування EndEncode, який гарантує, що всі черги кодера спорожняться, щоб їх можна було звільнити. Ця функція необхідна для повного звільнення кодера методом DestroyEncoder. Це звільняє всі виділені ресурси та завершує сеанс кодування. Вихідний файл перед ним закривається. Потім сама нитка закінчується [28].

```
int channels = 3;
int dwStride = width * 4;
unsigned char* ptr_in = frame;

for (int j = 0; j < height; j++)
{
    unsigned char* ptr_in_row = ptr_in + j * width * channels;
    unsigned char* p4 = pRGB + j * (dwStride);

    for (int i = 0; i < width; i++)
    {
        p4[i * 4 + 0] = *(ptr_in_row++);
        p4[i * 4 + 1] = *(ptr_in_row++);
        p4[i * 4 + 2] = *(ptr_in_row++);
        p4[i * 4 + 3] = 0;

    }
}
```

Рис. 4.6. Код, який показує розширення кадру з допомогою нульового альфа - каналу

### 4.5 Впровадження графічного інтерфейсу

Для реалізації графічного інтерфейсу використовували Windows Forms. Він був обраний, оскільки це бібліотека з відкритим кодом, яка є частиною .NET Framework. Це платформа для написання клієнтських настільних додатків. Зовнішній вигляд заявки наведено на рис. 4.7. Для реалізації роботи були використані такі елементи:

• Форма - основний елемент відображення. Вікно, що відображає всі інші елементи.

• RadioButton - можна вибрати єдиний параметр у групі елементів radioButton.

• Етикетка - Зазвичай відображається текстовий опис або інструкції для іншого елемента управління або форми.

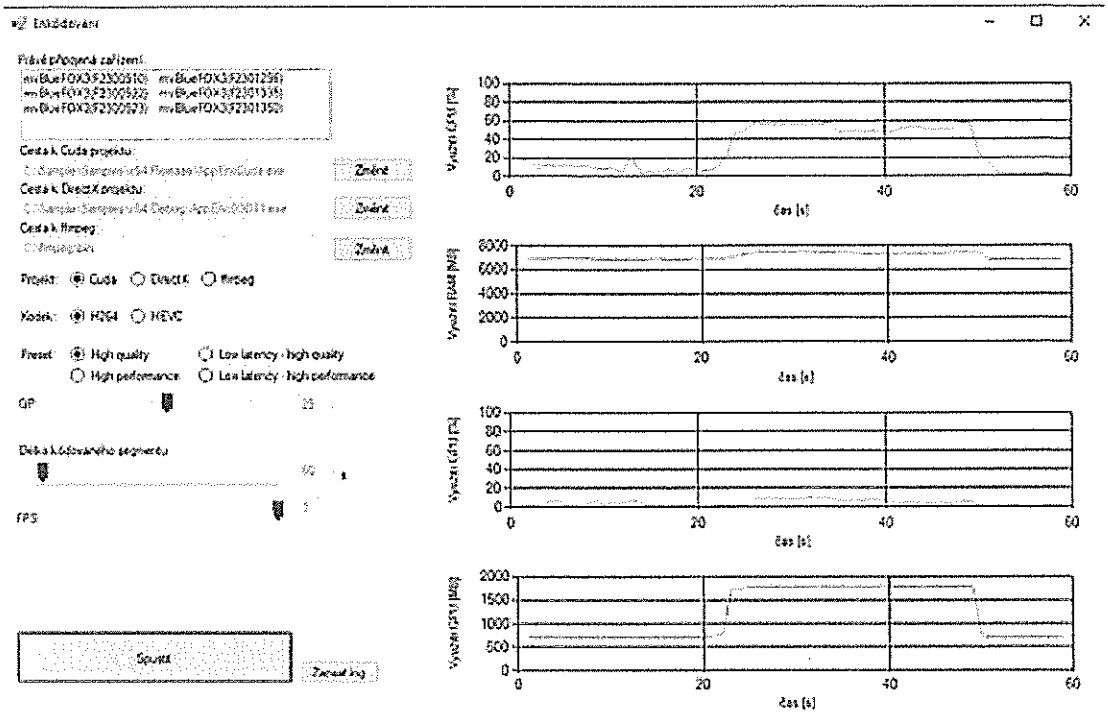


Рис. 4.7 Приклад створеного графічного інтерфейсу

• TextBox - Відображає значення, введене користувачем. Він може отримувати маски та відображати паролі, може бути лише для читання.

• NumericUpDown - Відображає числове значення. Ви можете додавати та видаляти значення за допомогою стрілок вгору та вниз.

• ProgressBar - має функцію встановлення значення з набору значень. Він регулюється перетягуванням повзунка між максимальним та мінімальним.

• ListView - Відображає елементи списку або підсписку. Він може відображати різні елементи в кожному рядку або створювати стовпці.

• Кнопка - представляє кнопку, виконує дію клацання або натискання клавіші.

• GroupBox - в основному використовується для збору декількох radioButtons, оскільки він може містити рівно одну вибрану radioButton. Зв'язані елементи лінією.

Ці графічні елементи підключені до керуючої частини, яку можна встановити за своїми властивостями або запрограмувати вручну. Після запуску програми всі компоненти ініціалізуються. Згодом створюються нові потоки для

моніторингу продуктивності процесора, оперативної пам'яті та продуктивності графічного процесора. Необхідно з'ясувати кількість камер і максимально можливе значення кадрів в секунду. Це робиться шляхом запуску нового процесу, який запускає програму з прапором -maxfps через командний рядок, який виводить назву камери, її серійний номер та максимально можливе значення кадрів в секунду. Додаток зчитує цей рядок і зчитує з нього доступну інформацію, яку використовує для обмеження максимального значення ProgressBar для встановлення кількості кадрів в секунду. Призначає імена камер ListView та відображає їх у вигляді рядків.

Клас PerformanceCounter використовується для процесора та оперативної пам'яті для отримання значень графіків. Щоб отримати значення з графічного процесора, потрібно було встановити nug-пакет NvAPIWrapper. Клас PhysicalGPU, який представляє підключену на даний момент графічну карту, містить інформацію про поточне використання. Графіки оновлюються за допомогою методу UpdateGPUChart, який додає нове значення у відображене поле. Весь процес показаний у коді рис. 4.8.

Користувач встановлює всі параметри програми, з якою він хоче запустити цільову програму. Потім він натискає кнопку запуску, яка завантажує всі значення елементів керування та запускає програму. Це робиться шляхом створення нового процесу, в якому відкривається командний рядок і там вводиться шлях до файлу, залежно від того, який проект було обрано. Згодом аргументи, які були підсумовані з елементів керування, додаються до шляху до файлу. Процес наведено у зразку коду рис. 4.9. Весь процес контролюється, і за 3 секунди до його початку записуються всі прочитані значення графіків, які поступово розділяються крапкою з комою. Після натискання другої кнопки «Записати журнал» буде створений файл із іменами параметрів у форматі csv, з якого можна легко створити графіки.

```
        private double[] GPUArray = new double[60];
        private void UpdateGPUChart()
        {
            GPUChart.Series["Series1"].Points.Clear();
            for (int i = 0; i < GPUArray.Length - 1; ++i)
            {
                GPUChart.Series["Series1"].Points.AddY(GPUArray[i]);
            }
        }

        private void getPerformanceGPUCounters()
        {
            var gpu = PhysicalGPU.GetPhysicalGPUs();
            while (true)
    {
        var usage = gpu[0].UsageInformation.GPU.Percentage;
        GPUArray[GPUArray.Length - 1] = usage;

        Array.Copy(GPUArray, 1, GPUArray, 0, GPUArray.Length - 1);

        if (GPUChart.IsHandleCreated)
        {
            this.Invoke((MethodInvoker)delegate { UpdateGPUChart(); });
        }
        Thread.Sleep(1000);
    }
}
```

Рис. 4.8 Код, що показує зчитування значень використання графічного процесора та оновлення графіку

```
string genArgs = codec + preset + fps + QP + length;
isLogging = true;
InitTimer();
Thread.Sleep(3000);
try
{
    runProg.StartInfo.FileName = pathToFile;
    runProg.StartInfo.Arguments = genArgs;
    runProg.StartInfo.CreateNoWindow = true;
    runProg.Start();
}
catch (Exception ex)
{
    MessageBox.Show("Nebylo možné spustit program" + ex);
}
```

Рис. 4.9. Зразок коду для створення нового процесу заявки

# 5 ТЕСТУВАННЯ РОЗШИРЕННЯ SDK VIDEO CODEC

## 5.1 Порівняння результатів аналізу окремих проектів

Тестування проводили на наборі тестів, показаному в таблиці 5.2. Під час тестування використовувались камери MvBlueFOX3. Усі тести проводились із графічною картою №1, якщо інше не вказано для тесту. Параметри цієї камери наведені в таблиці 5.1. Всі тести для двох камер проводились на камерах з дуже подібною сценою. Коли використовували кілька камер, захоплені сцени були різними. Усі тести виконувались у режимі налагодження. Тести поділяються на ефективні та якісні. Продуктивність включає порівняння заданих параметрів для завантаження графічного процесора, процесора та оперативної пам'яті. Крім того, тест на середню тривалість частини коду та тест на максимальне навантаження кодера в режимі реального часу. У тестах якості спостерігатимуться втрати на стиск закодованого зображення порівняно з оригіналом.

Коли обидва проекти мають однакові вхідні файли, можна порівняти швидкість кодування. Тестування проводили на наборі тестів 5.2. Як вхідні файли використовувались дві послідовності введення, параметри яких наведені в таблиці 5.3. Порівняння результатів було зафіксовано в таблиці 5.4. У таблиці записано значення з п'ятнадцяти запусків проекту за тими ж параметрами та входами. Стандартне відхилення розраховувалося з отриманої послідовності. Графіки спостерігалися в диспетчері завдань на маркер продуктивності. На рис. 5.1 і 5.2 показані графіки, які були створені при кодуванні вихідного файлу з роздільною здатністю 3840x2160, що називається бджолами.

На рис. 5.1 показані графіки, що порівнюють обчислювальну потужність GPU та розподіл пам'яті для проекту AppEncCuda. У верхньому правому куті є графік, що показує навантаження мікросхеми NVENC. У верхньому лівому куті є графік, що показує використання обчислювальної потужності CUDA.

Наступні два графіки стосуються розподілу пам'яті. Перший показує виділення пам'яті, зарезервованої для графічного процесора (GPU (buery)), а

другий - використання спільної пам'яті графічного процесора GPU.

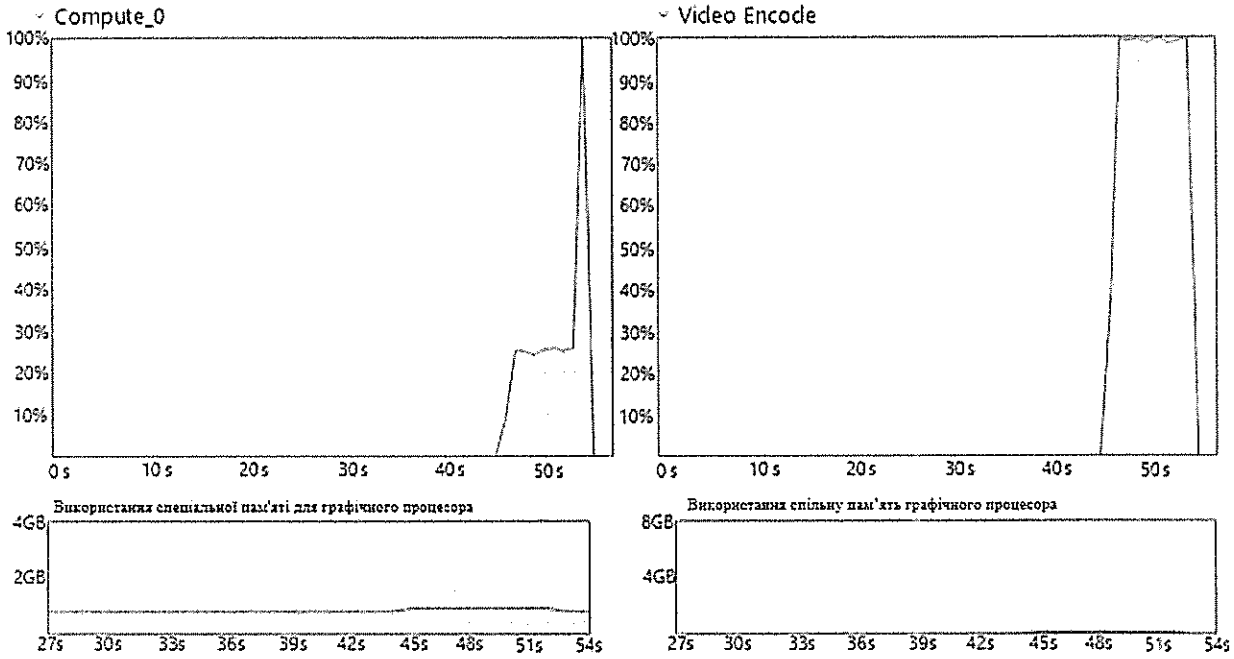



Рис. 5.1. Графік із використанням потужності CUDA та діаграми, де зображено дискримінаційне використання мікросхеми NVENC

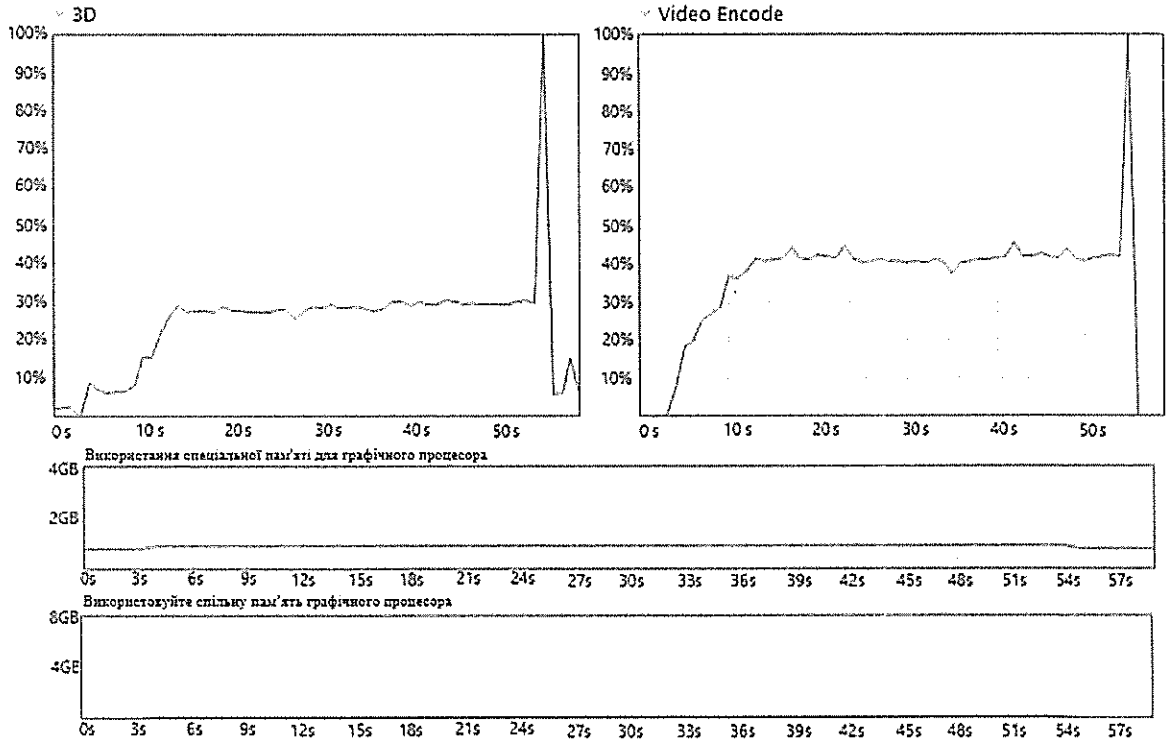



Рис. 5.2. Графіки з використанням потужності DirectX і діаграми, де зображено дискримінаційне використання мікросхеми NVENC

Таблиця 5.1.

Апаратні ресурси, що використовуються для запуску кодера

| MvBlueFox3 | |
|---|---|
| Модель | 1100 |
| Варіант | C. |
| Відзнака | 3856 x 2764 |
| MPixel | 10.7 |
| Макс. частота кадрів | 7,3 Гц |

На рис. 5.2 приведено графік, що порівнює обчислювальну потужність GPU та виділення пам'яті для проекту AppEncD3D11.

Таблиця 5.2.

Апаратні ресурси, що використовуються для запуску кодера

| Тестовий набір | |
|---|---|
| Операційна система | ОС для Windows 10, 64 біт |
| Процесор | Процесор Intel (R) Core (TM) i5-8400 на 2,80 ГГц |
| Пам’ять | 16 ГБ DDR4, 3000 МГц |
| Відеокарта 1 | NVIDIA GeForce GTX 1050 Ti, Pascal, 4 ГБ GDDR5 |
| Відеокарта 2 | NVIDIA Quadro P2000, 5 ГБ GDDR5 |
| Диск | SSD-накопичувач Intel 600p M.2 256GB |

Таблиця 5.3.

Таблиця містить деякі атрибути та опис вихідних відеопослідовностей перед стисненням

| Ім'я | Відзнака | FPS | Опис |
|---|---|---|---|
| зайчик | 1280x720 | 30 | анімована казка, динамічні зміни, космічні зміни |
| бджола | 3840x2160 | 24 | бджолиний рій, рух бджіл |

NVENC. У верхньому лівому куті є графік, що показує обчислювальну

потужність DirectX. Наступні два графіки стосуються розподілу пам'яті. Перший показує виділення виділеної пам'яті GPU (буфери), а другий - спільної пам'яті GPU.

Таблиця 5.4.

Результати кодування відео в обох проектах із використанням стандартів H.264 із порівнянням у H.265.

| Відео | зайчик | | бджоли | |
|---|---|---|---|---|
| Відзнака | 1280x920 | | 3840x2160 | |
| Початковий розмір для CUDA | 5 765 850 кБ | | 8 723 700 кБ | |
| Кодек | H.264 | H.265 | H.264 | H.265 |
| Кінцевий розмір бітового потоку | 13610 кБ | 14259 кБ | 29 195 кБ | 35 111 кБ |
| Отриманий час кодування | Δ 873,62 ± 5,86 мс | Δ 511,9 ± 5,62 мс | 7 436,55 ± 13 мс | 7357,05 ± 74,5 мс |
| Середній час для одного кадру | 1,83 ± 0 мс | 1,29 ± 0,12 мс | 10,94 ± 0,24 мс | 10,56 ± 0,29 мс |
| Початковий розмір для DirectX | 10912645 кБ | | 16 654 214 кБ | |
| Кодек | H.264 | H.265 | H.264 | H.265 |
| Кінцевий розмір бітового потоку | 14 511 кБ | 15 294 кБ | 31 005 кБ | 36 215 кБ |
| Отриманий час кодування | 35 504 ± 557 мс | 35441 ± 354,7 мс | 51 546,33 ± 530,4 мс | 51784 ± 197 мс |
| Середній час для одного кадру | 8,31 ± 0,129 мс | 8,3 ± 0,07 мс | 71,79 ± 0,74 мс | 72,12 ± 0,274 мс |

Виходячи з таблиці та графіків, видно, що проект, в якому використовувалася CUDA, виконував кодування за менший час, ніж проект, що

використовував DirectX. Кожен із проектів із цієї перевіреної попередньої версії SDK Video Codec повинен мати інший формат вхідних даних. Для проекту, що використовує CUDA, вхідним форматом є YUV. Для проекту, що використовує DirectX, вхідним форматом був каталог із зображеннями BMP. Як результат, розмір вхідних даних для DirectX майже вдвічі перевищує розмір CUDA. Це також уточнює загальну тривалість кодування, оскільки проект DirectX повинен був завантажувати вхідні дані більш складним способом. Кожне зображення потрібно було завантажувати окремо від файлу, тому чіп NVENC використовувався при 40% загальної потужності. Проект, що використовує CUDA, лише відкривав файл на початку кодування, а весь час прокручував і зчитував дані в ньому. Це пояснює використання мікросхеми NVENC до максимального значення. Розподіл пам'яті був відносно подібним як для CUDA, так і для DirectX, як і розмір отриманих файлів.

## 5.2 Тести продуктивності

Практичні тести продуктивності використовуються для визначення меж програми та того, як її робота впливає на систему. Найчастіше вимірювані величини – це пропускна здатність, пам’ять, час відгуку, продуктивність процесора. У тестах, розроблених для цієї роботи, найважливішими аспектами є продуктивність процесора, продуктивність графічного процесора, використання оперативної пам'яті та пам'яті графічного процесора.

*Використання процесора, графічного процесора та оперативної пам'яті*

Для моніторингу продуктивності процесора, графічного процесора і оперативної пам'яті використовували функції зйомки, що діють за допомогою навантаження. Код, що показує отримання цих даних та їх імпорт на графік, показаний у прикладі 4.7. Для отримання значень центрального процесора і оперативної пам'яті використовувалися класи об'єктів за formanceCounter. Пакет NvAPIWrapper був використаний для отримання значень графічного процесора. Після запуску програми із заданими параметрами отримані значення будуть

завантажені в локальну змінну. Ці значення є крапкою з комою після останнього викиду, що один підряд є розривом рядка тексту. Дотримуючись цих правил, можна легко досягти однієї реєстрації. Після натискання кнопки для запису журналу створюється файл із іменем, який складається із введених параметрів, розділених дефісом та суфіксом csv. Коли відкривається цей файл у програмі Excel від Microsoft, створюються 4 стовпці та відповідна кількість рядків зі значеннями. Так як значення RAM пам'яті і GPU вичитав початкове значення пам'яті, які використовують систему без починає Ене Appli катіона. Значення продуктивності процесора та графічного процесора були в порядку менших одиниць до запуску програми, і тому ці значення зберігалися. Тестування проводили за кількома комбінаціями параметрів.

Моніторинг продуктивності кодування в режимі реального часу в залежності від попередньо встановлених параметрів.

У цьому тесті ефективність системи під час кодування контролювалася залежно від заданого параметра. Таблиця доступних пресетів з їх описом наведена тут 3.1. Тест запустив додаток із використанням CUDA та DirectX із кодеком H.264. Нарешті, було проведено порівняння з результатами пресета високої продуктивності для кодека HEVC. На обох рис. 5.3 графіках, які порівнюють продуктивність системи при кодуванні з використанням CUDA (вище) і DirectX (нижче) для попереднього встановлення High Performance. Діаграми для порівняння використовують живу пам’ять на рис. 5.4. Порівняння інших пресетів з кодеком H.264 наведено в додатку.

*Моніторинг продуктивності системи під час кодування в реальному часі залежно від параметра QP*

Параметр QP визначає якість кодування. Чим вище значення QP, тим менший файл і нижча якість. Для цього тесту для вхідних параметрів було встановлено проект, який використовує DirectX для кодування, а кодек - H.264. Виміряно три значення QP: 1, 22 і 50. На рисунках зображені графіки відповідно до набору QP, де QP = 1 знаходиться вгорі, QP = 22 посередині і QP = 50 внизу. граничні значення QP 1 і 50.

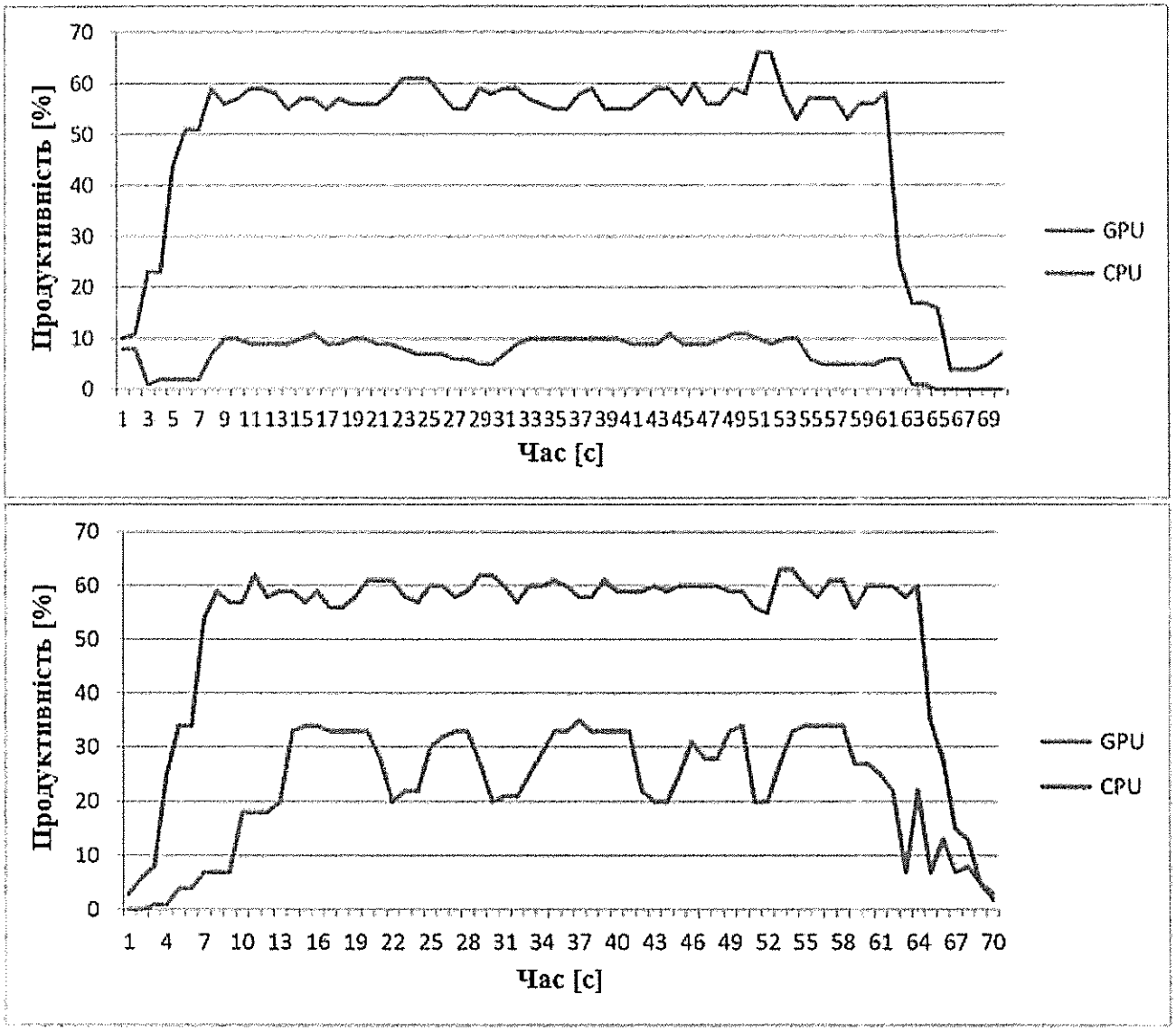



Рис. 5.3. Графіки продуктивності процесора та графічного процесора при кодуванні до H.264. Вгорі графік для CUDA, а внизу для DirectX

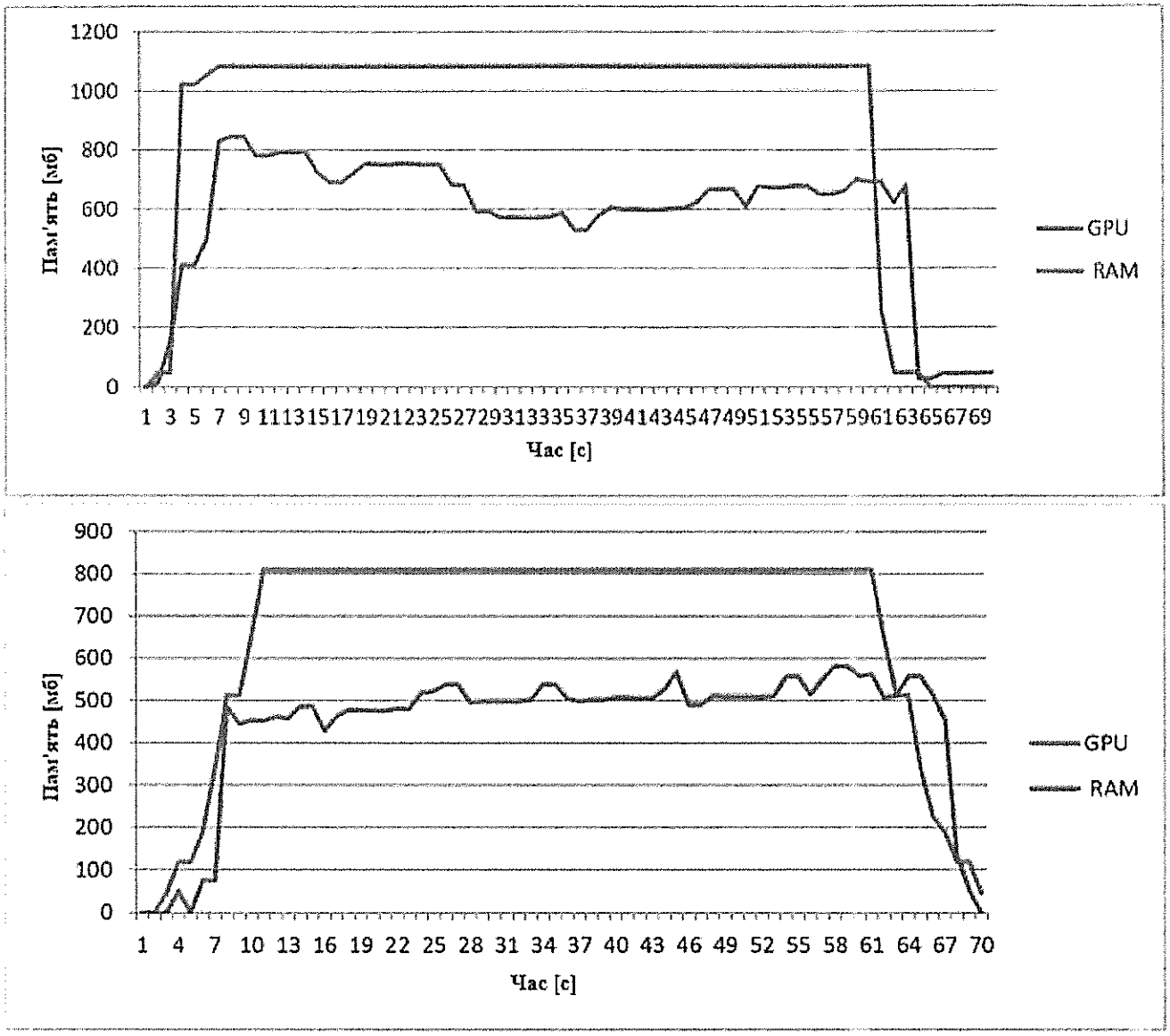



Рис. 5.4. Графіки оперативної та графічної процесорів при кодуванні до H.264. Вгорі графік для CUDA, а внизу – для DirectX. Графік пов'язаний із пресетом високої продуктивності

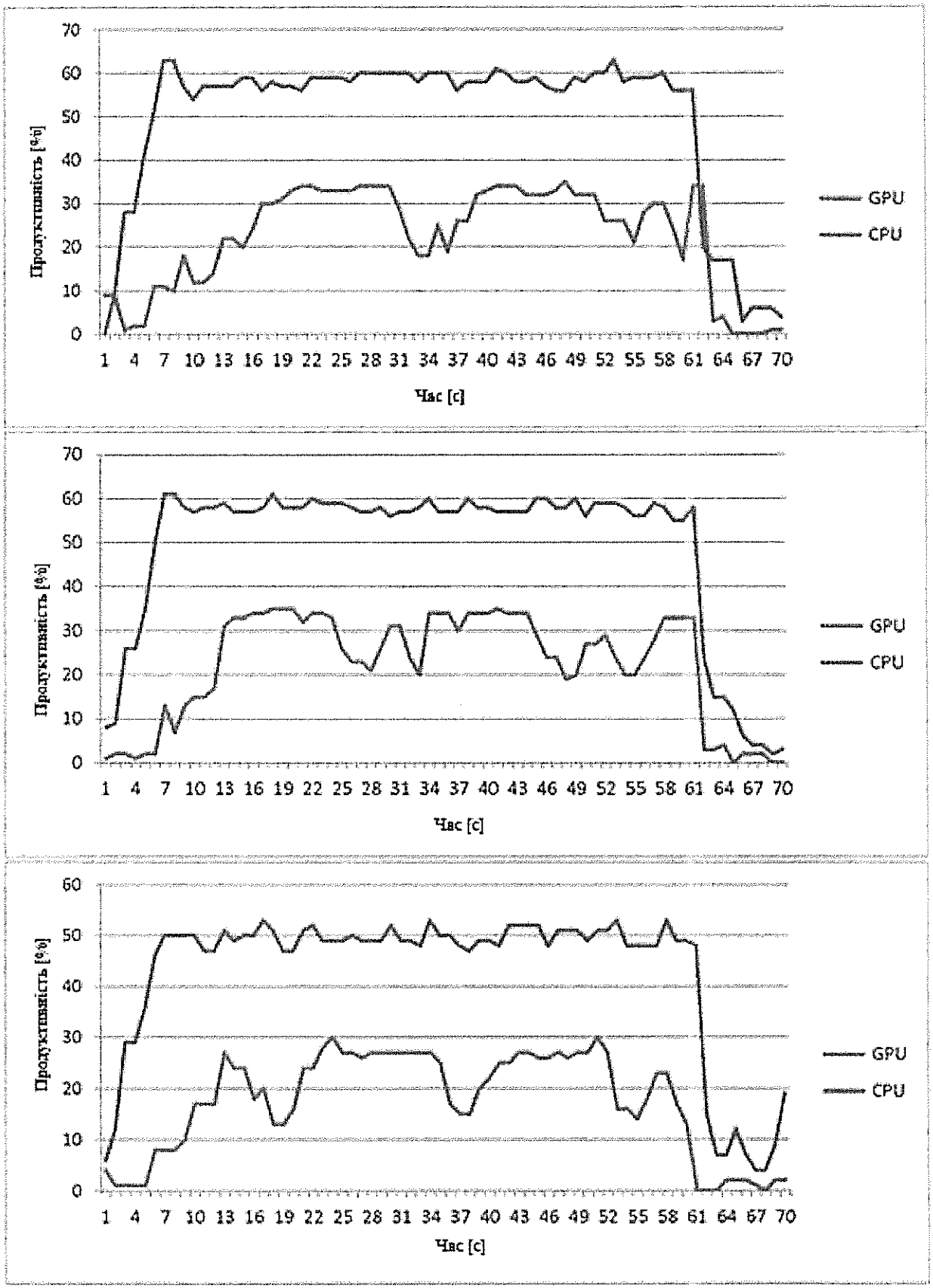



Рис. 5.5. Графіки роботи процесора та графічного процесора під час кодування залежно від параметра QP. Вгорі графік для QP = 1, посередині QP = 22 і нижче QP = 50

*Середній час оцінки частини коду*

Для цього тесту автор обрав ту частину коду, де дані зображення копіюються в буфер певного пристрою (DirectX та CUDA) та процес кодування. Ця частина була обрана, оскільки передбачається, що саме тут буде найбільша можлива різниця в часі обробки.

Для тесту була використана бібліотека std :: chrono1, яка використовується для відстеження часу з різним ступенем точності. Клас high_resolution_clock, який представляє годинник з найменшим можливим періодом тикання, використовується для запису часу, коли частина коду почала виконуватися. Бібліотека тривалості була використана для обчислення тривалості коду. Об'єкт у цьому класі зберігає інформацію про тривалість однієї години тиків та кількість кліщів. Бібліотека також надає функцію для перетворення цього об'єкта у вибрані одиниці часу типу int. Виконання тесту можна побачити в коді рис. 5.6.

```
vector<float> values;
while (!pNvencThreadParams->terminated()){
        ...
        auto start = high_resolution_clock::now();
        // měřená část kódu
        auto stop = high_resolution_clock::now();
        auto duration = duration_cast<microseconds>(stop - start);
        if (nFrame >= 0 && nFrame <= 500)
            values.push_back(duration.count());
        ...
}
float sum = 0, mean = 0, variance = 0, standardDeviation = 0;
for (int i = 0; i < values.size(); ++i)
{
    sum += values[i];
}
mean = sum / values.size();
for (i = 0; i < values.size(); ++i)
    standardDeviation += pow(values[i] - mean, 2);
```

Рис. 5.6 Код, що показує тест для обчислення часу виконання коду

# ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Презентація по магістерській роботі на тему:

# Методи стиснення потокових відеоданих на основі сучасних відеокодеків

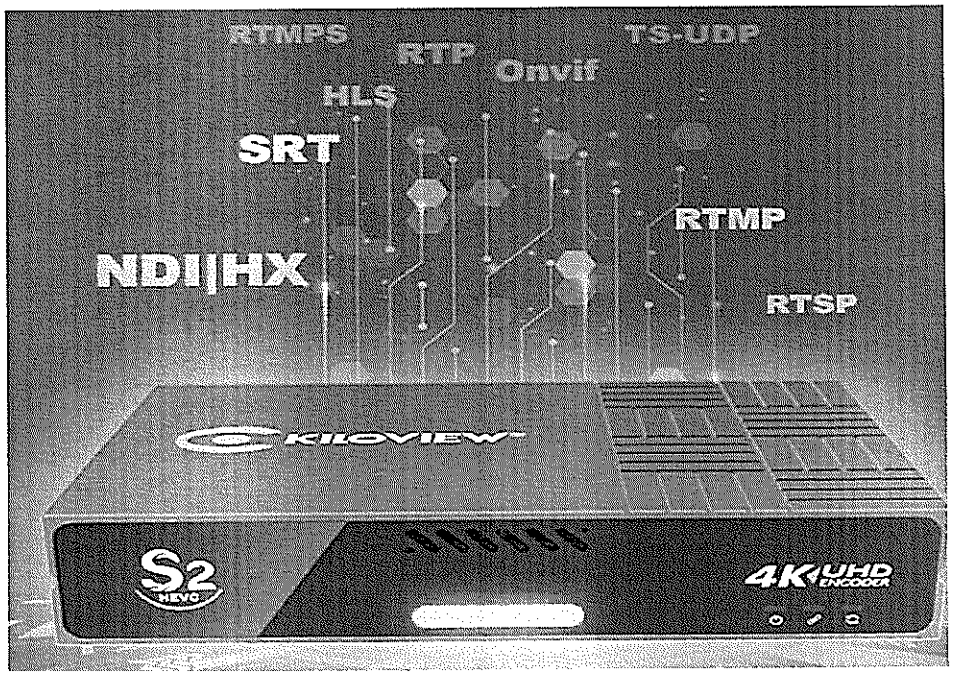
Студента
Лиценка Юрія Вадимовича

# Мета роботи

***Об'єктом дослідження*** є процес кодування відеоданих.

***Предметом дослідження*** є методи кодування 4k-відео в реальному часі.

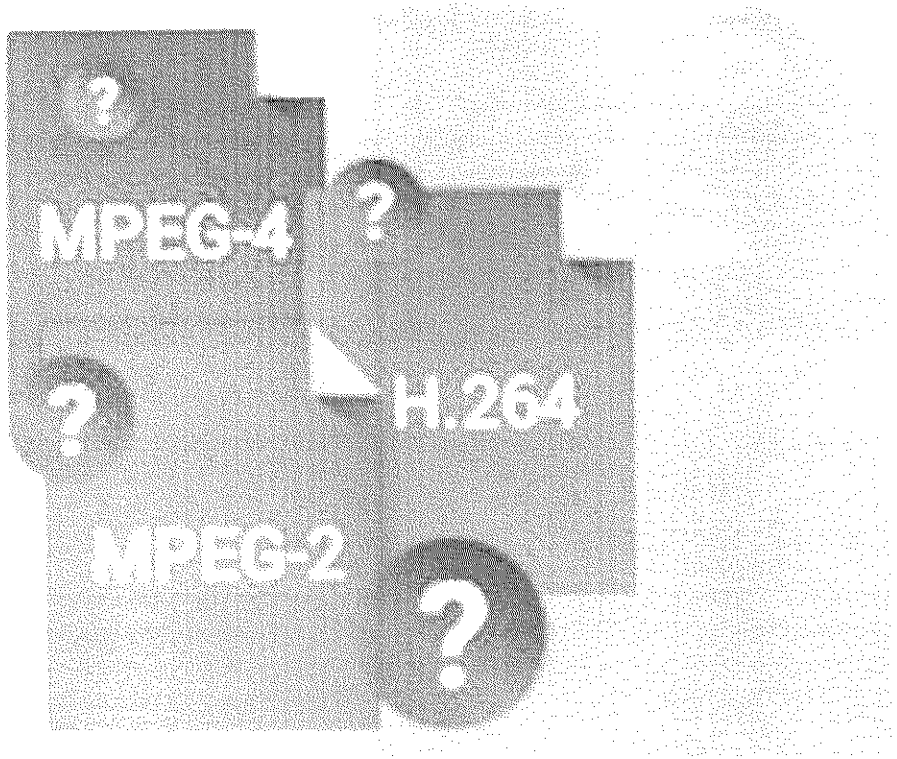
***Мета роботи.*** Розробка власного програмного забезпечення кодування 4k-відео в реальному часі на основі SDK Video Codec.



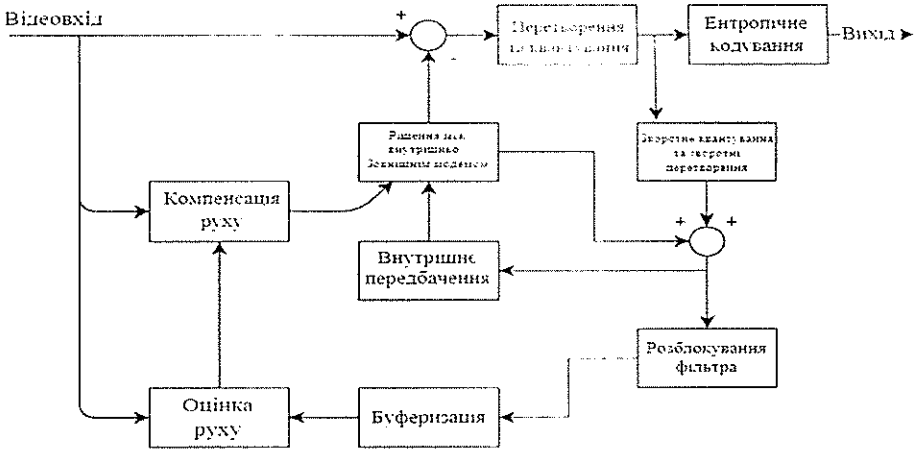


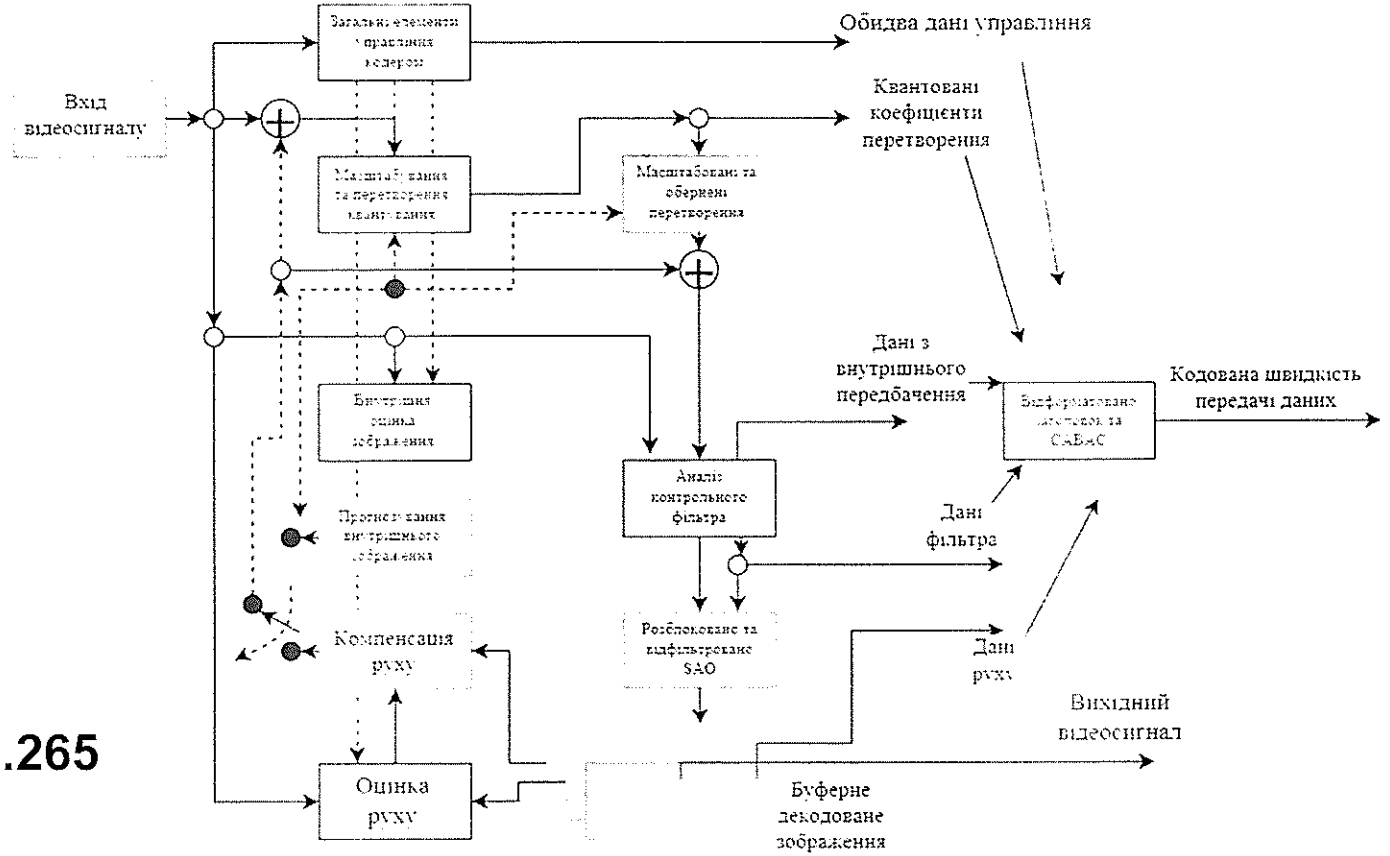
2

# Процес відтворення адитивних матеріалів, що використовується у відео



# Схема кодера H.264 і H.265

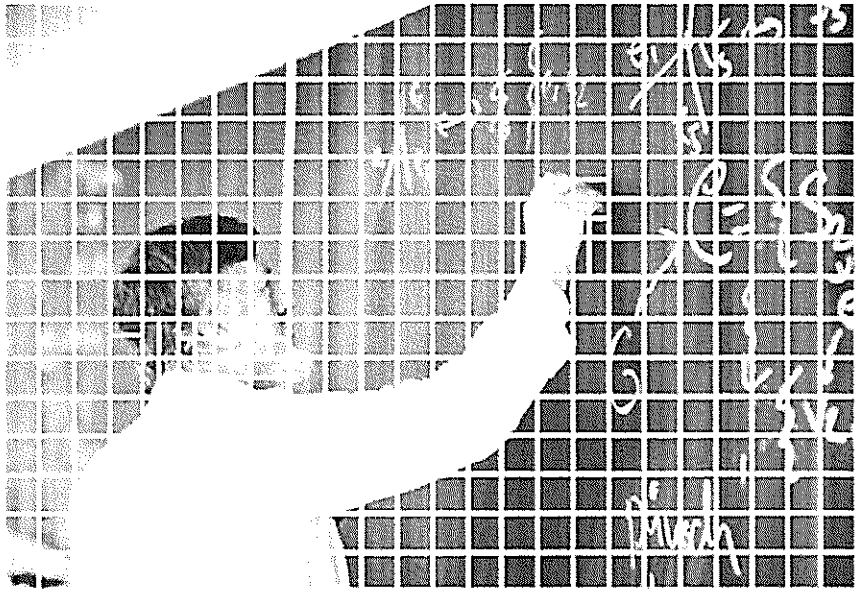


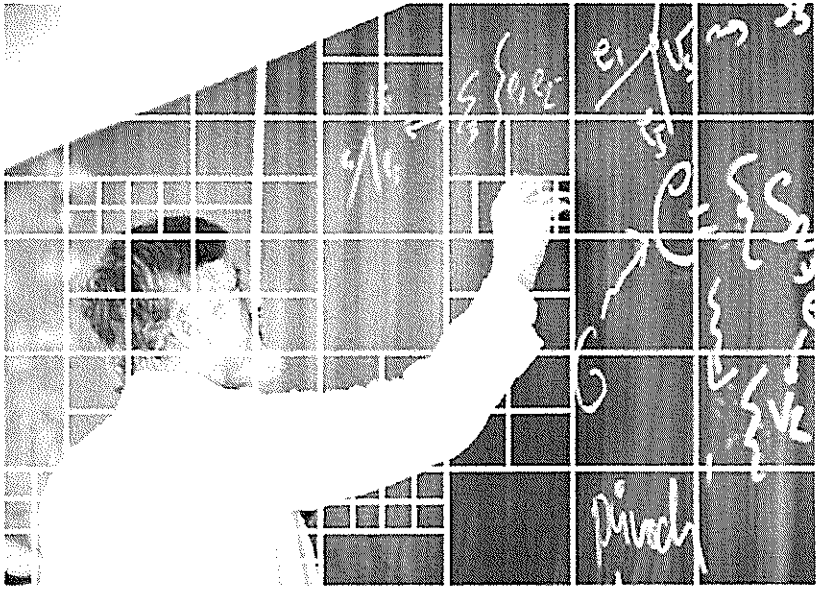

H.265

# Наочна різниця між H.264 і H.265

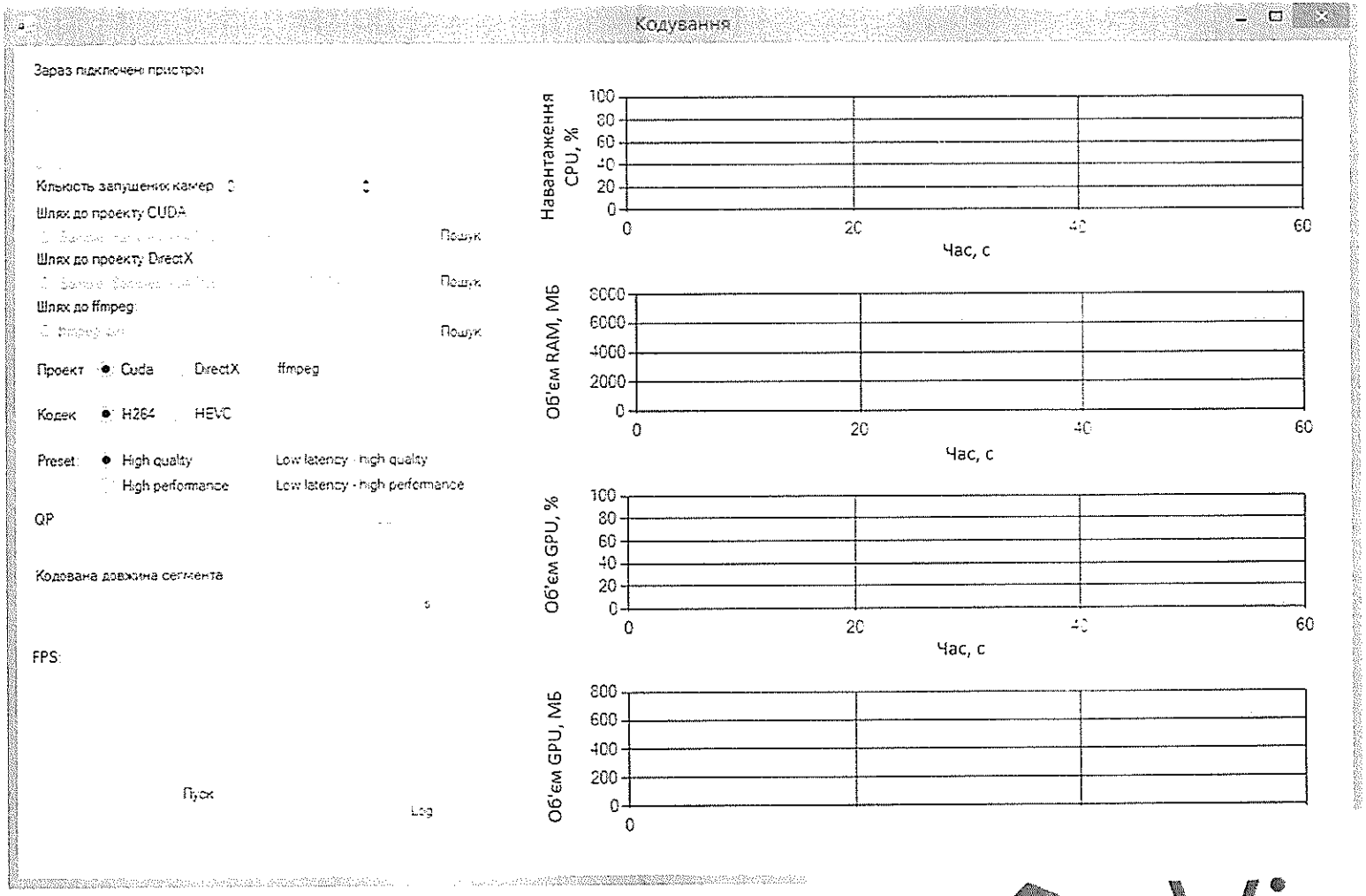
H.264



H.265

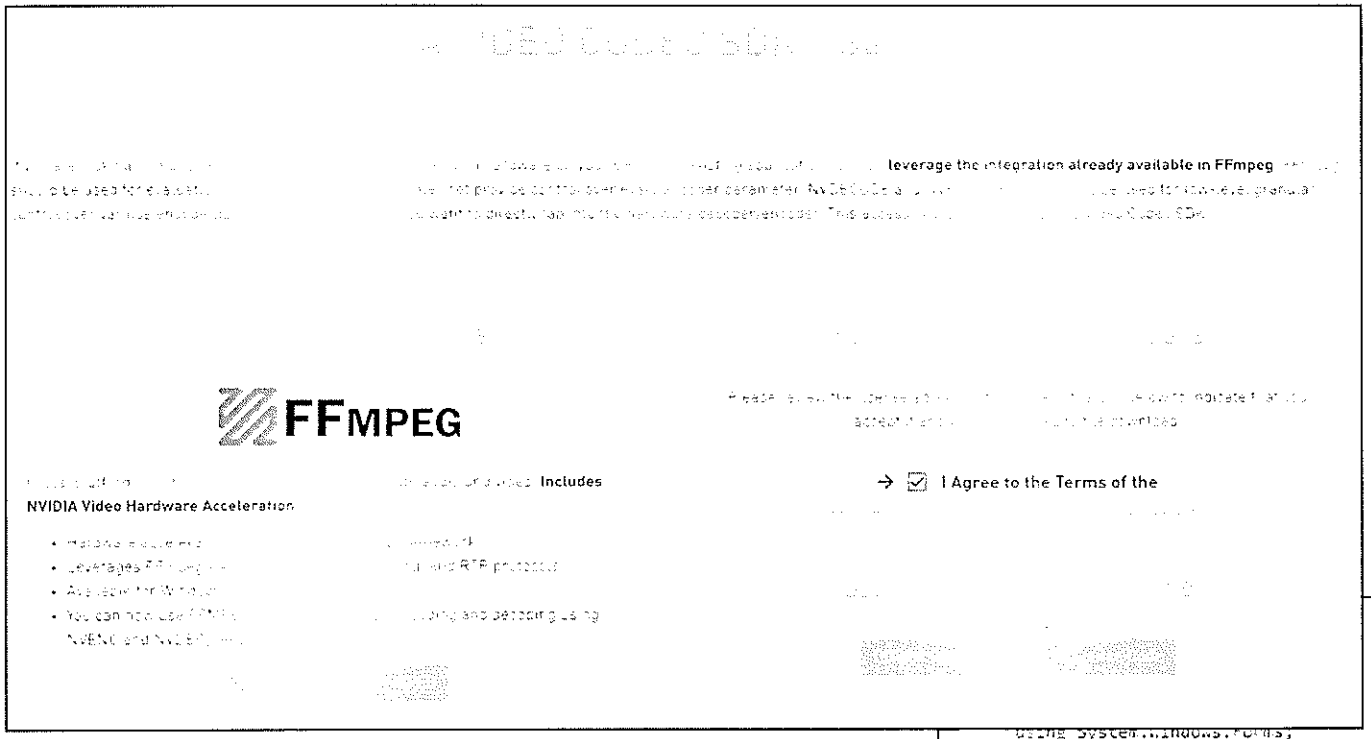


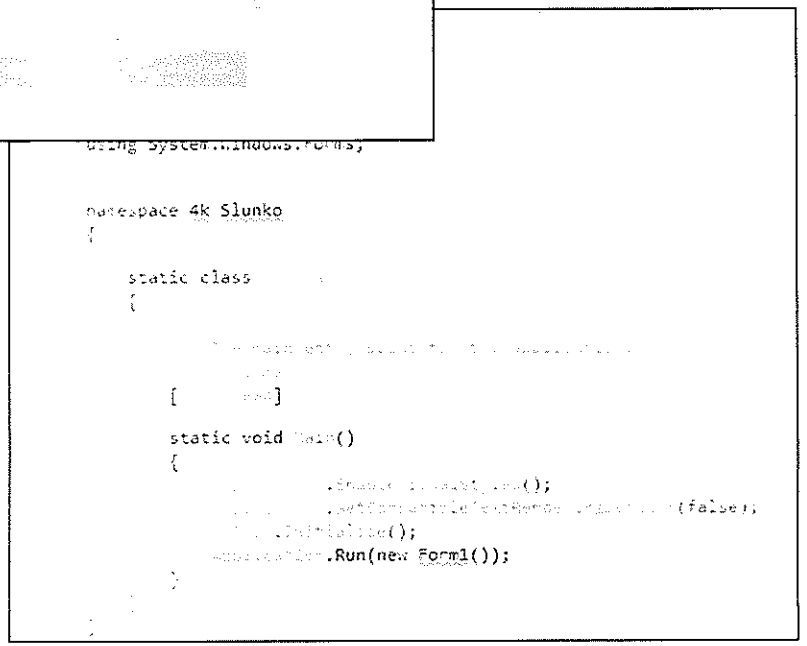
# Розроблений графічний інтерфейс




Visual Studio

# Використані та розроблені самостійно програмні коди



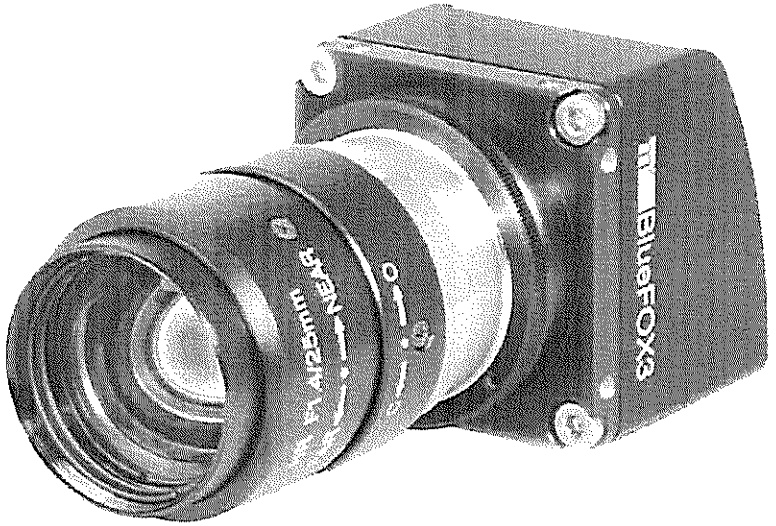




Частина коду

# Апаратне забезпечення тестування розробленого ПЗ

Апаратні ресурси, що використовуються для запуску кодера

| Тестовий набір | |
|---|---|
| Операційна система | ОС для Windows 10, 64 біт |
| Процессор | Процесор Intel (R) Core (TM) i5-8400 на 2,80 ГГц |
| Пам'ять | 16 ГБ DDR4, 3000 МГц |
| Відеокарта 1 | NVIDIA GeForce GTX 1050 Ti, Pascal, 4 ГБ GDDR5 |
| Відеокарта 2 | NVIDIA Quadro P2000, 5 ГБ GDDR5 |
| Диск | SSD-накопичувач Intel 600p M.2 256GB |



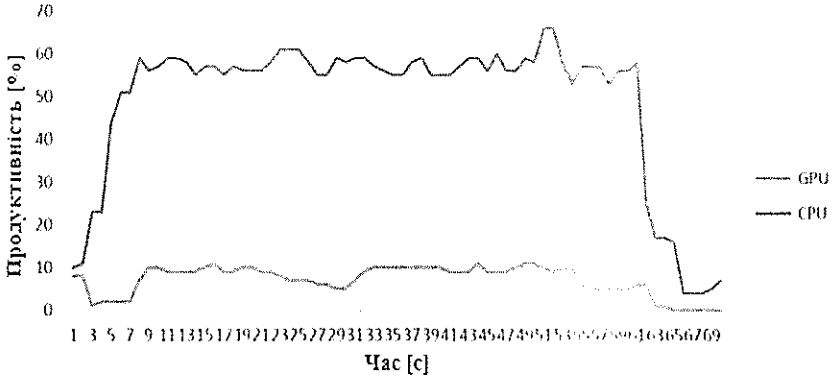
# Результати із використанням стандартів H.264 і H.265.

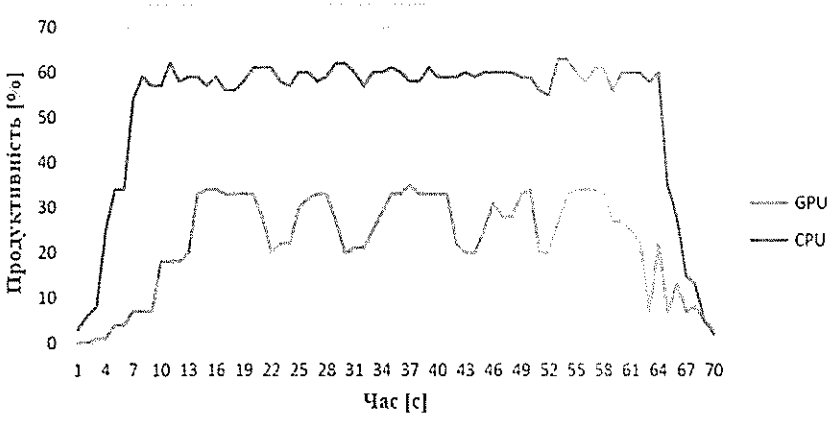
| Відео | Один рухомий об'єкт | | Велика кількість рухомих об'єктів | |
|---|---|---|---|---|
| Якість відео | 1280x920 | | 3840x2160 | |
| Розмір файлу для CUDA | 5 765 850 кБ | | 8 723 700 кБ | |
| Кодек | H.264 | H.265 | H.264 | H.265 |
| Кінцевий розмір бітового потоку | 13610 кБ | 14259 кБ | 29 195 кБ | 35 111 кБ |
| Отриманий час кодування | Δ 873,62 ± 5,86 мс | Δ 511,9 ± 5,62 мс | 7 436,55 ± 13 мс | 7357,05 ± 74,5 мс |
| Середній час для одного кадру | 1,83 ± 0 мс | 1,29 ± 0,12 мс | 10,94 ± 0,24 мс | 10,56 ± 0,29 мс |
| Розмір файлу для DirectX | 10 912 645 кБ | | 16 654 214 кБ | |
| Кодек | H.264 | H.265 | H.264 | H.265 |
| Кінцевий розмір бітового потоку | 14 511 кБ | 15 294 кБ | 31 005 кБ | 36 215 кБ |
| Отриманий час кодування | 35 504 ± 557 мс | 35441 ± 354,7 мс | 51 546,33 ± 530,4 мс | 51784 ± 197 мс |
| Середній час для одного кадру | 8,31 ± 0,129 мс | 8,3 ± 0,07 мс | 71,79 ± 0,74 мс | 72,12 ± 0,274 мс |

Результати вимірювання пікового співвідношення сигналу до шуму залежно від глибини стиснення

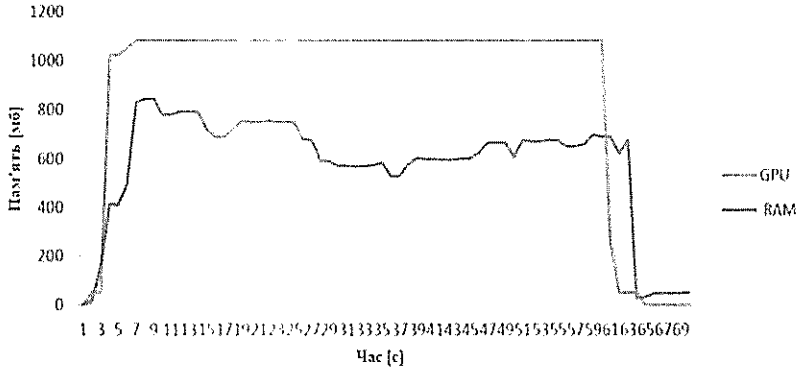
| Кодек | H.265 | H.264 |
|---|---|---|
| QP = 1 | 39,05 дБ | 37,12 дБ |
| QP = 15 | 36,12 дБ | 35,82 дБ |
| QP = 22 | 35,17 дБ | 33,72 дБ |
| QP = 30 | 34,49 дБ | 33,83 дБ |
| QP = 40 | 33,43 дБ | 33,45 дБ |
| QP = 50 | 32,82 дБ | 32,80 дБ |

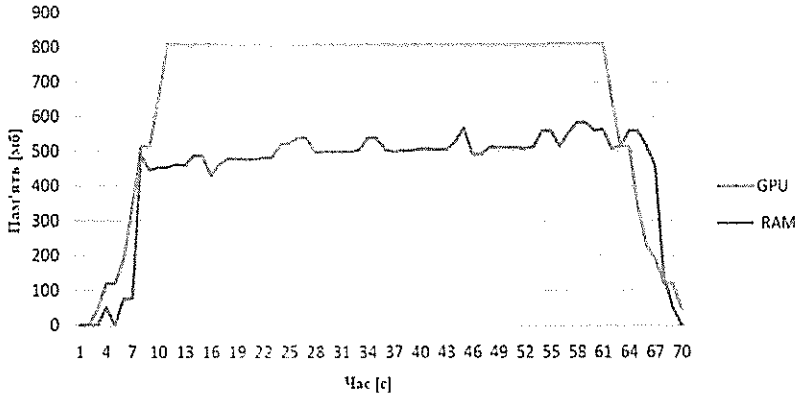
# Графіки продуктивності CPU та GPU при кодуванні




Завантаження у %




Завантаження у МБ

# Висновки

Магістерська робота присвячена питанням кодування відео 4K у режимі реального часу. Містить основний опис найбільш використовуваних відеокодеків H.264 та H.265. Показано принципи роботи графічних карт та їх програмованих блоків. Проаналізовано рішення вільно доступного SDK Video Codec. Розроблено програму, яка може кодувати відео з декількох камер 4K в режимі реального часу.

Практична частина роботи реалізована на мові програмування C ++. Тут описані найважливіші частини реалізації програми. Описано розподіл процесів на кілька потоків за допомогою технології Visual Studio 2019.

Важлива частина опису реалізації присвячена процесу кодування. Додаток було успішно реалізовано, і, крім роботи, був реалізований графічний інтерфейс для запуску отриманих додатків та контролю навантаження кодування в системі.

Проведено тестування ефективності та якості роботи розробленого ПЗ кодування відео з декількох камер 4K в режимі реального часу.

# Апробація результатів магістерської роботи

Основні результати магістерської роботи доповідалися на Всеукраїнській науково-практичній конференції «Telecommunication: problems and innovation», Київ: ДУІКТ, 20 грудня 2023 р.

Основні наукові результати магістерської роботи опубліковано в науковому журналі “Наукові записки ДУІКТ, – 2023, – №2”.


12

**Протокол аналізу звіту подібності науковим керівником**

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

**Автор:** Юрій ЛИЦЕНКО

**Назва:** Методи стиснення потокових відеоданих на основі сучасних відеокодеків

**Координатор:** Віталій СТРЕЛЬНІКОВ

**Підрозділ:** ННІТ

**Коефіцієнт подібності 1** :8.6

**Коефіцієнт подібності 2** :4.2

**Тривога:** 5

**Після аналізу Звіту подібності констатую наступне:**

☑ виявлені в роботі запозичення є сумлінними і не мають ознак плагіату. Тому робота визнається самостійною і допускається до захисту;

☐ виявлені в роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і самостійності її автора. Роботу направити на доопрацювання;

☐ виявлені в роботі запозичення є недобросовісними і мають ознаки плагіату або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. У зв'язку з чим, робота не допускається до захисту.

..... .....

*Дата* *Підпис Наукового керівника*