НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
Кафедра Системного проектування

«До захисту допущено»
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
"___"_____2015 р.

Дипломна робота

першого (бакалаврського) **рівня вищої освіти**

зі спеціальності 7.050102, 8.050102 Інформаційні технології проектування

на тему: Відеокодек стандарту H.264

Виконав: студент 3 курсу, групи ДА-11

Солдатенко Ярослав Валерійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник ст. викладач Бритов О.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант Охорона праці доцент Гусєв А.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Нормоконтроль ст. викладач Бритов О.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2015

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
Кафедра Системного проектування
Рівень вищої освіти Перший (Бакалаврський)
Спеціальність 7.050102, 8.050102 Інформаційні технології проектування

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис)
«___»_____2015 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Солдатенко Ярославу Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)) Відеокодек стандарту H/264
керівник проекту (роботи) Бритов Олексій Анатолійович, ст.викладач
затверджені наказом по університету від «02» квітня 2015 р. № 30/1-ст

2. Строк подання студентом проекту (роботи) – 24 червня 2015 року

3. Вихідні дані до проекту (роботи)

Стандарт H.264 (MPEG-4 AVC)
Мова реалізації – довільна
Базовий набір бібліотек – x264

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)

1. Розглянути структуру сучасного відеокодеку
2. Визначити функції та мову програмування, необхідні для самостійної реалізації відеокодеку
3. Розробити алгоритм роботи та структуру програми
4. Розробити план тестування програмної реалізації
5. Програмно реалізувати відеокодек та протестувати його роботу
6. Порівняти власну реалізацію з існуючими

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів тощо)

1. Структурна схема сучасного відеокодеку – плакат
2. Технологія передбачення руху - плакат
3. Порівняння сучасних реалізацій - плакат
4. Результати тестування програмної реалізації – плакат

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Гусєв А.М, доцент		

7. Дата видачі завдання - 01.02.2015

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2015	
2	Збір інформації	15.02.2015	
3	Вивчення структури відеокодеку	28.02.2015	
4	Визначення функцій, необхідних для реалізації відеокодеку	10.03.2015	
5	Розробка плану тестування	20.03.2015	
6	Програмна реалізація відеокодеку	25.04.2015	
7	Тестування програмної реалізації	30.04.2015	
8	Оформлення дипломної роботи	31.05.2015	
9	Отримання допуску до захисту та подача роботи в ДЕК	24.06.2015	

Студент _____ Я.В.Солдатенко
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ О.А.Бритов
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

до бакалаврської дипломної роботи Солдатенко Ярослава Валерійовича
на тему «Відеокодек стандарту H.264»

Дана дипломна робота присвячена дослідженню сучасних засобів та методів обробки мультимедійних потоків. Приведено огляд існуючих реалізацій відеокодеків. Розглянуто типову структуру систем кодування відео, проаналізовано алгоритми реалізації окремих її частин, визначено їх переваги та недоліки. Розроблено комп'ютерну програму, в якій реалізовано можливості примітивного відеокодека. Проведено тести існуючих відеокодеків для визначення їх характеристик та порівняння з написаною власноруч реалізацією. Дану роботу можна використовувати як посібник для самостійної розробки відеокодека як розглянутого, так і будь-якого іншого стандарту .

Загальний обсяг роботи - 79 сторінок, з них основна частина – 54 сорынки, 17 рисунків, 3 таблиці, 9 посилань, 2 додатки.

Ключові слова: H.264, MPEG-4 AVC, відеокодек, стиснення відео, дискретне косинусне перетворення, багатокадрове передбачення.

АННОТАЦИЯ

к бакалаврской дипломной работе Солдатенко Ярослава Валериевича
на тему «Видеокодек стандарта H.264»

Данная дипломная работа посвящена исследованию современных средств и методов обработки мультимедийных потоков. Приведены обзор существующих реализаций видеокодеков. Рассмотрена типовая структура систем кодирования видео, проанализированы алгоритмы реализации отдельных ее частей, определены их преимущества и недостатки. Разработана компьютерная программа, в которой реализовано возможности простого видеокодека. Проведены тесты существующих видеокодеков для определения их характеристик и сравнение с написанной собственноручно реализацией. Данную работу можно использовать как пособие для самостоятельной разработки видеокодека как рассматриваемого, так и любого другого стандарта.

Общий объем работы - 79 страниц, из них основная часть – 54 страницы, 17 рисунков, 3 таблицы, 9 ссылок, 2 приложения.

Ключевые слова: H.264, MPEG-4 AVC, видеокодек, сжатие видео, дискретное косинусное преобразование, многокадровые предсказания.

ABSTRACT

for a bachelor’s degree work, made by Soldatenko Yaroslav Valerievich

“H.264 Video Codec”

This work is devoted to research modern methods of multimedia streams handling. Paper is powered by reviews of existing implementations. Considered typical structure of video encoding algorithms, analyzed implementation of certain parts identified their advantages and disadvantages. Computer program was made that realize the possibilities of primitive video codec. Existing codecs tests were made to determine their characteristics and comparisons with my own written implementation. This work can be used as a guide for self-development of video codec as reporting and any other standard.

The total amount of work is 79 pages, with the bulk of 54 pages, 17 pictures, 3 tables, 9 references, 2 addition.

Keywords: H.264, MPEG-4 AVC, video codec, video compression, discrete cosine transform, multiframe prediction.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AVC – Advanced Video Coding

CABAC – Context Adaptive Binary Arithmetic Coding (Контекстно-адаптивне бінарне арифметичне кодування)

CAVLC – Context Adaptive Variable Length Coding (Контекстно-адаптивне кодування змінної довжини).

DCT – Discrete Cosine Transform (Дискретне косинусне перетворення)

Intel IPP – Intel Integrated Performance Primitives (бібліотека для створення програмних додатків)

MPEG – Moving Picture Experts Group

YUV – кольорова модель

ВСТУП

Важко переоцінити роль інформації в сучасному світі. Вона пронизує всю область життя суспільства. Недарма існує вислів: «Хто володіє інформацією - володіє світом». І не менш важливим є питання ефективної передачі інформації. Якісна і вдала передача є умовою успішної практичної діяльності людей, умовою подальшого інтенсивного розвитку. Обсяг інформації, необхідний для нормального функціонування сучасного суспільства, зростає приблизно пропорційно квадрату розвитку промислового потенціалу, тому питання стиснення інформації є надзвичайно актуальним.

Стиснення інформації є одним з найбільш цікавих напрямків сучасної науки, що дуже динамічно розвивається - теорії інформації. Питання стиснення даних досить гостро стоїть в різних галузі науки і техніки, скрізь, де потрібні зберігання та передача інформації. Навіть у сучасному світі, зі швидкісним інтернет-з'єднанням і необмеженими сховищами інформації, стиснення даних актуально так само, як і раніше, особливо для мобільних пристроїв і країн з повільним інтернетом.

Стиснення цифрових зображень та відеоматеріалів – це взяття «сирого» (RAW), нестисненого відео (наприклад, знятого цифровою камерою) і використання надлишкової інформації як з одного нерухомого зображення (так званого фрейму), так і між попереднім і наступним кадрами. На відміну від інших способів стиснення, алгоритми стиснення відео без втрат (lossless) не призводить до тієї ж ступені зменшення розміру, як при стисненні, наприклад, текстового документу.

Внаслідок цього цифрове відео часто стискають з використанням алгоритмів стиснення з втратами (lossy). Через те, що стиснуте відео займає набагато менше місця, ніж нестиснуте, ціною відносно незначного зниження якості зображення, майже все сучасне цифрове відео передається (а іноді й зберігається) в стиснутому вигляді.

У деяких ситуаціях є вимога до стискання відео паралельно його створенню; наприклад, у відеотрансляціях, відеотелефонії чи відеоконференціях. У таких ситуаціях відправнику важливо мати можливість швидко стискати і передавати кадри в той же час, як вони записуються. В іншому випадку необхідно мати великий обсяг пам'яті чи тимчасового сховища для утримання відеоматеріалу. У відеотелефонії та відеоконференціях додатково потрібно мати низьку затримку кодування. (Тут затримка визначається як час від моменту, коли відео захопили, до моменту, коли стисла версія досягає адресата і відображаються розпаковані кадри.) Великі затримки у відеотелефонії роблять двосторонній зв'язок і неприродним, і громіздким.

Мета цієї роботи - розібратися зі структурою та існуючою реалізацією відеокодека стандарту H.264 / AVC та спробувати розробити реалізацію власну. Також вона досліджує, як на складність кодування (час кодування відео), ефективність стиснення (різниця в розмірі до і після стиснення) і якість впливає оптимізація відеокодека. Незважаючи на те, що немає конкретних сценаріїв в даному дослідженні, затримка кодування також вважається важливим фактором при порівнянні різних реалізацій.

1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО ВІДЕОКОДУВАННЯ

1.1 Поняття відеокодека

Відеокодек – це програма чи алгоритм стиснення (тобто зменшення розміру) відеоданих (відеофайлу, відеопотоку) з подальшим відновленням стиснених даних. Фактично кодек - файл-формула, яка визначає, яким чином можна «упакувати» відеоконтент і, відповідно, відтворити відео. Також можливе шифрування відео- й аудіоінформації, додавання субтитрів, векторних ефектів і т. п.

Формат стислих даних, як правило, відповідає стандартній специфікації стиснення відео. Стиснення з втратами означає, що у стисненому відео відсутня частина інформації, що присутня в оригінальному відео. Наслідком цього є те, що розпаковане відео має нижчу якість, ніж оригінал, тому що інформації для точного відновлення вихідного відео недостатньо.

1.2 Принцип роботи найпростішого кодека

1.2.1 Кодування статичного зображення

Спершу розглянемо кодування статичного зображення (або одиночного кадру). Кожен кадр відеопотоку складається з точок (пікселів), що утворюють матрицю (растр). Кодек може відстежувати схожі масиви точок з однаковими атрибутами (наприклад, синій колір фону на зображенні неба) і, замість того, щоб запам'ятовувати інформацію про кожну точку (яскравість і колір) в наступних кадрах окремо, може записати лише першу (ключову) точку і лічильник з кількістю повторень цієї точки до моменту зміни кольору даної точки. Тобто замість опису, наприклад, 1000 точок, може бути достатньо описати всього 1 точку, порахувавши її повторення. Якістю тут можна управляти, задаючи величину, таку, що якщо відмінність між точками менше, то вони вважаються однаковими. Це найпростіший з методів стиснення.

1.2.2 Типи кадрів

Для побудови динамічного зображення (відеоряду) використовуються різні типи кадрів. Типи кадрів відеопотоку - це способи кодування і зберігання інформації про чергове кадрі, що відрізняються один від одного наявністю або відсутністю залежностей цього кадру від попередніх і наступних. Зазвичай кадр розбивається на квадратні макроблоки, і тип посилання для кожного з макроблоків визначається індивідуально, проте з обмеженням, заданим типом всього кадру: крім I-кадрів (які також називаються ключовими (англ. keyframes) або «опорними») які можуть містити тільки незалежно стислі макроблоки, існують P-кадри («різницеві»), які можуть містити як незалежно стислі макроблоки, так і макроблоки з посиланням на інший I- або P-кадр, та B-кадри («двонаправлені», «зворотні»), що можуть містити незалежні (intra) макроблоки, макроблоки з посиланням на один кадр (predicted) або з посиланням на 2 кадри (bi-predicted). B-кадри посилаються на найближчі I-, P- або B-кадри. У стандарті MPEG-4 AVC / H.264 також вводиться поняття Ci-і SP-кадрів.

Наприклад, у кодеках MJPEG і DV всі кадри відеопотоку - I-типу. Сімейство кодеків MPEG4 «третьої версії» (найбільш популярними буди DivX та OpenDivX) має два типи кадрів - I та P. B-кадри не передбачені. Така ж ситуація в сімействі кодеків від On2: VP3, VP6, VP8. Крім того, багато сучасних кодеків мають налаштування, що вимикають створення B-кадрів для зниження витрат процесорної потужності на обробку в реальному часі.

У стандартах на стиснення відео зазвичай стискається тільки різниця між кадрами. Наприклад, у сцені, де людина йде на тлі нерухомих об'єктів, потрібно зберігати тільки інформацію про області, що змінюються (наприклад, використовується компенсація руху, при якій зберігається вектор зміни положення блоку або, якщо схожа область в попередньому кадрі не знайдена, дана область стискається як незалежна зображення). Частини сцени, які не змінюються, не зберігаються в потік, за рахунок чого значно зростає ступінь стиснення у порівнянні з форматами, що використовують незалежне стиснення кожного кадру. Наприклад, для I-P і кадрів у потоці утворюються ланцюжки

IPPPPPPPPPPP, коли перший кадр стискається незалежно, а наступні - з посиланням на перший кадр. Це найпростіший приклад використання різних типів кадрів в потоці.

У той час як основною перевагою використання P-кадрів є збільшення ступеня стиснення, їх основним недоліком є різко зростаючий час доступу до кадру, оскільки для отримання потрібного кадру необхідно повністю розпакувати весь ланцюжок кадрів від найближчого I-кадру. Зокрема, якщо при стисненні були задані параметри, що максимізують ступінь стиснення, при яких I-кадри зустрічаються рідко, час затримки показу довільного кадру в потоці може бути дуже помітним.

У стисненому відеокодеком потоці для стандартів MPEG-2, MPEG-4, H.261 і H.263 використовуються кадри всіх трьох основних типів – I, P і B. Використання B-кадрів означає, що даний кадр посилається на два сусідніх I- або P-кадри в потоці, в цьому випадку вид ланцюжка кадрів може бути таким: IBPBPBPBPBPBPBP. Найчастіше використовуються ланцюжка (так звані GOP - групи зображень чи "структура групи кадрів») виду IBBPBBPBBPBBPBBPBBPBBPBBP, в яких B-кадри посилаються на два найближчих сусідніх I- або P-кадри і незалежні між собою. Дана структура дозволяє в 2-3 рази прискорити час отримання довільного кадру в потоці, оскільки для його отримання необхідно розпакувати тільки кожен другий (третій) кадр, починаючи з I-кадру. Також в кілька разів зростає швидкість «швидкого перемотування з показом».

Більш розвинені формати стиснення враховують, крім вищезгаданої технології, ще й принципи руху масивів точок у зображенні, сегментування картинки на блоки з різною якістю стиснення, застосування послідовності кадрів, кодованих по-різному і показаних в певній послідовності. Найновіші кодеки враховують психофізичні властивості сприйняття відео людським оком і мозком, що дозволяє ще сильніше зменшувати розмір даних без «видимої втрати якості». Також, алгоритми використовують схожість сусідніх кадрів у відеоряді.

1.3.1 Компенсація руху

1.3.1 Загальні відомості

Компенсація руху (англ. Motion Compensation) - один з основних алгоритмів, застосовуваних при обробці і стисненні відеоданих. Алгоритм використовує схожість сусідніх кадрів у відеопослідовності і знаходить вектори руху окремих частин зображення (зазвичай блоків 16x16 і 8x8). Використання компенсації дозволяє при стисненні багаторазово збільшити ступінь стиснення за рахунок видалення надмірності у вигляді співпадаючих частин кадрів. Використовується не тільки при стисненні, але і при фільтрації відео, зміні частоти кадрів і т.д.

1.3.2 Ідея алгоритму

Рішення проблеми стиснення стало першочерговим завданням, починаючи з самої появи цифрового відео. Для оцінки візьмемо відеоряд з наступними параметрами:

- Розмір кадру: 720x576 (Стандартний розмір для Європейського телебачення (PAL), 414720 пікселів)
- Частота кадрів: 25 к / сек (Також стандартно для PAL)
- Кольоропредставлення: YUV 4:2:0 (12 біт на піксель)

У підсумку на запис або передачу однієї секунди такого відео без застосування стиснення буде потрібно 14,8 мегабайта без урахування звуку та службової інформації. Для зберігання півторагодинного фільму вже буде потрібно 79 920 мегабайт (78 гігабайт).

Практично в будь-якому відео сусідні кадри схожі, мають спільні об'єкти, які, як правило, зміщуються один відносно одного. Цілком природнім є бажання закодувати відео так, щоб об'єкти не кодувалися багаторазово, а просто описувалися деякі їх зміщення.

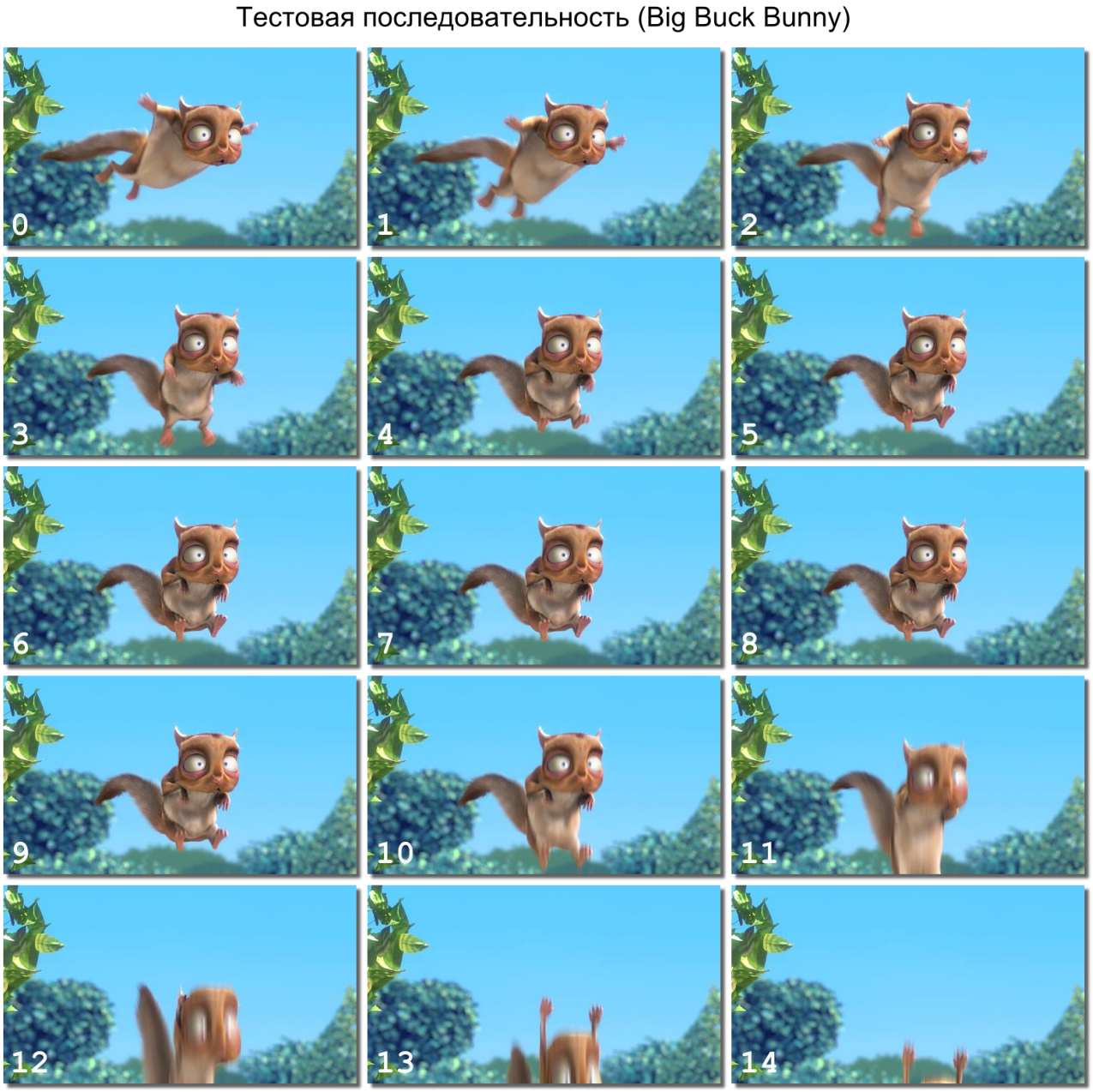

Рисунок 1.1 - У цьому фрагменті явно видно схожість сусідніх кадрів, що типово для будь-якого відео [1].

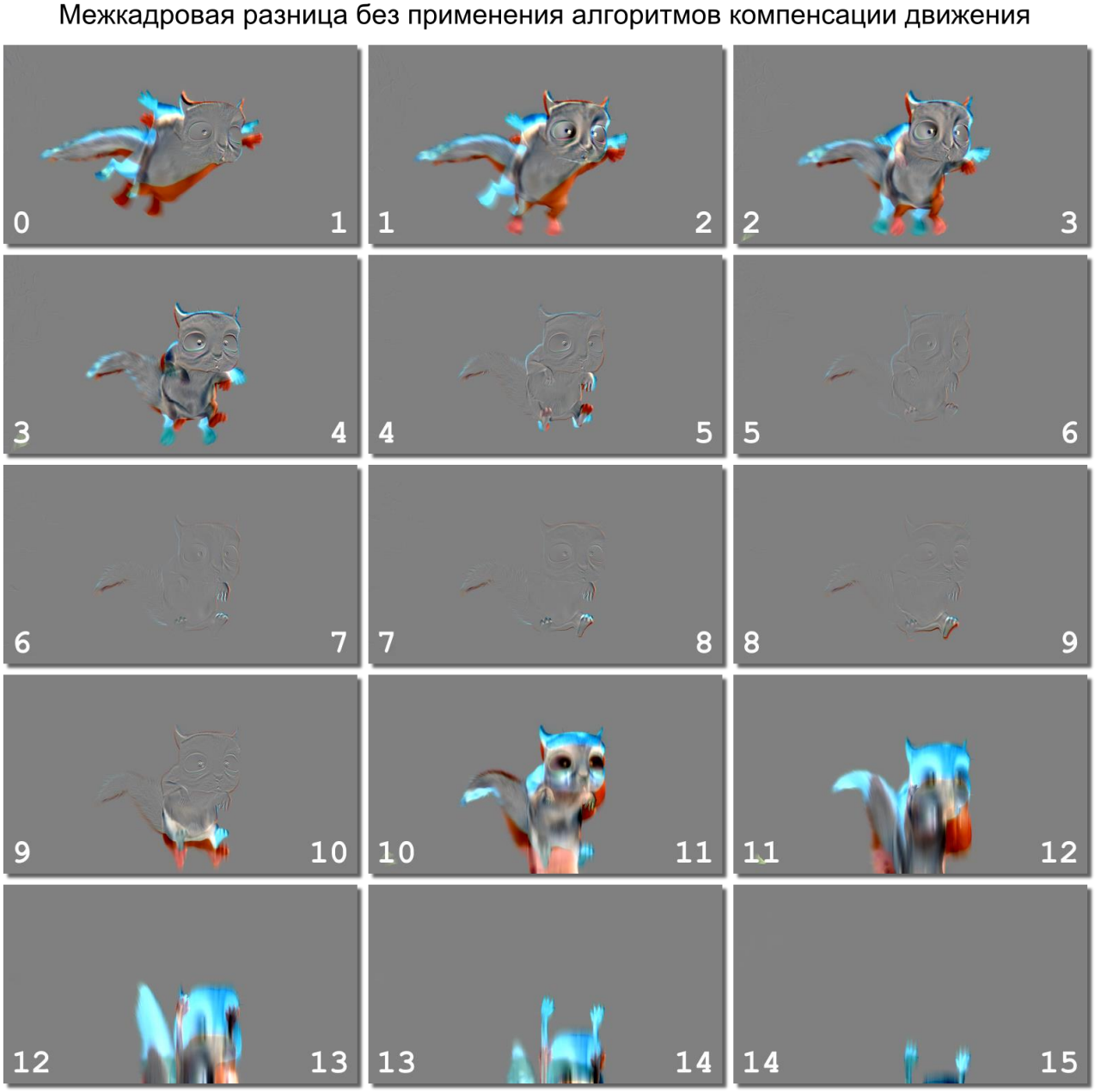

Рисунок 1.2 - Видно, що зображення міжкадрової різниці мають більш просту структуру, фон став одноколірним [1]

Навіть у цьому прикладі якщо взяти і запакувати архіватором 0-й кадр і всі зображення міжкадрового різниці, вийде помітний виграш при стисненні. Але цей виграш можна суттєво збільшити.

1.3.3 Приклад роботи алгоритму

У зв'язку з високою обчислювальною складністю алгоритмів розпізнавання образів і недостатньої точності їх роботи застосовують різні методи, що дозволяють швидко знаходити вектори руху (звісно, не без втрат):

1. Завантажується поточний кадр.
2. Кадр ділиться на блоки (наприклад, 16x16).

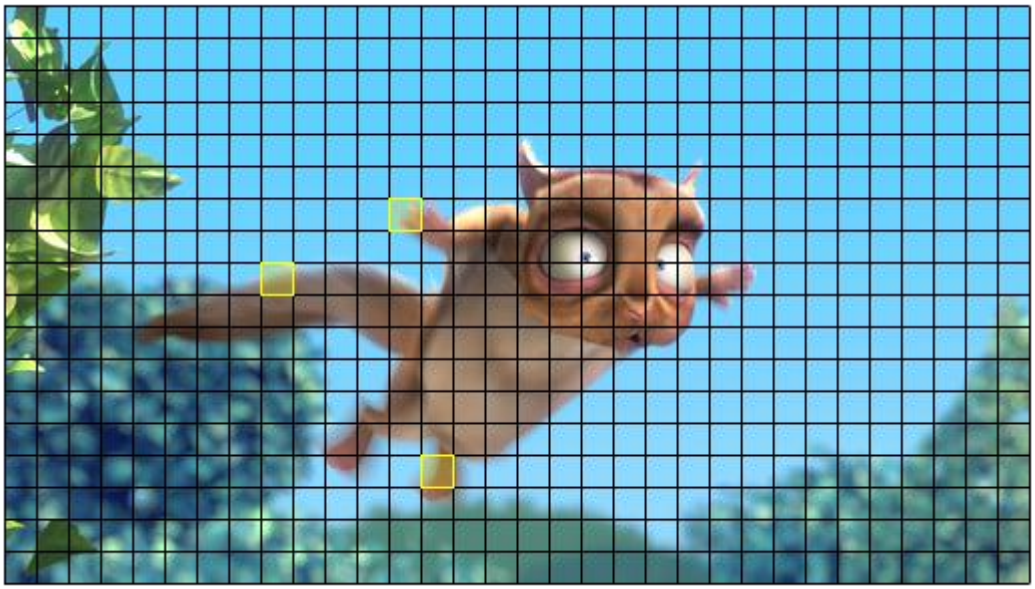

Рисунок 1.3 - Поділений на блоки кадр [1]

3. Проводиться обхід блоків (кожен блок в даному випадку обробляється окремо).
4. При проході кожного блоку проводиться обхід деякої околиці блоку в процесіпошуку максимальної відповідності зображенню блоку на попередньому кадрі в межах цієї околиці.

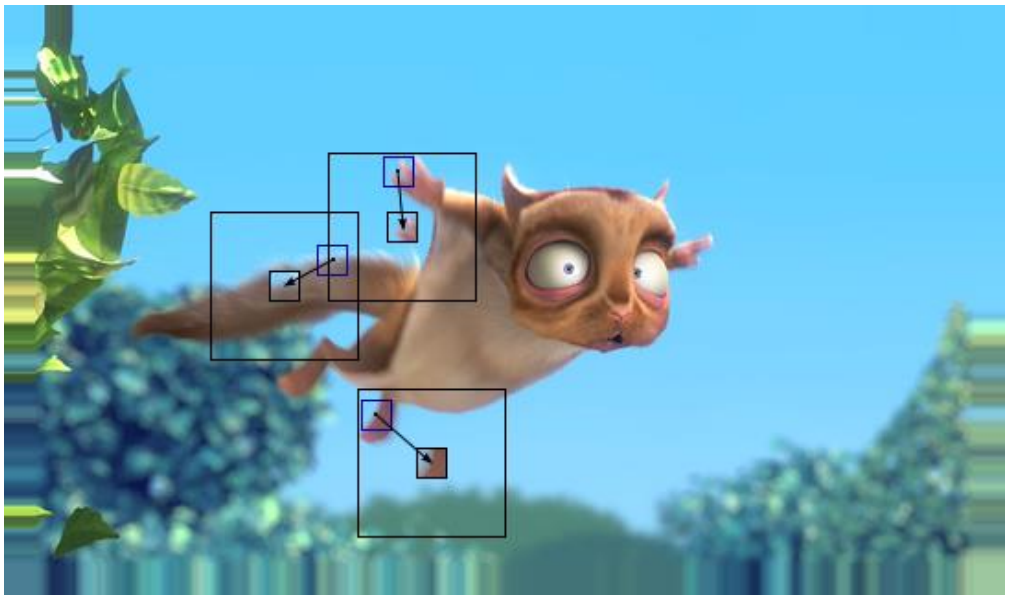

Рисунок 1.4 - Наочна ілюстрація пошуку: зображено попередній кадр (той в якому проводиться пошук) і три блоки нового кадру, які ми хочемо наблизити фрагментами попереднього [1]

5. Таким чином, після завершення пошуку ми отримуємо набір векторів, який вказує "рух" блоків зображення між кадрами. Ці вектори можуть бути використані для створення зображення скомпенсованого кадру, який краще наближає кадр, для якого виконувалась компенсація руху.

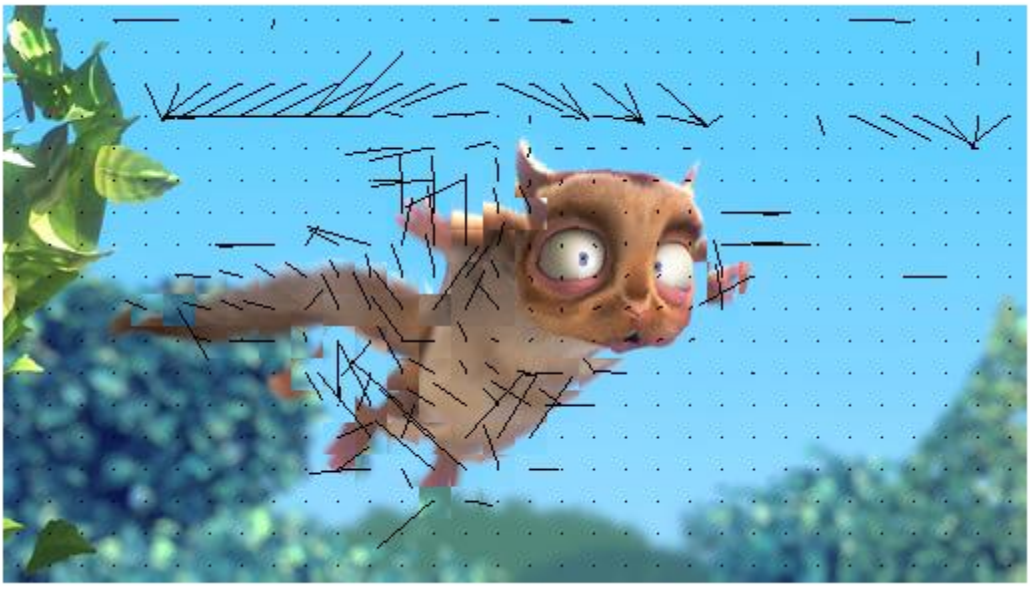

Рисунок 1.5 - Тут показано скомпенсований кадр з векторами руху для кожного блоку (точка - це нульовий вектор) [1]

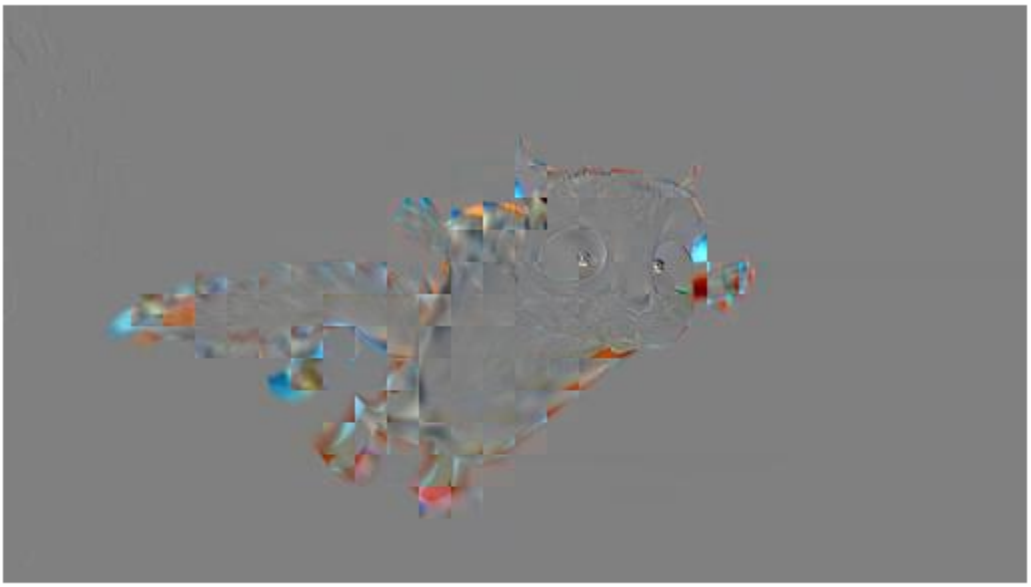

Рисунок 1.6 - Як видно, різниця між скомпенсованим кадром і поточним значно менше, ніж між нескомпенсованими кадрами [1]

1.3.4 Проблеми реалізації

Перше питання, яке виникає при написанні алгоритму компенсації руху: "Як оцінювати схожість фрагментів зображення?"

Варіантів може бути декілька:

- Обчислення SSD (суми квадратичних відхилень). Для пари блоків дає хороші результати за якістю (особливо при еталонних тестах, таких як метрика PSNR (обчислюється на основі середнього квадратичного відхилення)), але вимагає значних витрат ресурсів (множення - операція повільна, навіть таблиця квадратів не сильно прискорює процес) і сильно чутливе до зміни яскравості. Чим менше SSD - тим більше схожі блоки.
- Порівняння по характерних точках. Може виконуватися дуже швидко (за рахунок обходу лише невеликого числа точок), але може дуже погано корелювати з більш якісними метриками.
- Обчислення SAD (суми абсолютних різниць). Виконується за розумний час і дає прийнятний результат за якістю (але має низьку стійкість до шуму). Реально застосовується і має хороші швидкісні показники за

рахунок використання SIMD-розширень (які дозволяють виконувати безліч розрахунків одночасно без використання "інтелектуальних" засобів процесора по розпаралелюванню обчислень).

Найбільш часто використовується обчислення SAD. Друге питання більш складний: "Як шукати потрібний блок?"

- Повний перебір (Full Search). В деякій області навколо оброблюваного блоку відбувається перебір координат шуканого блоку. Якщо маємо блок 16x16 і область пошуку ±32 x ±32, то нам потрібно буде 4096 разів порахувати SAD для кожного оброблюваного блоку. Це повільно, але дає гарантовано кращий результат за заданою метриці.
- Пошук за шаблоном. Виконується швидко, дає не найкращі результати.
- Спіральний пошук. Вважається, що чим ближче блок до поточного, тим більша ймовірність того, що він шуканий, і його точність зменшується від центру до країв області пошуку. На картинці з векторами руху видно довгі вектори на небі, так як використовувався повний перебір починаючи з лівого верхнього кута області пошуку, хоча очевидно, що з нульовими векторами немає практично ніякої різниці, але довгі вектори погіршують стисливість поля векторів, а нульові вектори ні. При спіральному пошуку на незмінних ділянках завжди стоять нульові вектори.

У стандарті MPEG-4 AVC / H.264 введені також не квадратні (прямокутні) блоки, розмір яких може дробитися до 4x4 пікселя. Таким чином вдається досить ефективно використовувати схожість сусідніх кадрів, а завдяки більш складній формі блоків зростає точність компенсації руху на кордонах рухомих об'єктів. Крім компенсації руху для подальшого уточнення зображення (або для знову з'являються областей, яких не було в минулих кадрах) використовується стиснення міжкадрової інформації та незалежне стиснення блоків.

Крім стиснення компенсація руху активно використовується у фільтрації відео, зокрема в якісних варіантах фільтрів: деінтерлейсингу (перетворення черезрядковою розгортки в прогресивну), шумозаглушенні, зміні частоти кадрів та інших.

1.3.2 Висновки до розділу 1

В даному розділі було наведено загальні відомості про відеокодеки та розглянуто основні принципи їх роботи. Наведена інформація про головне джерело зменшення розміру стиснутого файлу – «неповноцінні» типи кадрів. Також приведено один з основних алгоритмів, що застосовується при обробці та стисненні відеоданих = компенсацію руху. Алгоритм супроводжується наочними ілюстраціями зі зрозумілими коментарями.

2. СТРУКТУРА ВІДЕОКОДЕКА

2.1 Основні принципи роботи кодека стандарту H.264

2.1.1 Режими роботи кодека

Як і в попередників - відеокодеків сімейства H.26X - H.264 має два режими кодування для окремих відеокадрів – intra та inter (внутрішній та міжкадровий). У першому випадку фрейм кодується як самостійне зображення, без посилання на інші зображення в послідовності. У міжкадровому ж режимі попередній і, можливо, майбутні кадри використовуються для прогнозування значення. На рисунку 2.1 показано блоки на високому рівні, що беруть участь у внутрішньому кодуванні і декодуванні в H.264. Рисунок 2.2 показує процес кодування і декодування для взаємопов'язаних кадрів.

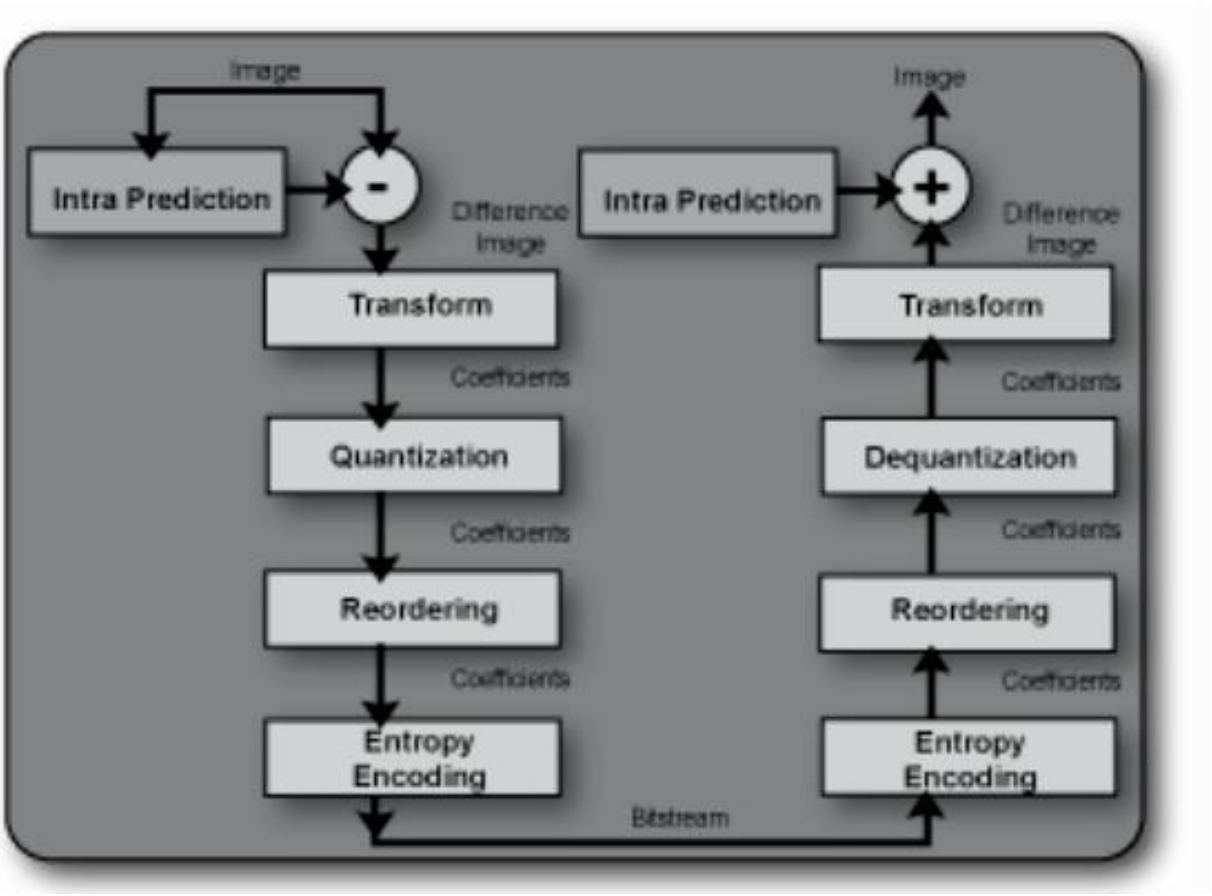

Рисунок 2.1 - Intra-режим кодування і декодування в H.264 [3]

Для обох режимів кодування блоки в H.264 можуть бути виражені по відношенню до попередніх і наступних блоків або кадрів. У взаємопов'язаних кадрах це називається «оцінки руху» і розраховується по відношенню до блоків в інших кадрах. Це є джерелом дуже значного зменшення розміру відео. Як і з іншими методами стиснення, цей використовує той факт, що існує значно менше ентропії в різниці між подібними блоками, ніж в абсолютних значеннях блоків. Це особливо вірно, якщо виражається різниця між оригінальним блоком і блоком, що побудовано зі зміщенням від цього блоку в іншому кадрі.

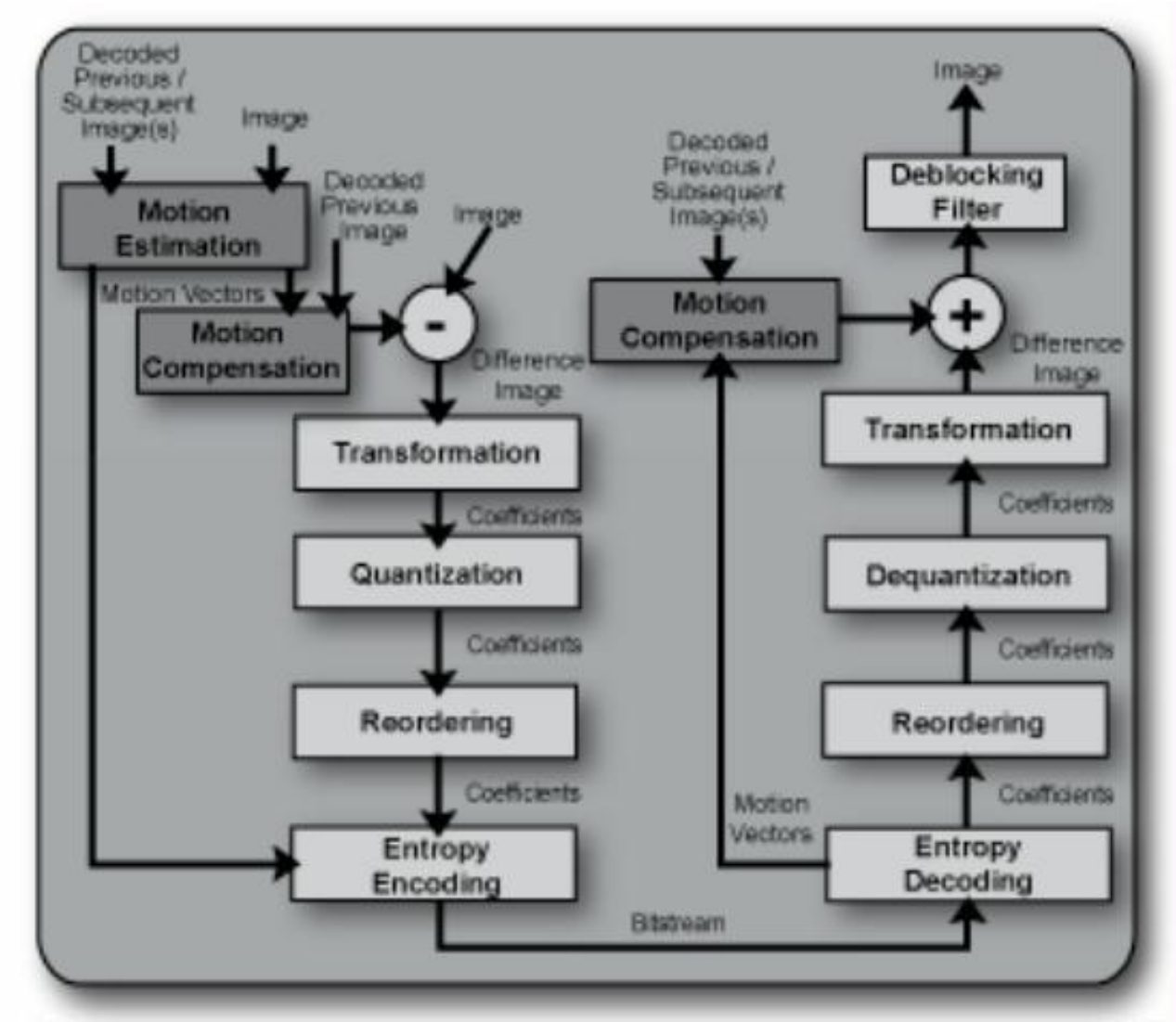

Рисунок 2.2 - Інтер-режим кодування і декодування в H.264 [3]

2.1.2 Оцінка руху

H.264 має дуже гнучку підтримку оцінки руху і може вибрати один з 32 інших кадрів як опорне зображення, і допускається позначати блоків, які можна побудувати шляхом інтерполяції. Кодер відповідає за визначення еталонного зображення, блоку і вектора руху. Цей блок, як правило, вибирається за допомогою деяких пошуку серед можливостей, починаючи з найбільш імовірних варіантів. Кодер обчислює і кодує різниця між раніше кодованими блоками і новими даними. Наприкінці, після декодування опорних блоків, кодер додає довідкові дані і декодовані різницеві дані. Блоки можуть бути декодовані не тимчасово, оскільки кадри можуть кодуватися по відношенню до майбутніх блоків та фреймів.

Стандарт H.264 підтримує дозвіл субпіксельних векторів руху, а це означає, що опорний блок фактично обчислюється шляхом інтерполяції всередині блоку реальних пікселів. Вектори руху для блоків яскравості виражені з роздільною здатністю у чверть пікселя, а для блоків кольоровості точність може бути на рівні восьмої частини пікселя. Такий розмір субпікселів значно збільшує алгоритмічну та обчислювальну складність. Декодування частини, яка вимагає виконання компенсації руху субпікселя тільки один раз в блоці, займає від 10 до 20 відсотків потужності декодування. Основна частина цього часу йде на інтерполяцію значень між пікселями для створення субпікселів зміщення опорних блоків.

Алгоритм інтерполяції для створення зміщення опорних блоків визначається по-різному для яскравості і кольоровості блоків. Для яскравості інтерполяція виконується в два етапи - до половини, а потім до чверті пікселя. Значення напівпікселей створюються шляхом фільтрації з цього ядра по горизонталі і вертикалі:

[1 -5 20 20 -5 1] / 32

Інтерполяція чвертьпікселя виконується взяттям лінійного середнього між сусідніми значеннями напівпікселів.

Компенсація руху для кольоровості блоків використовує білінійну інтерполяцію з чверті чи восьмої частини пікселя, залежно від формату кольоровості. Кожна позиція субпикселів є лінійною комбінацією сусідніх пікселів.

Рисунок 2.3 ілюструє, які пікселі, таким чином, використовуються для обох підходів інтерполяції.

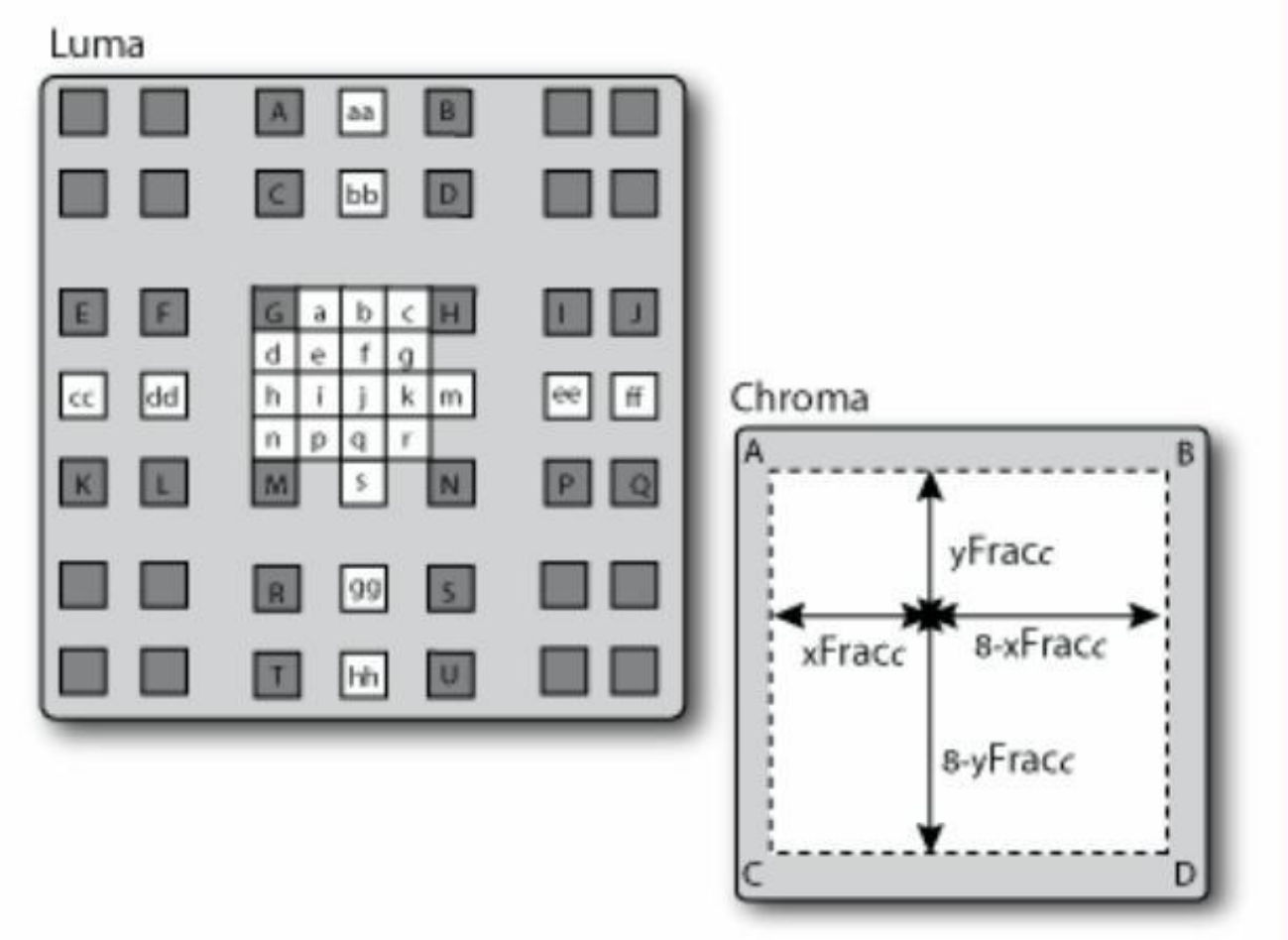

Рисунок 2.3 - Субпіксельна інтерполяція Motion Compensation в H.264 [3]

Після інтерполяції для створення опорного блоку, алгоритм додає цей опорний блок до декодованої різницево\ інформації, щоб отримати відновлений блок. Кодер виконує цей крок, щоб отримати відновлені опорні кадри, а декодер виконує цей крок, щоб отримати вихідні кадри.

2.1.3 Intra Prediction

Intra-кадри за своєю природою не залежать від ранніх або пізніх кадрів. Тим не менш, в H.264 кодер може використовувати ранні блоки всередині того ж кадру в якості посилання для нових блоків. Цей процес (внутрішнє передбачення) може дати додаткове стиснення для внутрішніх макроблоків, і бути особливо ефективним, якщо буде знайдено відповідний опорний блок.

Опорні блоки не використовують піксельну різницю фактичних блоків в сусідніх кадрах, так як взаємопов'язані блоки. Замість цього прогнозування поточного блоку розраховується як середнє значення деяких з пікселів, що обмежують його. Які з пікселів вибираються і як вони використовуються для розрахунку блоку - залежить від режиму внутрішнього передбачення. На рисунку 2.4 показані напрямки, які можуть бути використані, а також номери режимів, визначені у специфікації H.264.

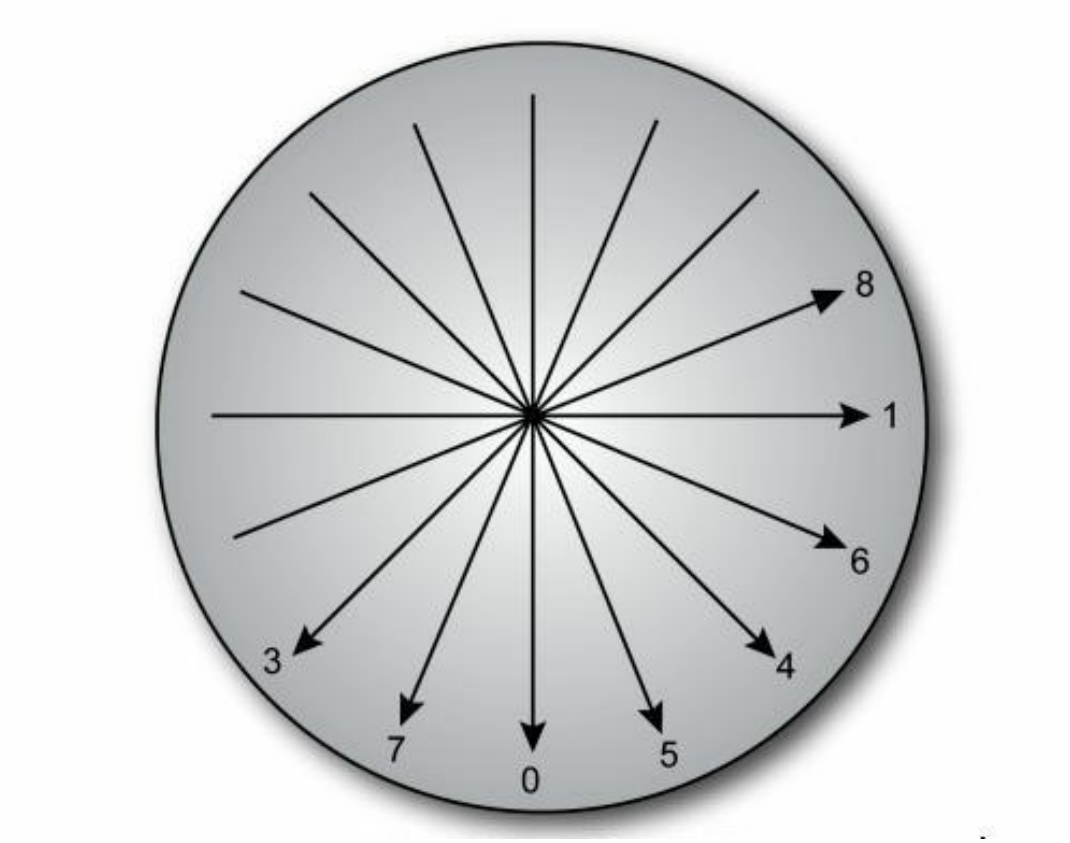

Рисунок 2.4 - Числа режимів для Intra Prediction в H.264 [3]

Це також може бути однією з найбільш інтенсивно обчислюваних частин процесу кодування. Для кодування виконується пошук по всіх варіантах - потрібно порівняти кожен блок яскравості 16x16 або блок кольоровості 8x8 з чотирма іншими блоками, і кожен блок яскравості 4x4 або 8x8 з 9 іншими блоками. Оскільки кодер може розглядати різні розміри блоків, бажано мати схему, яка оптимізує компроміс між числом бітів, необхідних для представлення відео та точністю результату.

2.1.4 Перетворення

Замість DCT, алгоритм H.264 використовує цілочисельне перетворення в якості основного перетворення, щоб перевести різницеві дані між просторовими і частотними доменами. Це перетворення є наближенням DCT, без втрат і обчислювально простіше. Ядро перетворення, показано на рисунку 2.5, може бути реалізоване з використанням тільки зсуву та додавання.

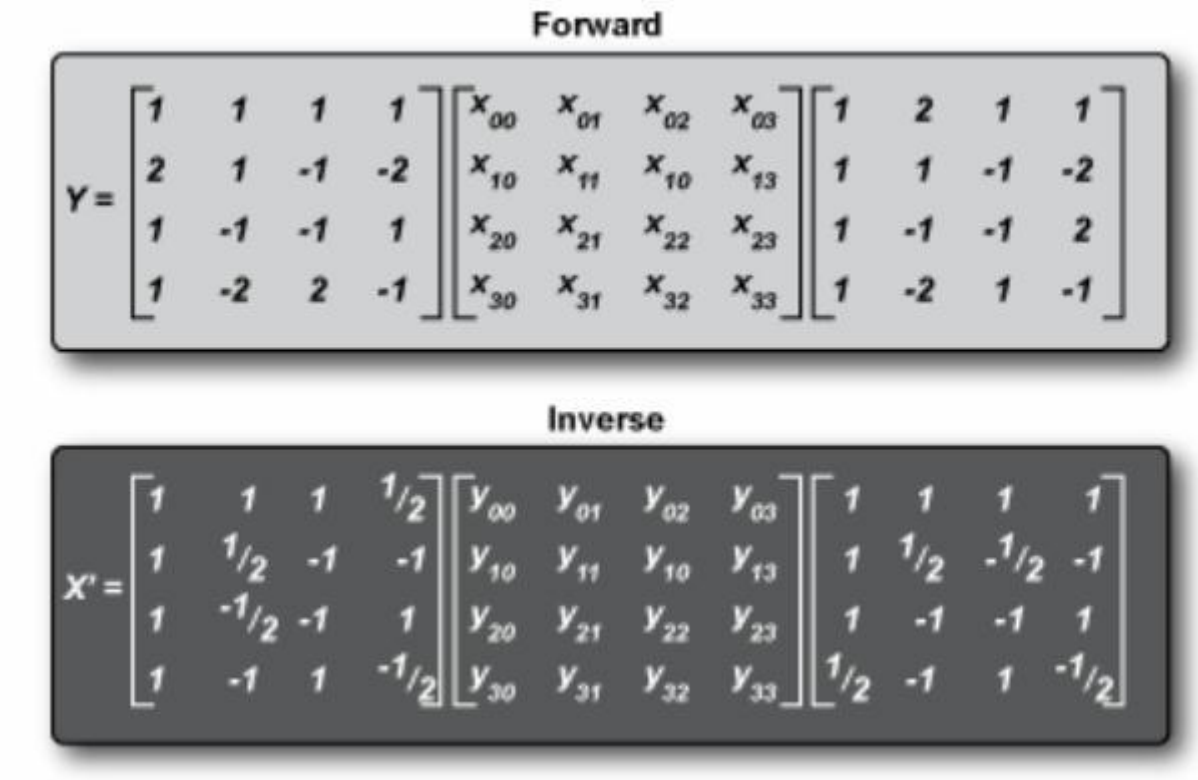

Рисунок 2.5 - Матриці для трансформації в H.264 [3]

Це перетворення для блоку 4x4 – лише одне в багатьох у H.264. Стандарт визначає перетворення блоків 2x2 і 4x4 у базовому профілі, а також підтримує додаткові профілі перетворення блоків великих розмірів, прямокутних або квадратних, з розмірами, кратними 2.

2.1.5 Квантування

Етап квантування зменшує кількість інформації шляхом ділення кожного коефіцієнта на конкретний номер, щоб зменшити кількість можливих значень. Так значення потрапляють в більш вузький діапазоні, що дозволяє використовувати ентропійне кодування для більш компактного зберігання значень.

Квантування в H.264 арифметично виражено як двухстадійна операції. Перший етап - множення кожного коефіцієнта в блоці 4x4 з фіксованим значенням коефіцієнта від конкретних умов. Цей етап дозволяє маштабувати коефіцієнти нерівномірно, виходячи з важливості чи інформації. Другий етап – ділення на регульоване значення параметром квантування (QP). Цей етап являє собою єдиний "важіль" для регулювання якості та результуючого бітрейту кодування. Дві операції можуть бути об'єднані в єдині операції множення і зсуву.

QP виражається як ціле число від 0 до 51. Це число перетворюється в розмір кроку квантування (QStep) нелінійно. Кожні шість кроків розмір кроку збільшується вдвічі, а між кожною парою кроків розміру N і 2N є 5 п'ять кроків: 1.125N, 1.25N, 1.375N, 1.625N, 1.75N.

2.1.6 Реорганізація

При кодуванні коефіцієнтів кожного макроблоку з використанням ентропійного кодування, кодек обробляє блоки в певному порядку. Порядок допомагає збільшити кількість послідовних нулів. Це природно впоратися з цим впорядкуванням на етапі квантування.

2.1.7 Ентропійне кодування

H.264 визначає два режими статистичного кодування – CAVLC і CABAC.

CAVLC можна розглядати як "базове" кодування. Це традиційний алгоритм кодування змінної довжини, з таблицею з однозначними префіксами, змінною бітовою довжиною кодів, а для додаткової ефективності стандарт визначає додаткові таблиці.

CAVLC вводить 12 додаткових кодових таблиць: 6 для характеристики змісту блоку перетворення в цілому, 4 для вказівки числа коефіцієнтів, 1 для індикації загальної величини квантованного значення коефіцієнта і 1 для відображення послідовних запусків нульових квантованих коефіцієнтів. Враховуючи ефективність таблиць VLC в поєднанні з адаптивним кодуванням для підвищення ефективності, це забезпечує непоганий компроміс між швидкістю та продуктивністю.

Режим CABAC було представлено для збільшення ефективності стиснення приблизно на 10 відсотків по відношенню до режимі CAVLC, хоча CABAC є значно складнішим в обчислюваннях. На першому етапі модель вибирається відповідно до набору останніх спостережень відповідних синтаксичних елементів; це називається контекстом моделювання. Якщо даний символ не є двійковим значенням, він буде відображатися на послідовності двійкових рішень, так звані бункерів, на другому етапі. Ця бінаризація здійснюється відповідно до специфікації, використовуючи деревовидну структуру, подібну коду VLC. Потім кожен бін кодується за допомогою імовірнісних оцінок, які залежать від конкретного контексту. Цю схему показано на рисунку 2.6.

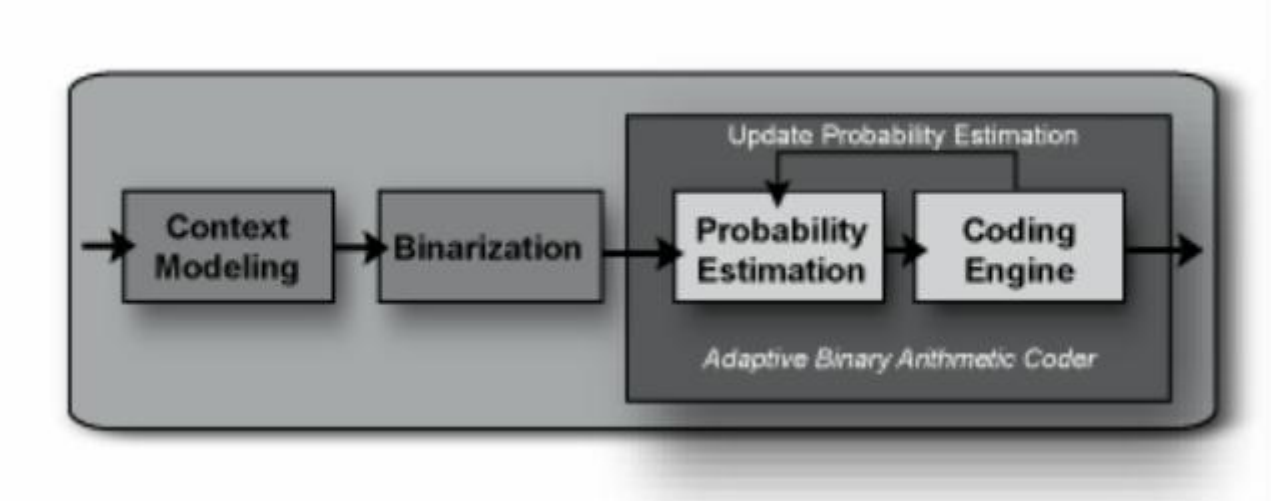

Рисунок 2.6. Арифметичне кодування в H.264 [3]

2.1.8 Деблоковий фільтр

Останній етап перед реконструкцією - фільтр деблокування. Цей фільтр призначено для згладжування візуальних розривів між блоками перетворення, і, таким чином застосовується тільки до пікселів, найближчих до цієї межі. Фільтр складається з відокремлених горизонтальних і вертикальних фільтрів. Рисунок 2.7 показує межі в макроблоках і пікселі, що представляють інтерес для горизонтального фільтра поперек вертикальної межі.

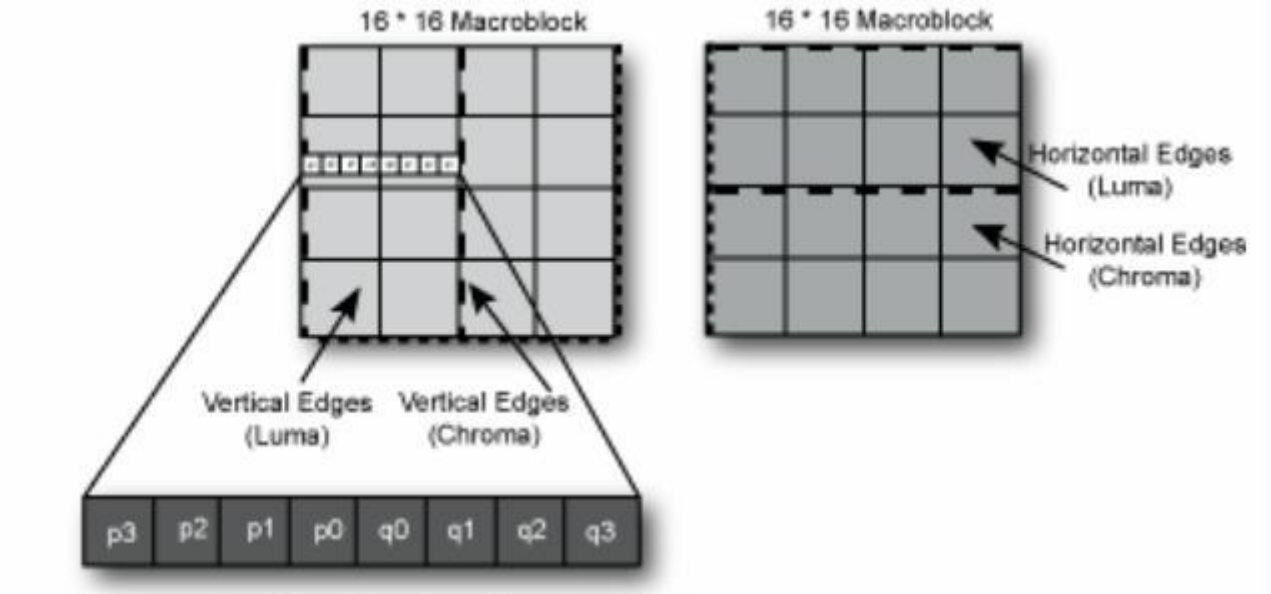

Рисунок 2.7 - Горизонтальний фільтр деблокування в H.264 [3]

В стандарті H.264 вказано, що фільтр застосовується в кадрах після деквантування і перед тим, як зображення використовується в якості опорного для компенсації руху. Для intra фреймів це слід застосовувати після intra prediction. Ця фільтрація є дуже важкою частиною декодеру в обчислювальному плані, забираючи від 15 до 30 відсотків потужності CPU для потоків з низьким бітрейтом, які найбільше вимагають фільтрації.

Фільтр блочності є адаптивним фільтром, потужність якого регулюється автоматично. Відмінності значення сусідніх пікселів повинні бути менше, ніж граничне значення, що зменшується зі збільшенням якості. Коли розмір кроку квантування дуже малий, фільтр відключається повністю. Кодер може також відключити фільтр явно або налаштувати його потужність на рівні зрізу.

2.2 Функціональна реалізація кодека стандарту H.264

2.2.1 H.264 в бібліотеці Intel IPP

Частина компенсації руху з найбільш інтенсивними обчисленнями в H.264 генерує опорні блоки. Оскільки H.264 дозволяє субпіксельні зсуви, реалізація повинна використовувати конкретний фільтр інтерполяції для розрахунку блоків.

Бібліотека Intel IPP визначає набір функцій інтерполяції для обробки інтерполяції в різних місцях зображення. Є такі функції:

- ippiInterpolateLuma_H264_ [8U | 16U] _C1R
- ippiInterpolateLumaTop_H264_ [8U | 16U] _C1R
- ippiInterpolateLumaBottom_H264_ [8U | 16U] _C1R
- ippiInterpolateLumaBlock_H264_ [8U | 16U] _C1R
- ippiInterpolateChroma_H264_ [8U | 16U] _C1R
- ippiInterpolateChromaTop_H264_ [8U | 16U] _C1R
- ippiInterpolateChromaBottom_H264_ [8U | 16U] _C1R
- ippiInterpolateChromaBlock_H264_ [8U | 16U] _C1R

Вони поділяються на функції обробки яскравості та кольоровості. Вони також діляться на функції, які обробляють блоки, для яких всі дані присутні і ті, які відбуваються на межі кадру, поза якою не існує ніяких даних.

Функції, які обробляють всі блоки не на краю кадру, функції **ippiInterpolateLuma_H264** і **ippiInterpolateChroma_H264**, не вважаються невід'ємною частина векторів руху. Вони лише виконують інтерполяцію. Вхідний покажчик для опорних даних вже повинен вказувати на інтегральне зміщення опорного блоку. Функції розраховують інтерпольований опорний блок, використовуючи 2 або 3 біти, що визначають вектор руху з роздільною здатністю в чверть- або восьму частку пікселя.

З інших функцій, ті, що з **Top** або **Bottom** в імені функції, використовуються для інтерполяції даних на межі зображення. Параметри дають їм інформацію, як далеко за межами зображення знаходиться опорний блок. Функція генерує дані за неіснуючими межами за рахунок повтору меж рядків, а потім виконує інтерполяцію, як звичайно.

Інші типи функцій – ті, які з **Block** в імені функції - виконують інтерполяцію опорного блоку повністю в межах зображення, але також беруть весь вектор руху так, що він може вплинути на розрахунок зміщення. Лістинг 1 показує ці функції в дії.

Функція **SelectPredictionMethod** визначає, чи потрібно алгоритму використовувати межеві версії функцій.

Основна частина функції готує всі аргументи для функцій інтерполяції. Змінні **mvx** і **mvy** зберігають повні вектори руху. Приведений код встановлює змінним **xh** і **yh** значення нижчих розрядів вектора руху, його дробової частини. Потім, після відсікання векторів руху максимальним діапазоном, код встановлює значення змінних **xint** і **yint** до інтегральної частини вектора руху. Нарешті, він обчислює значення зміщення опорного блоку і викликає відповідну функцію Intel IPP.

2.2.2 Intra Prediction

Бібліотека Intel IPP має три функції для прогнозування стосовно внутрішніх блоків. Це:

ippiPredictIntra_4x4_H264_8u_C1IR для 4x4 блоків,

ippiPredictIntra_16x16_H264_8u_C1IR для 16x16 блоків,

ippiPredictIntraChroma8x8_H264_8u_C1IR для кольоровості блоків.

Ці функції приймають в якості аргументів покажчик на місце старту блоку і значення кроку буфера в режимі передбачення, як в лістингу 2, і набір прапорів, що вказують, які блоки даних зверху або зліва доступні. Лістинг 2 використовує ці функції для передбачення.

Є три можливих шляхи в цьому коді: 16x16, 8x8 і 4x4. Блоки 16x16 негайно викликають **ippiPredictIntra**, блоки 8x8 **AddResidualAndPredict8x8**, а блоки 4x4 **AddResidualAndPredict**. Менші блоки включають в себе багато видів меж з іншими блоками і цикл всередині макроблоку. З цих функцій, тільки 4x4 версії показано. Версія 8x8 майже ідентичний.

Ці функції прогнозування використовують певний алгоритм зі стандарту для розрахунку опорного блоку з попередніх блоків. Режим роботи визначає напрямок даних, що представляють інтерес, а потім алгоритм обчислює передбачення для кожного пікселя на основі середнього з одного або більш доступних пікселів в цьому напрямку.

Цей код вибирає режим, вже підібраний в іншому місці, в якості аргументу. Так, основна частина коду присвячена визначенню, які з зовнішніх контрольних блоків доступні, і розрахунку місця розташування блоку в пам'яті. Межеві блоки доступні, якщо передбачений блок не знаходиться на цій межі з іншим макроблоком, або якщо змінна **edge_type** не показує, що цей макроблок знаходиться на межі фрейму.

2.2.3 Перетворення і квантування

У функціях Intel IPP перетворення та функціональні квантування об'єднані для більшої ефективності. Є чотири функції для декодування H.264:

- ippiTransformDequantLumaDC_H264_16s_C1I
- ippiTransformDequantChromaDC_H264_16s_C1I
- ippiDequantTransformResidual_H264_16s_C1I
- ippiDequantTransformResidualAndAdd_H264_16s_C1I

Є аналогічні функції для кодування:

- ippiTransformQuantLumaDC_H264_16s_C1I
- ippiTransformQuantChromaDC_H264_16s_C1I
- ippiTransformQuantResidual_H264_16s_C1I

Лістинг 3 містить код, який використовує ці функції.

Змінна **cbp4x4** - це бітова маска, яка вказує, чи містять коефіцієнти DC всередині макроблоку які-небудь дані, і чи має кожен AC блок в макроблоках які-небудь дані. Змінна **QP** зберігає параметр, який визначає ступінь квантування. Якщо бітова маска вказує, що є які-небудь дані з яскравості, код перетворює їх за допомогою функції **ippiTransformDequantLumaDC**. Потім код перебирає 16 блоків в макроблоках. Для кожного блоку, якщо в ньому є залишкові дані, виконується перетворення і деквантування.

2.2.4 Розпаралелювання і кодування відео

H.264 і MPEG-4 в цілому піддаються розпаралелюванню. Лістинг 4 показує ключовий шматок коду з кодека зразка Intel IPP для H.264, який використовує OpenMP для розпаралелювання роботи кодеру. Ключовим аспектом цього коду є сегмент. Цей сегмент визначається як незалежна частина зображення, яка не використовує для прогнозування сегменти відео, що вже використовуються іншими сегментами. Це робить його ідеальним для розпаралелювання, а кодек може обробляти декілька таких сегментів одночасно і не буде змушений переходити в однопотоковий режим через компенсацію руху.

2.3 Висновки до розділу 2

В даному розділі розглянуто структуру типового відеокодека – режими його роботи, технології оцінки та передбачення руху, процеси перетворення, квантування та деблокування потоку, ентропійне кодування, процес реорганізації і т.д. Також приведено стандартну реалізацію кодека, засновану на вільній бібліотеці Intel IPP, та розібрано основні функції, необхідні для реалізації кодека на даній бібліотеці.

3. РЕАЛІЗАЦІЇ ВІДЕОКОДЕКІВ

3.1 Реалізації стандарту H.264

3.1.1 OpenH264

Cisco зробили свою реалізацію стандарту H.264 з відкритим вихідним кодом під ліцензією BSD. Розробка та підтримка будуть продовжуватись як професійною командою, так і спільнотою. Крім того, він надається у бінарному вигляді, придатному для включення в програми через ряд різних операційних систем, а при бажанні модуль для власної збірки доступний для скачування з Інтернету. Компанія не буде витрачатися на ліцензування в MPEG-LA - на основі нинішніх умов ліцензування буде ефективніше зробити H.264 безкоштовним для використання на підтримуваних платформах.

3.1.2 x264

x264 - це вільна бібліотека програмних компонентів для кодування відеопотоків H.264 з відкритим програмним кодом. Код цієї бібліотеки був написаний з нуля. Ряд незалежних порівнянь показує, що він є одним з кращих кодеків даного стандарту. Як не дивно, розробники не роблять офіційних закінчених версій (релізів), тобто кодек весь час змінюється. Нові версії вихідних текстів з'являються мало не кожен день, тому іноді буває важко встежити за змінами, що відбуваються. Ця реалізація заснована на принципах GNU, однак ця ліцензія може бути несумісна з патентної ліцензією MPEG LA щодо розуміння патентів на програмне забезпечення.

3.2 Реалізації конкурентів стандарту H.264

3.2.1 VP8

Розробником кодека VP8 є компанія On2 Technologies, яку на початку 2010 року купила компанія Google. На конференції I/O Google оголосила про відкриття початкових кодів VP8 і запропонувала використовувати його в стандарті HTML5 за замовчуванням для програвання відео.

Фахівці показали, що відеокодек VP8 перевершує відеокодеки стандарту H.264 Після проведення повноцінного дослідження та порівняння відеокодеків стандарту H.264 і VP8 в різних режимах відеозв'язку, фахівці компанії заявляють що відеокодек VP8 показує відмінні результати роботи на високих бітрейтах, перевершуючи за цим характеристикам кращі кодеки стандарту H.264. За результатами дослідження було вирішено впровадити кодек VP8 в продукти компанії кодеком за замовчуванням, вибираним автоматично для використання на Інтернет-каналах, що мають високу пропускну здатність.

3.2.2 HEVC (H.265)

HEVC є наступним поколінням стандарту стиснення відеосигналів, що йде на зміну використовуваному H.264 / MPEG-4 AVC. Новий стандарт дозволяє вдвічі знижувати обсяг переданих даних або вдвічі підвищувати швидкість реальної передачі, а також підтримка дозволу 8192x4320. Високоефективне відеокодування, а саме так розшифровується назва HEVC - High Efficiency Video Coding, є спадкоємцем популярного сьогодні кодека H.264, був спільно розроблений експертними групами ISO / IEC Moving Picture Experts Group і ITU-T Video Coding Experts Group.

HEVC (H.265) обіцяє можливість передачі відео дозволом 4K з 60 к/с по каналах з пропускною здатністю в 10 Мб/с. У порівнянні зі своїм попередником H.264 / MPEG-4 AVC, новий кодек використовує для стиснення відео більш досконалі алгоритми, що в свою чергу сприятливо позначається не тільки на візуальному сприйнятті відео, але і в кращу сторону відбивається на апаратних можливості нового стандарту HEVC.

3.3 Власна реалізація

Попередні специфікації MPEG дозволяли кодувати фрейми як I-, P- або B-кадри. H.264 є більш складним – він дозволяю кодувати окремі фрейми у вигляді декількох сегментів, кожен з яких може бути I, P, або B типу, і навіть більше. Цей функціонал можна використана для досягнення різних цілей. Проте у моїй першій реалізації для простоти буде використано лише один сегмент на кадр, і тільки I-кадри (тобто кодек не буде стискати відеопотік).

Як і в попередніх специфікаціях MPEG, в H.264 кожен сегмент складається з одного або більше 16 × 16 макроблоків. Кожен макроблок в схемі вибірки 4:2:0 містить блок 16 × 16 зі зразком яскравості і два блоки 8 × 8 зі зразками кольоровості. Ці зразки будуть безпосередньо скопійовані у вихідний h264 файл покадрово.

Враховуючи специфікацію H.264 та все вищеописане, ми повинні виділити такі речі:

- набір параметрів послідовності (SPS): один на потік
- набір параметрів картинки (PPS): один на потік
- заголовок сегменту: в кожному фреймі

    заголовочна інформація

    заголовок макроблоку: в кожному макроблоці

    закодовані дані макроблоку: власне закодоване відео

SPS, PPS, і заголовок будуть однаковими для всіх створених в додатку файлів, оскільки вони включені в програму у вигляді послідовності бітів. Для перевірки працездатності додатку буде використано отриманий за допомогою ffmpeg файл у форматі YUV420p, роздільною здатністю 128×96 пікселів (хоча можна й більшої).

Програму потрібно запускати через командний рядок, де параметрами виступають: Типовий виклик програми виглядає так:

codec_v2.exe sample.yuv test.h264 128 96 25 16 9

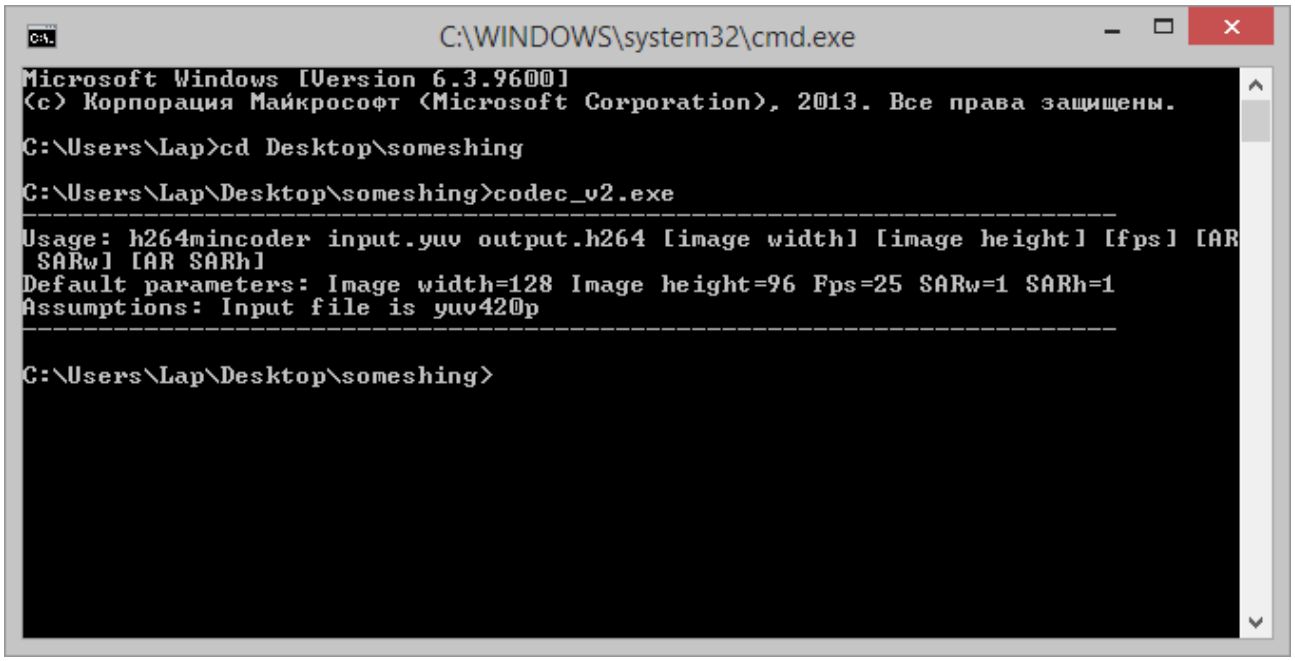

Рисунок 3.1 - Зовнішній вигляд додатку

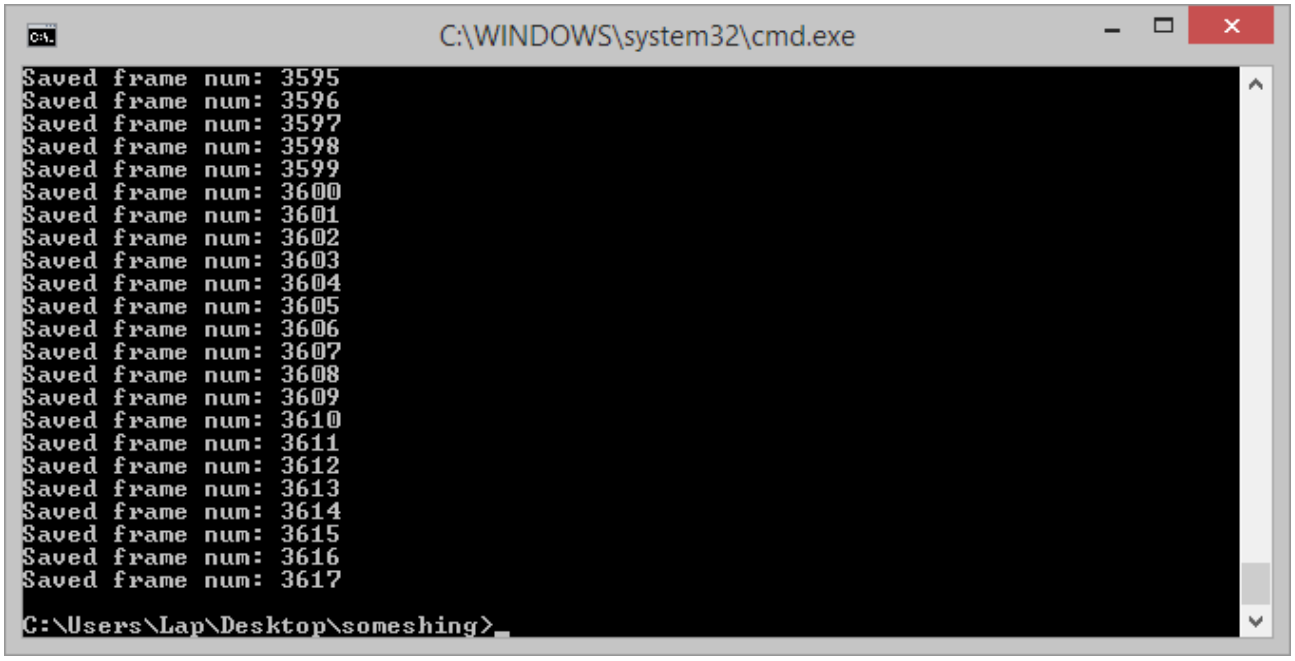

Рисунок 3.2 - Результат роботи програми

Лістинг коду наведено в додатку Б.

3.4 Висновок до розділу 3

В даному розділі було розглянуто існуючі реалізації як кодека стандарту H.264, так і його основних конкурентів, проведено оцінку їх позицій в сучасному світі кодування відео. Також саме тут знаходиться кульмінація дипломної роботи – самописна реалізація кодека.

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Вступ

Як зазначено в ст.1 Закону України «Про охорону праці», Охорона праці — це система правових, соціально-економічних, організаційно-технічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження здоров'я і працездатності людини в процесі трудової діяльності. Також в поняття охорони праці входять заходи, призначені для створення особливих полегшених умов праці для жінок і неповнолітніх, а також працівників зі зниженою працездатністю.

В даному розділі проводиться аналіз робочого приміщення на відповідність нормам охорони праці, оскільки необхідність забезпечити максимально сприятливі і безпечні умови праці має першочергове значення.

Комфортні та безпечні умови праці значно підвищують рівень ефективності. Крім того, іноді вони виступають важливим фактором при виборі робочого місця, тому роботодавець має бути зацікавлений в створенні сприятливих умов та застосуванні сучасних засобів безпеки для своїх підлеглих. Працівник має право відмовитись від роботи, поєднаної з небезпекою для життя або в умовах, що не відповідають нормам законодавства.

4.2 Аналіз умов праці

Робоче приміщення знаходиться на сьомому поверсі дев’ятиповерхової офісної будівлі. Вікна виходять на південний схід, площа засклення – 50%. Кімната має як природне, так і штучне освітлення. План-схема приміщення наведена на рис. 4.1

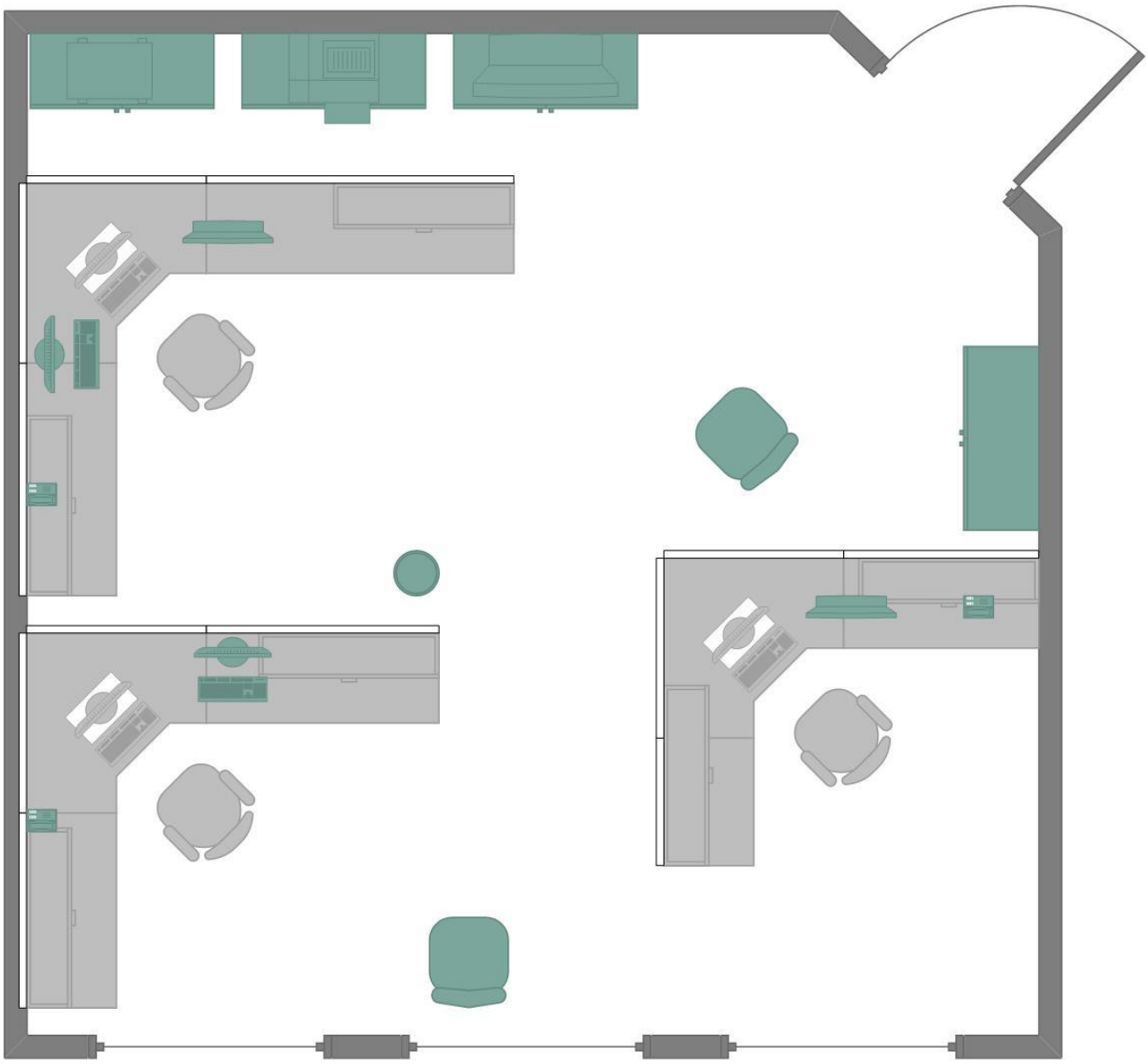

Рисунок. 4.1 – План-схема робочого приміщення

Характеристики приміщення:

- Довжина – a = 7 м;
- Ширина – b = 7 м;
- Висота стелі – h = 3.6 м;

- Площа – S = a * b = 7 * 7 = 49 м^2;
- Об’єм – V = S * h = 49 * 3.2 ≈ 156 м^3;

В приміщенні одночасно працюють три особи. Відповідно до ДСанПіН 3.3.2.007-98, площа повинна складати не менше 6 м^2 на людину, а об’єм – 20 м^3. Розрахуємо ці показники для нашого приміщення:

$$S' = \frac{S}{N} = \frac{49}{3} = 16.3(\frac{м^2}{люд.})$$

$$V' = \frac{V}{N} = \frac{156}{3} = 52(\frac{м^3}{люд.})$$

Отже, приміщення відповідає нормам з площі та об’єму.

Тепер розглянемо відповідність характеристик робочого місця нормативним. Для цього зведемо основні вимоги до організації робочого місця з фактичними значеннями до таблиці 4.1 та порівняємо їх:

Таблиця 4.1 – Параметри організації робочих місць

Найменування параметра	Значення	
	фактичне	нормативне
Висота робочої поверхні, мм	750	680 ÷ 800
Висота простору для ніг, мм	730	не менше 600
Ширина простору для ніг, мм	>1000	не менше 500
Глибина простору для ніг, мм	>1000	не менше 650
Висота поверхні сидіння, мм	450	400 ÷ 500
Ширина сидіння, мм	500	не менше 400
Глибина сидіння, мм	480	не менше 400
Висота поверхні спинки, мм	600	не менше 300
Ширина опорної поверхні спинки, мм	480	не менше 380
Радіус кривини спинки в горизонтальній площині, мм	400	400
Відстань від очей до екрану дисплея, мм	800	700 ÷ 800

Також відстань між моніторами сусідніх робочих місць більша за 2,5 м, а ширина проходів перевищує 1 м, тому можна зробити висновок, що організація робочих місць виконана з урахуванням усіх норм.

4.3 Аналіз мікрокліматичних умов

Мікроклімат робочих приміщень – це клімат внутрішнього середовища, що визначається діючим на організм людини поєднанням температури, вологості та швидкості переміщення повітря. Він має значний вплив на стан організму працівників та їх працездатність.

Параметри, за якими оцінюється мікроклімат, встановлюється відповідно до пори роки і категорії роботи. Робота оператора ПК, яка розглядається, виконується сидячи і не потребує фізичного напруження: витрати енергії становлять до 120 ккал/год, відповідно вона відноситься до категорії легка - 1а.

У таблиці 4.2 наведені оптимальні значення параметрів мікроклімату для категорії робіт 1а, а також фактичні значення цих параметрів у досліджуваному приміщенні.

Таблиця 4.2 – Параметри мікроклімату на робочому місці

Період року	Параметр	Оптимальний	Фактичний
Теплий	Температура	23 - 25 °С	22 - 26 °С
	Вологість	40 - 60 %	35 %
	Швидкість повітря	≤ 0.1 м/с	
Холодний	Температура	22 - 24 °С	21 - 23 °С
	Вологість	40 - 60 %	45 %
	Швидкість повітря	≤ 0.1 м/с	

Всі показники задовольняють вимогам для робіт категорії 1а і є задовільними для здоров’я людини.

Слід зазначити, що у приміщеннях з ПК рекомендується дотримуватися саме оптимальних параметрів мікроклімату, тобто таких, які забезпечують відчуття теплового комфорту та створюють передумови для високого рівня працездатності.

4.4 Аналіз освітлення

Для приміщень, в яких робочі місця обладнані ПК, важливо організувати правильні умови освітлення. Нормування умов освітлення здійснюється згідно будівельних норм ДБН В.2.5-28-2006.

Кабінет має як природне, так і штучне освітлення. На південно-східній стіні розташовано 3 вікна шириною 1,8 м і висотою 2 м, відповідно площа кожного вікна – 3,6 $м^2$. Вікна мають регульовані пристрої для відкривання та обладнані жалюзями з можливістю захисту персоналу від прямих сонячних променів і регулювання рівня освітленості в приміщенні.

Стіни обклеєні світлими шпалерами, стеля переважно білого кольору, у якості підлогового покриття використано темний ковролін з коефіцієнтом відбиття 0,2. Поверхня підлоги рівна, неслизька, з антистатичними властивостями. Відблискування поверхонь обмежується за рахунок правильного вибору світильників та розташування робочих місць відносно джерел освітлення. Яскравість відблисків на сучасних моніторах не перевищує 32 $кд/м^2$.

В приміщенні використовується система загального рівномірного штучного освітлення. Мається два ряди світильників Л201Б 2х40-0.3, у кожному з яких знаходиться по дві люмінесцентні лампи типу ЛБ-40. Їх технічні характеристики:

- потужність – 40 Вт;
- напруга на лампі – 103 В;
- світловий потік: номінальний – 3120 лм,
  мінімальний – 2810 лм;
- довжина лампи: без штирків – 1199.4 мм,
  із штирками – 1213.6 мм;
- діаметр – 40 мм.

План освітлення наведено на рисунку 4.2:

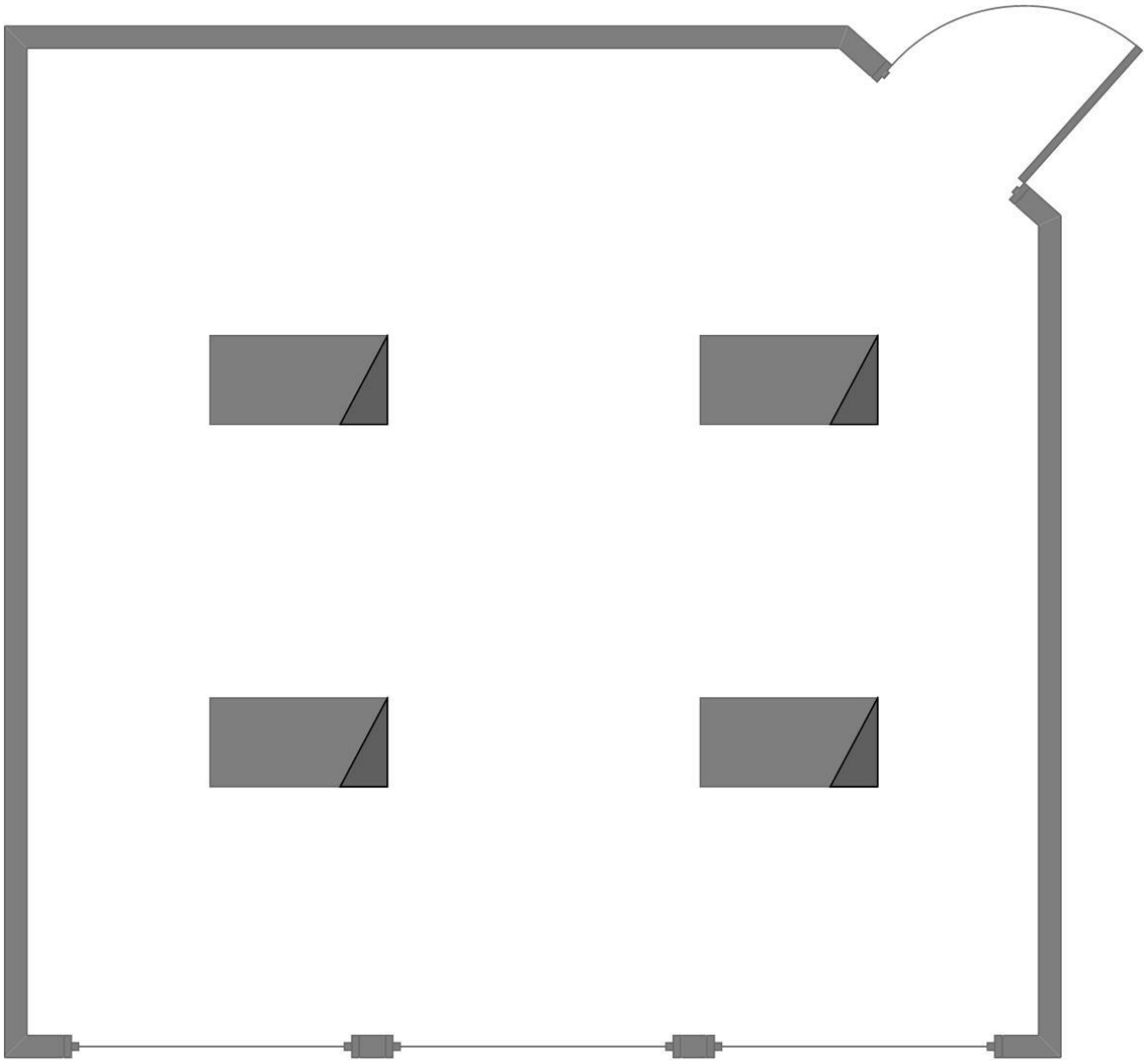

Рис. 4.2 – План освітлення

Освітлення рівномірно розподілено в приміщенні завдяки комбінації штучного та природнього освітлення, різкі тіні уникаються, а системи тонкого регулювання забезпечують такий стан протягом усього дня. Чистка вікон та світильників проводиться не рідше двох разів на рік, а лампи, що перегорають, замінюють по мірі їх виходу з ладу.

Доречно нагадати, що для збереження зору працівників у хорошому стані необхідно робити невеликі перерви, щоб дати очам відпочити, а також дотримуватися певної відстані, на якій працівники повинні перебувати від комп'ютера.

4.5 Аналіз шуму в робочому приміщенні

Шум може негативно впливати на працездатність працюючих. Основними фізичними параметрами звуку є інтенсивність, звуковий тиск і частота коливань. Нормування шумів, ультра- та інфразвуків здійснюється згідно ДСН 3.3.6.037-99. Відповідно до цих норм, рівень шуму не має перевищувати 50 дБ.

Можливий список джерел шуму у нашому приміщенні:

- система охолодження ПК
- принтер під час операцій друку
- шум працюючого кондиціонеру
- система охолодження джерела безперебійного живлення

Сумарний рівень інтенсивності звуку можна розрахувати за формулою:

$$L = 10 \cdot lg\left(\frac{1}{T} \cdot \sum_{i=1}^{n} t_i \cdot 10^{0.1 \cdot L_i} \right)$$

де

T – робочий час протягом дня;

t_i – час надходження звуку від і-го джерела;

L_i – рівень звукового тиску і-го джерела.

В таблиці 4.3 наведені джерела шуму, рівень звукового тиску та час дії протягом робочого дня

Таблиця 4.3 – Джерела шуму на робочому місці

Джерело шуму	Рівень шуму L, дБА	Час дії шуму t, год	Кількість джерел шуму N
Кондиціонер	32	6	1
Матричний принтер	55	0.5	1
Система охолодження ПК	40	8	3
Джерело безперебійного живлення	48	8	1

Визначимо $L_{екв.}$ еквівалентний рівень шуму за 8 робочих годин:

$$L_{\text{екв}} = 10 * \lg\left(\frac{1}{T}\sum t_i * 10^{0.1*Li}\right) = 10 * \lg(\frac{1}{8} * (6 * 10^{3.2} + 0{,}5 * 10^{5.5} + 3 * 8 * 10^{4} + 8 * 10^{4.8})) \approx 50 \text{ дБА}$$

Еквівалентний рівень шуму дорівнює максимально дозволеному нормативному, а отже рівень звукового тиску та рівень звуку на робочих місцях відповідає вимогам.

Для зниження шуму можна зробити віброізоляцію, яка здійснюється за допомогою спеціальної прокладки під системний блок, що послаблює передачу вібрацій робочого столу.

4.6 Електробезпека

Проаналізуємо приміщення на можливість ураження персоналу електричним струмом. Визначимо групу електронебезпечності даного приміщення. Ознаки підвищеної небезпеки ураження електрострумом:

- висока вологість;
- температура вища за 35 ºС;
- наявність струмопровідного пилу;
- струмопровідна підлога;
- можливість одночасного дотику до корпусів чи струмопровідних елементів та до елементів, що мають зв’язок з землею.

Ознаки особливої небезпеки ураження електрострумом:

- дуже висока вогкість;
- наявність хімічно активного середовища.

Наше приміщення не має жодної ознаки особливої або підвищеної небезпеки ураження персоналу струмом. Тому за групою електронебезпечності воно відноситься до приміщень без підвищеної небезпеки ураження струмом.

Розглянемо технічні рішення із запобігання електротравм від контакту зі струмопровідними елементами електроустаткування:

- всі струмопровідні елементи (в першу чергу електричні дроти) вкриті ізоляційними матеріалами;
- в джерелі безперебійного живлення використовується механічне захисне блокування, що забезпечує вимикання напруги при его відкриванні;
- електромережа в приміщенні розведена в спеціальних каналах міжстінного простору та підлоги.

Споживачі електроенергії в робочому приміщенні: 3 ПК, 7 моніторів, 1 принтер, 1 телевізор, 4 світильники, 1 кондиціонер та 1 джерело безперебійного живлення. Кожне робоче місце обладнане 3-ма розетками по 220 В, окремо існують розетки для кондиціонеру та джерела безперервного живлення. Всі прилади використовують саме цю напругу. Усі кабелі ізольовані. Заземлені конструкції захищені діелектричними сітками від випадкового дотику. Усе електроустаткування має апаратуру захисту від струму короткого замикання.

Лінія електромережі для живлення ПК та периферійних пристроїв виконується як окрема групова трьохпровідна мережа шляхом прокладання фазового, нульового робочого та нульового захисного провідників.

При виконанні робіт по ремонту і обслуговуванню ПК обслуговуючий персонал зобов'язаний керуватися «Правилами техніки безпеки при експлуатації електроустановок споживачами». До роботи не допускаються особи, які не пройшли навчання з техніки безпеки.

Можна зробити висновок, що даний кабінет задовольняє вимогам електробезпеки у приміщенні, в якому встановлені ПК, відображеним в НПАОП 0.00-1.28-10.

4.7 Пожежна безпека

В даному приміщенні наявні такі потенційні джерела пожежної небезпеки, як ПК, світильники, принтер, кондиціонер. Також тут присутні меблі з горючих і легкозаймистих матеріалів (ДСП, дерево, пластмаса, синтетичні тканини), папір, тканеві жалюзі. Застосовуються дроти з важкогорючою і негорючою ізоляцією. В робочому приміщенні відсутні горючі рідини, пил та волокна не виділяються, а отже, згідно з НАПБ Б.03.002-2007 дане приміщення можна віднести до зони П-IIа і до категорії пожежної небезпеки В.

Відповідно до списку потенційних джерел пожежної небезпеки, наведеного вище, пожежа може виникнути лише внаслідок порушення правил пожежної безпеки з боку працівників: паління в кабінеті, використання побутових нагрівачів, тощо.

Для уникнення такої ситуації, кожен працівник має проходити первинний і повторні інструктажі з правил пожежної безпеки згідно чинних нормативних актів, а також в приміщенні потрібно розміщувати пам’ятку у вигляді витягу з правил пожежної безпеки для постійного нагадування останніх працівникам.

Крім того, слід використовувати такий комплекс заходів:

- наявність системи автоматичної пожежної сигналізації з димовими пожежними оповіщувачами;
- ступінь вогнестійкості будівлі, у якій розташовано приміщення – II;
- наявність шляхів евакуації при виникненні пожежі;
- розміщення схеми евакуації людей при пожежі і ознайомлення з нею персоналу.

Згідно з ППБУ, в приміщенні, що аналізується, мають бути наявні:

1. Переносні засоби пожежогасіння: по одному вуглекислотному вогнегаснику з величиною заряду вогнегасної речовини 3 кг і більше на кожні 20 $м^2$ площі приміщення, де використовується ПК. В нашому випадку, це має бути як мінімум 3 вогнегасники марки ВВ-5 (величина заряду вогнегасної речовини 3,5 кг)

2. План евакуації з приміщення

3. Система протипожежних датчиків, пожежна сигналізація

Приміщення має один вихід, оскільки в ньому працює менше 25 чоловік. Ширина проходу між робочими місцями у приміщенні перевищує 1 м. Сходова клітка має природне освітлення в комбінації зі штучним. Сходи та приміщення обладнані системою аварійного освітлення. Співробітники ознайомлені з порядком і планом евакуації.

Отже, шляхи евакуації з приміщення повністю відповідають нормам.

4.8 Рекомендації щодо поліпшення умов праці

У зв’язку зі специфікою робіт з ПК можна порекомендувати виконання комплексів вправ для психічного та психологічного розвантаження.

При інтенсивній роботі з вхідними даними, редагуванні програм, читанні інформації з екрану монітора безперервна тривалість роботи не повинна перевищувати 4-х годин (при 8-годинному робочому дні). Задля зниження напруженості праці необхідно, якщо це можливо, рівномірно розподіляти навантаження і раціонально чергувати характер діяльності.

Варто щогодини робити перерву на 15 хвилин. Один або кілька разів у годину необхідно виконувати серію легких вправ на розтягування, що можуть зменшити напругу, накопичену в м’язах при тривалій роботі за комп’ютером. Рекомендується робити вправи для м’яз очей, оскільки тривала робота за комп’ютером може значно погіршити його стан.

З інших рекомендацій щодо поліпшення умов праці відповідно до можна навести наступні:

- у приміщенні слід щоденно проводити вологе прибирання;
- у приміщенні повинні бути медичні аптечки першої допомоги.

4.9 Висновки

У даному розділі були розглянуті умови праці для виробничого приміщення, у якому операторами експлуатується програмний продукт. Розміри приміщення та параметри робочих місць відповідають нормам чинного законодавства з охорони праці.

Приведені рекомендації щодо організації робочого місця на підприємстві дозволяють підвищити рівень безпеки праці, попередити виникнення надзвичайних ситуацій та надати першу медичну допомогу при виникненні надзвичайної ситуації. Служби охорони праці, а саме відповідні служби і структурні підрозділи підприємства повинні здійснювати постійний контроль за виконанням робіт у відповідності з вимогами з охорони праці, електро-, газо- і пожежобезпеки, не допускати до роботи осіб, які не пройшли інструктаж та не здали заліки по питаннях охорони праці.

Було проведено аналіз шкідливих та небезпечних виробничих чинників, наявних у даному приміщенні. Значення параметрів мікроклімату, виробниче освітлення та засоби пожежної безпеки приміщення відповідають необхідним нормативам.

ВИСНОВКИ

Світ розвивається дуже швидко, і утримання технологій на належному рівні – важка та важлива задача. Сучасні технічні спеціалісти повинні якнайкраще намагатися йти в ногу з часом, для чого їм потрібно слідкувати за стандартами та тенденціями.

Дана дипломна робота присвячена аналізу, моделюванню та реалізації відеокодека стандарту H.264.

В першому розділі детально розглянуто, власне, поняття відеокодека, основні принципи їх роботи, головні алгоритми.

Другий розділ містить детальний структурний опис відеокодека, розглянуто режими його роботи, різноманітні технології оцінки та передбачення руху, процеси перетворення, квантування та деблокування потоку, ентропійне кодування, процес реорганізації і т.д. Особливу увагу присвячено реалізацію кодека на базі бібліотеки Intel IPP.

У третьому розділі було розглянуто існуючі реалізації як кодека стандарту H.264, так і його основних конкурентів, проведено оцінку їх позицій в сучасному світі кодування відео. Також була розроблена та протестована власна базова реалізація кодека стандарту H.264.

Четвертий розділ присвячено охороні праці на виробництві.

Перспективою для подальшого розвитку проекту є покращення програмного додатку в бік повноцінної реалізації – з подальшою оптимізацією, додаванням функціоналу та графічного інтерфейсу. Також можливе використання додатку в якості наочного тренажеру для студентів, які захоплюються відеокодуванням.

Мною в процесі написання дипломної роботи було отримано багато незамінних знань, корисного досвіду та задоволення від цікавого роду зайнятості.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. King N. Ngan. Advanced Video Coding Principles & Techniques / University of Western Australia, 1999. – 431 pages

2. Richardson E.G. Video Codec Design / John Wiley & Sons LTD, 2002. – 313 pages

3. Borko Furht. Real-Time video compression / Kluwer Academic publishers, 1997. – 172 pages

4. Iain E. Richardson. The H.264 Advanced Image Compression / John Wiley & Sons LTD, 2003. – 348 pages

5. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98 (затверджено Постановою Головного державного санітарного лікаря України від 10.12.1998 р. №7).

6. Правила охорони праці під час експлуатації електронно-обчислювальних машин. НПАОП 0.00-1.28-10

7. Санітарні норми мікроклімату виробничих приміщень : ДСН 3.3.6.042-99– К. : МОЗ України, 2000. – 42 с. – (Національні стандарти України).

8. Природнє і штучне освітлення : ДБН В.2.5-28:2006 — К. : Міністерство будівництва, архітектури та житлово-комунального господарства України, 2006. – 68 с.

9. Санітарні норми виробничого шуму, ультразвуку та інфразвуку : ДСН 3.3.6.037-99.– К. : МОЗ України, 2000. – 37 с. – (Національні стандарти України).

ДОДАТОК А

ЛІСТИНГ КОДУ, НАПИСАНОГО З ВИКОРИСТАННЯМ БІБЛІОТЕКИ INTEL IPP

```
inline
Ipp8s SelectPredictionMethod(Ipp32s MBYoffset,Ipp32s mvy,
  Ipp32s sbheight,Ipp32s height)
{
  Ipp32s padded_y = (mvy&3)>0?3:0;
  mvy>>=2;
  if (mvy-padded_y+MBYoffset<0)
  {
    return PREDICTION_FROM_TOP;
  }
  if (mvy+padded_y+MBYoffset+sbheight>=height)
  {
    return PREDICTION_FROM_BOTTOM;
  }

  return ALLOK;
}

{
  ...
  // set pointers for this subblock
  pMV_sb = pMV + (xpos>>2) + (ypos>>2)*4;
  mvx = pMV_sb->mvx;
  mvy = pMV_sb->mvy;
   ...
   xh = mvx & (INTERP_FACTOR-1);
  yh = mvy & (INTERP_FACTOR-1);

  Ipp8u pred_method = 0;
  if (ABS(mvy) < (13 << INTERP_SHIFT))
  {
    if (is_need_check_expand)
    {
      pred_method = SelectPredictionMethod(
        mbYOffset+ypos,
        mvy,
        roi.height,
        height);
    }
  } else {
    pred_method = SelectPredictionMethod(
      mbYOffset+ypos,
      mvy,
      roi.height,
      height);

    mvy = MIN(mvy, (height - ((Ipp32s)mbYOffset + ypos +
      roi.height -
      1 - D_MV_CLIP_LIMIT))*INTERP_FACTOR);
    mvy = MAX(mvy, -((Ipp32s)(mbYOffset + ypos +
      D_MV_CLIP_LIMIT)*INTERP_FACTOR));
  }

  if (ABS(mvx) > (D_MV_CLIP_LIMIT << INTERP_SHIFT))
```

```
    {
      mvx = MIN(mvx, (width - ((Ipp32s)mbXOffset + xpos +
        roi.width -
        1 - D_MV_CLIP_LIMIT))*INTERP_FACTOR);
      mvx = MAX(mvx, -((Ipp32s)(mbXOffset + xpos +
        D_MV_CLIP_LIMIT)*INTERP_FACTOR));
    }


    mvyc = mvy;

    xint = mvx >> INTERP_SHIFT;
    yint = mvy >> INTERP_SHIFT;

    pRef = pRefY_sb + xint + yint * pitch;
    switch(pred_method)
    {
      case ALLOK:
        ippiInterpolateLuma_H264_8u_C1R(pRef, pitch,
          pTmpY, nTmpPitch,
          xh, yh, roi);
        break;
      case PREDICTION_FROM_TOP:
        ippiInterpolateLumaTop_H264_8u_C1R(pRef, pitch,
          pTmpY, nTmpPitch,
          xh, yh, - ((Ipp32s)mbYOffset+ypos+yint),roi);
        break;
      case PREDICTION_FROM_BOTTOM:
        ippiInterpolateLumaBottom_H264_8u_C1R(pRef, pitch,
          pTmpY, nTmpPitch,
          xh, yh, ((Ipp32s)mbYOffset+ypos+yint+roi.height)-
          height,roi);
        break;

      default:VM_ASSERT(0);
          break;
    }
  }
```

Лістинг 1 - Рамки для інтерполяції в H.264

```
  void AddResidualAndPredict(Ipp16s ** luma_ac,
      Ipp8u * pSrcDstPlane,
      Ipp32u step,
      Ipp32u cbp4x4,
      const IppIntra4x4PredMode_H264 *pMBIntraTypes,
      Ipp32s edge_type,
      bool   is_half,
      Ipp32s bit_depth)

  {
    Ipp32s srcDstStep = step;
    Ipp8u * pTmpDst = pSrcDstPlane;

    /* bit var to isolate cbp for block being decoded */
    Ipp32u uCBPMask = (1 << IPPVC_CBP_1ST_LUMA_AC_BITPOS);
    for (Ipp32s uBlock = 0; uBlock < (is_half ? 8 : 16);
         uBlock++, uCBPMask <<= 1)
    {
      pTmpDst = pSrcDstPlane;
      Ipp32s left_edge_subblock = left_edge_tab16[uBlock];
      Ipp32s top_edge_subblock = top_edge_tab16[uBlock];
```

```
      Ipp32s top = top_edge_subblock  &&
        (edge_type & IPPVC_TOP_EDGE);
      Ipp32s left = left_edge_subblock &&
        (edge_type & IPPVC_LEFT_EDGE);
      Ipp32s top_left = ((top || left) && (uBlock != 0)) ||
        ((edge_type & IPPVC_TOP_LEFT_EDGE) && (uBlock == 0));
      Ipp32s top_right = (top && (uBlock != 5)) ||
        (!above_right_avail_4x4[uBlock]) ||
        ((edge_type & IPPVC_TOP_RIGHT_EDGE) && (uBlock == 5));
       Ipp32s avail = (left == 0)*IPP_LEFT +
        (top_left == 0)*IPP_UPPER_LEFT +
        (top_right == 0)*IPP_UPPER_RIGHT +
        (top == 0)*IPP_UPPER;

      ippiPredictIntra_4x4_H264_8u_C1IR(pTmpDst,
          srcDstStep, pMBIntraTypes[uBlock], avail);

      if ((cbp4x4 & uCBPMask) != 0)
      {
        const Ipp8u * pTmp = pSrcDstPlane;
        ippiMC4x4_8u_C1(pTmp, srcDstStep, *luma_ac, 8,
          pSrcDstPlane, srcDstStep, IPPVC_MC_APX_FF, 0);

        *luma_ac += 16;
      }
      pSrcDstPlane +=
        xyoff[uBlock][0] + xyoff[uBlock][1]*srcDstStep;
    }
  }

  {
   ...

  Ipp32s availability =
    ((edge_type & IPPVC_LEFT_EDGE) == 0)*IPP_LEFT +
    ((edge_type & IPPVC_TOP_LEFT_EDGE) == 0)*IPP_UPPER_LEFT +
    ((edge_type & IPPVC_TOP_RIGHT_EDGE) == 0)*IPP_UPPER_RIGHT +
    ((edge_type & IPPVC_TOP_EDGE) == 0)*IPP_UPPER;

  if (mbtype == MBTYPE_INTRA_16x16)
  {
    ippiPredictIntra_16x16(
      context->pYPlane + offsetY,
      rec_pitch_luma,
      (IppIntra16x16PredMode_H264) pMBIntraTypes[0],
      availability);

    if (luma_ac)
      AddResidual(luma_ac,
        context->pYPlane + offsetY,
        rec_pitch_luma,
        sd->m_cur_mb.LocalMacroblockInfo->cbp4x4_luma,
        sd->bit_depth_luma);
  }
  else // if (intra16x16)
  {
    if (is_high_profile)
    {
      switch (special_MBAFF_case)
      {
        default:
          if (pGetMB8x8TSFlag(sd->m_cur_mb.GlobalMacroblockInfo))
          {
            AddResidualAndPredict_8x8(
```

```
89                &luma_ac,
90                context->pYPlane + offsetY,
91                rec_pitch_luma,
92                sd->m_cur_mb.LocalMacroblockInfo->cbp,
93                (IppIntra8x8PredMode_H264 *) pMBIntraTypes,
94                edge_type_2t,
95                true,
96                sd->bit_depth_luma);
97              AddResidualAndPredict_8x8(
98                &luma_ac,
99                context->pYPlane + offsetY + 8*rec_pitch_luma,
100               rec_pitch_luma,
101               sd->m_cur_mb.LocalMacroblockInfo->cbp >> 2,
102               (IppIntra8x8PredMode_H264 *) pMBIntraTypes + 2,
103               edge_type_2b,
104               true,
105               sd->bit_depth_luma);
106           }
107           else
108           {
109             AddResidualAndPredict(
110               &luma_ac,
111               context->pYPlane + offsetY,
112               rec_pitch_luma,
113               sd->m_cur_mb.LocalMacroblockInfo->cbp4x4_luma,
114               (IppIntra4x4PredMode_H264 *) pMBIntraTypes,
115               edge_type_2t,
116               true,
117               sd->bit_depth_luma);
118             AddResidualAndPredict(
119               &luma_ac,
120               context->pYPlane + offsetY + 8*rec_pitch_luma,
121               rec_pitch_luma,
122               sd->m_cur_mb.LocalMacroblockInfo->cbp4x4_luma >> 8,
123               (IppIntra4x4PredMode_H264 *) pMBIntraTypes + 8,
124               edge_type_2b,
125               true,
126               sd->bit_depth_luma);
127           }
128           break;
129         case 0:
130           if (pGetMB8x8TSFlag(sd->m_cur_mb.GlobalMacroblockInfo))
131           {
132             AddResidualAndPredict_8x8(
133               &luma_ac,
134               context->pYPlane + offsetY,
135               rec_pitch_luma,
136               sd->m_cur_mb.LocalMacroblockInfo->cbp,
137               (IppIntra8x8PredMode_H264 *) pMBIntraTypes,
138               edge_type,
139               false,
140               sd->bit_depth_luma);
141           }
142           else
143           {
144             AddResidualAndPredict(
145               &luma_ac,
146               context->pYPlane + offsetY,
147               rec_pitch_luma,
148               sd->m_cur_mb.LocalMacroblockInfo->cbp4x4_luma,
149               (IppIntra4x4PredMode_H264 *) pMBIntraTypes,
150               edge_type,
151               false,
152
153
154
```

```
155                 sd->bit_depth_luma);
156             }
157             break;
158         }
159       }
160       else
161       {
162         switch (special_MBAFF_case)
163         {
164           default:
165             AddResidualAndPredict(
166               &luma_ac,
167               context->pYPlane + offsetY,
168               rec_pitch_luma,
169               sd->m_cur_mb.LocalMacroblockInfo->cbp4x4_luma,
170               (IppIntra4x4PredMode_H264 *) pMBIntraTypes,
171               edge_type_2t,
172               true,
173               sd->bit_depth_luma);
174             AddResidualAndPredict(
175               &luma_ac,
176               context->pYPlane + offsetY + 8*rec_pitch_luma,
177               rec_pitch_luma,
178               sd->m_cur_mb.LocalMacroblockInfo->cbp4x4_luma >> 8,
179               (IppIntra4x4PredMode_H264 *) pMBIntraTypes + 8,
180               edge_type_2b,
181               true,
182               sd->bit_depth_luma);
183             break;
184           case 0:
185             AddResidualAndPredict(
186               &luma_ac,
187               context->pYPlane + offsetY,
188               rec_pitch_luma,
189               sd->m_cur_mb.LocalMacroblockInfo->cbp4x4_luma,
190               (IppIntra4x4PredMode_H264 *) pMBIntraTypes,
191               edge_type,
192               false,
193               sd->bit_depth_luma);
194             break;
195         }
196       }
197     ...
198   }
```

Лістинг 2 – Intra Prediction в H.264

```
1   if ((cbp4x4 & (IPPVC_CBP_LUMA_AC | IPPVC_CBP_LUMA_DC)) != 0)
2   {
3     Ipp16s *pDC;
4     Ipp16s DCCoeff;
5
6     Ipp16s *tmpbuf;
7
8     /* bit var to isolate cbp for block being decoded */
9     Ipp32u uCBPMask = (1 << IPPVC_CBP_1ST_LUMA_AC_BITPOS);
10
11    if ((cbp4x4 & IPPVC_CBP_LUMA_DC) != 0)
12    {
13      luma_dc = (*ppSrcCoeff);
14      *ppSrcCoeff += 16;
15      ippiTransformDequantLumaDC_H264_16s_C1I(luma_dc, QP);
```

```
    }

    tmpbuf = 0;  /* init as no ac coeffs */
    pDC = 0;  /* init as no dc */

    ac_coeffs = pDstCoeff;

    for (Ipp32s uBlock = 0; uBlock < 16;
         uBlock++, uCBPMask <<= 1)
    {
      DCCoeff = (Ipp16s)luma_dc[block_subblock_mapping[uBlock]];
      if (DCCoeff != 0)
        pDC = &DCCoeff; /* dc coeff presents */

      if ((cbp4x4 & uCBPMask) != 0)
      {
        memcpy(pDstCoeff, *ppSrcCoeff, 16*sizeof(Ipp16s));
        tmpbuf = pDstCoeff;
        pDstCoeff += 16;
        *ppSrcCoeff += 16;
      }

      Ipp32s hasAC = tmpbuf != 0;
      if (tmpbuf || pDC)
      {
        if (!pDC)
        {
          if (tmpbuf)
          {
            if (dc_present)
              tmpbuf[0] = 0;
          }
        }
        else
        {
          if (!tmpbuf)
          {
            tmpbuf = pDstCoeff;
            pDstCoeff += 16;
            cbp4x4 |= uCBPMask;
          }
        }
        ippiDequantTransformResidual_H264_16s_C1I(tmpbuf, 8, pDC,
          hasAC, QP);
        tmpbuf = 0;
        pDC = 0;
      }
    }
  }
```

Лістинг 3: Перетворення і квантування в H.264

```
template <class PixType, class CoeffsType> Status
  H264CoreEncoder<PixType,CoeffsType>::CompressFrame(
    EnumPicCodType &    ePictureType,
    EnumPicClass   &    ePic_Class,
    MediaData*        dst)
{
  Status      status = UMC_OK;
  Ipp32s  slice;

  for (m_field_index=0;
```

```
    m_field_index <= (Ipp8u)
    (m_pCurrentFrame->m_PictureStructureForDec< FRM_STRUCTURE);
    m_field_index++)
  {
    ...

#if defined _OPENMP
     vm_thread_priority mainTreadPriority = vm_get_current_thread_priority();
#pragma omp parallel for private(slice)
#endif // _OPENMP
      for (slice = (Ipp32s)m_info.num_slices*m_field_index;
           slice < m_info.num_slices*(m_field_index+1);
           slice++)
      {
#if defined _OPENMP
        vm_set_current_thread_priority(mainTreadPriority);
#endif // _OPENMP

        UpdateRefPicList(m_Slices + slice,
          m_pCurrentFrame->GetRefPicLists(slice),
          m_SliceHeader, &m_ReorderInfoL0,
          &m_ReorderInfoL1);

        // Compress one slice
        if (m_is_cur_pic_afrm)
          m_Slices[slice].status =
            Compress_Slice_MBAFF(m_Slices + slice);
        else{
          m_Slices[slice].status =
            Compress_Slice(m_Slices + slice,
            slice == m_info.num_slices*m_field_index);
        }
      ...
      }
```

Лістинг 4 - Розпаралелювання кодеру H.264

ДОДАТОК Б

ЛІСТИНГ ВЛАСНОЇ РЕАЛІЗАЦІЇ КОДЕКА

h264simpleCoder.cpp

```
#include <iostream>
#include "CJOCh264encoder.h"
#pragma warning (disable : 4996)
using namespace std;

int main(int argc, char **argv)
{
	int nRc = 1;

	if (argc < 3)
	{
		puts("-------------------------------------------------------------------------
----");
		puts("Usage: h264mincoder input.yuv output.h264 [image width] [image
height] [fps] [AR SARw] [AR SARh]");
		puts("Default parameters: Image width=128 Image height=96 Fps=25 SARw=1
SARh=1");
		puts("Assumptions: Input file is yuv420p");
		puts("-------------------------------------------------------------------------
----");
		return nRc;
	}

	char szInputFile[512];
	char szOutputFile[512];
	int nImWidth = 128;
	int nImHeight = 96;
	int nFps = 25;
	int nSARw = 1;
	int nSARh = 1;

	//Get input file
	strncpy_s (szInputFile,argv[1],512);

	//Get output file
	strncpy_s (szOutputFile,argv[2],512);

	//Get image width
	if (argc > 3)
	{
		nImWidth = (int) atol (argv[3]);
		if (nImWidth == 0)
			puts ("Error reading image width input parameter!");
	}

	//Get image height
	if (argc > 4)
	{
		nImHeight = (int) atol (argv[4]);
		if (nImHeight == 0)
			puts ("Error reading image height input parameter!");
```

```
	}

	//Get fps
	if (argc > 5)
	{
		nFps = (int) atol (argv[5]);
		if (nFps == 0)
			puts ("Error reading FPS parameter!");
	}

	//Get SARw
	if (argc > 6)
	{
		nSARw = (int) atol (argv[6]);
		if (nSARw == 0)
			puts ("Error reading AR SARw input parameter!");
	}

	//Get SARh
	if (argc > 7)
	{
		nSARh = (int) atol (argv[7]);
		if (nSARh == 0)
			puts ("Error reading AR SARh input parameter!");
	}

	FILE *pfsrc = NULL;
	FILE *pfdst = NULL;

	pfsrc = fopen (szInputFile,"rb");
	if (pfsrc == NULL)
	{
		puts ("Error opening source file!");
		return nRc;
	}

	pfdst = fopen (szOutputFile,"wb");
	if (pfdst == NULL)
	{
		puts ("Error opening destination file!");
		return nRc;
	}

	try
	{
		//Instantiate the h264 coder
		CJOCh264encoder *ph264encoder = new CJOCh264encoder(pfdst);

		//Initialize the h264 coder with frame parameters
		ph264encoder-
>IniCoder(nImWidth,nImHeight,nFps,CJOCh264encoder::SAMPLE_FORMAT_YUV420p, nSARw,
nSARh);

		int nSavedFrames = 0;
		char szLog[256];

		//Iterate through all frames
		while (!feof(pfsrc))
		{
			//Get frame pointer to fill
			void *pFrame = ph264encoder->GetFramePtr ();
```

```
                        //Get the size allocated in pFrame
                        unsigned int nFrameSize = ph264encoder->GetFrameSize();

                        //Load frame from disk and load it into pFrame
                        size_t nreaded = fread (pFrame,1, nFrameSize, pfsrc);
                        if (nreaded != nFrameSize)
                        {
                              if (! feof(pfsrc))
                                    throw "Error: Reading frame!";
                        }
                        else
                        {
                              //Encode & save frame
                              ph264encoder->CodeAndSaveFrame();

                              //Get the number of saved frames
                              nSavedFrames = ph264encoder->GetSavedFrames();

                              //Show the number of saved / encoded frames iin console
                              sprintf(szLog,"Saved frame num: %d", nSavedFrames - 1);
                              puts (szLog);
                        }
                  }

                  //Close encoder
                  ph264encoder->CloseCoder();

                  //Set return code to 0
                  nRc = 0;
            }
            catch (const char *szErrorDesc)
            {
                  //Show the error description on console
                  puts (szErrorDesc);
            }

            //Close previously opened files
            if (pfsrc != NULL)
                  fclose (pfsrc);

            if (pfdst != NULL)
                  fclose (pfdst);

            return nRc;
      }
```

CJOCh264encoder.cpp

```
#include "CJOCh264encoder.h"

//Private functions

//Contructor
CJOCh264encoder::CJOCh264encoder(FILE *pOutFile):CJOCh264bitstream(pOutFile)
{
	m_lNumFramesAdded = 0;

	memset (&m_frame, 0, sizeof (frame_t));
	m_nFps = 25;
}

//Destructor
CJOCh264encoder::~CJOCh264encoder()
{
	free_video_src_frame ();
}

//Free the allocated video frame mem
void CJOCh264encoder::free_video_src_frame ()
{
	if (m_frame.yuv420pframe.pYCbCr != NULL)
		free (m_frame.yuv420pframe.pYCbCr);

	memset (&m_frame, 0, sizeof (frame_t));
}

//Alloc mem to store a video frame
void CJOCh264encoder::alloc_video_src_frame ()
{
	if (m_frame.yuv420pframe.pYCbCr != NULL)
		throw "Error: null values in frame";

	int nYsize = m_frame.nYwidth * m_frame.nYheight;
	int nCsize = m_frame.nCwidth * m_frame.nCheight;
	m_frame.nyuv420pframesize = nYsize + nCsize + nCsize;

	m_frame.yuv420pframe.pYCbCr = (unsigned char*) malloc (sizeof (unsigned char) *
m_frame.nyuv420pframesize);

	if (m_frame.yuv420pframe.pYCbCr == NULL)
		throw "Error: memory alloc";
}

//Creates and saves the NAL SPS (including VUI) (one per file)
void CJOCh264encoder::create_sps (int nImW, int nImH, int nMbW, int nMbH, int nFps, int
nSARw, int nSARh)
{
	add4bytesnoemulationprevention (0x000001); // NAL header
	addbits (0x0,1); // forbidden_bit
	addbits (0x3,2); // nal_ref_idc
	addbits (0x7,5); // nal_unit_type : 7 ( SPS )
	addbits (0x42,8); // profile_idc = baseline ( 0x42 )
	addbits (0x0,1); // constraint_set0_flag
	addbits (0x0,1); // constraint_set1_flag
	addbits (0x0,1); // constraint_set2_flag
	addbits (0x0,1); // constraint_set3_flag
```

```
        addbits (0x0,1); // constraint_set4_flag
        addbits (0x0,1); // constraint_set5_flag
        addbits (0x0,2); // reserved_zero_2bits /* equal to 0 */
        addbits (0x0a,8); // level_idc: 3.1 (0x0a)
        addexpgolombunsigned(0); // seq_parameter_set_id
        addexpgolombunsigned(0); // log2_max_frame_num_minus4
        addexpgolombunsigned(0); // pic_order_cnt_type
        addexpgolombunsigned(0); // log2_max_pic_order_cnt_lsb_minus4
        addexpgolombunsigned(0); // max_num_refs_frames
        addbits(0x0,1); // gaps_in_frame_num_value_allowed_flag

        int nWinMbs = nImW / nMbW;
        addexpgolombunsigned(nWinMbs-1); // pic_width_in_mbs_minus_1
        int nHinMbs = nImH / nMbH;
        addexpgolombunsigned(nHinMbs-1); // pic_height_in_map_units_minus_1

        addbits(0x1,1); // frame_mbs_only_flag
        addbits(0x0,1); // direct_8x8_interfernce
        addbits(0x0,1); // frame_cropping_flag
        addbits(0x1,1); // vui_parameter_present

        //VUI parameters (AR, timming)
        addbits(0x1,1); //aspect_ratio_info_present_flag
        addbits(0xFF,8); //aspect_ratio_idc = Extended_SAR

        //AR
        addbits(nSARw, 16); //sar_width
        addbits(nSARh, 16); //sar_height

        addbits(0x0,1); //overscan_info_present_flag
        addbits(0x0,1); //video_signal_type_present_flag
        addbits(0x0,1); //chroma_loc_info_present_flag
        addbits(0x1,1); //timing_info_present_flag

        unsigned int nnum_units_in_tick = TIME_SCALE_IN_HZ / (2*nFps);
        addbits(nnum_units_in_tick,32); //num_units_in_tick
        addbits(TIME_SCALE_IN_HZ,32); //time_scale
        addbits(0x1,1);  //fixed_frame_rate_flag

        addbits(0x0,1);  //nal_hrd_parameters_present_flag
        addbits(0x0,1);  //vcl_hrd_parameters_present_flag
        addbits(0x0,1);  //pic_struct_present_flag
        addbits(0x0,1);  //bitstream_restriction_flag
        //END VUI

        addbits(0x0,1); // frame_mbs_only_flag
        addbits(0x1,1); // rbsp stop bit

        dobytealign();
}

//Creates and saves the NAL PPS (one per file)
void CJOCh264encoder::create_pps ()
{
        add4bytesnoemulationprevention (0x000001); // NAL header
        addbits (0x0,1); // forbidden_bit
        addbits (0x3,2); // nal_ref_idc
        addbits (0x8,5); // nal_unit_type : 8 ( PPS )
        addexpgolombunsigned(0); // pic_parameter_set_id
        addexpgolombunsigned(0); // seq_parameter_set_id
        addbits (0x0,1); // entropy_coding_mode_flag
        addbits (0x0,1); // bottom_field_pic_order_in frame_present_flag
        addexpgolombunsigned(0); // nun_slices_groups_minus1
        addexpgolombunsigned(0); // num_ref_idx10_default_active_minus
```

```
        addexpgolombunsigned(0); // num_ref_idx11_default_active_minus
        addbits (0x0,1); // weighted_pred_flag
        addbits (0x0,2); // weighted_bipred_idc
        addexpgolombsigned(0); // pic_init_qp_minus26
        addexpgolombsigned(0); // pic_init_qs_minus26
        addexpgolombsigned(0); // chroma_qp_index_offset
        addbits (0x0,1); //deblocking_filter_present_flag
        addbits (0x0,1); // constrained_intra_pred_flag
        addbits (0x0,1); //redundant_pic_ent_present_flag
        addbits(0x1,1); // rbsp stop bit

        dobytealign();
}

//Creates and saves the NAL SLICE (one per frame)
void CJOCh264encoder::create_slice_header(unsigned long lFrameNum)
{
        add4bytesnoemulationprevention (0x000001); // NAL header
        addbits (0x0,1); // forbidden_bit
        addbits (0x3,2); // nal_ref_idc
        addbits (0x5,5); // nal_unit_type : 5 ( Coded slice of an IDR picture  )
        addexpgolombunsigned(0); // first_mb_in_slice
        addexpgolombunsigned(7); // slice_type
        addexpgolombunsigned(0); // pic_param_set_id

        unsigned char cFrameNum = lFrameNum % 16; //(2⁴)
        addbits (cFrameNum,4); // frame_num ( numbits = v = log2_max_frame_num_minus4 + 4)

        unsigned long lidr_pic_id = lFrameNum % 512;
        //idr_pic_flag = 1
        addexpgolombunsigned(lidr_pic_id); // idr_pic_id
        addbits(0x0,4); // pic_order_cnt_lsb (numbits = v =
log2_max_fpic_order_cnt_lsb_minus4 + 4)
        addbits(0x0,1); //no_output_of_prior_pics_flag
        addbits(0x0,1); //long_term_reference_flag
        addexpgolombsigned(0); //slice_qp_delta

        //Probably NOT byte aligned!!!
}

//Creates and saves the SLICE footer (one per SLICE)
void CJOCh264encoder::create_slice_footer()
{
        addbits(0x1,1); // rbsp stop bit
}

//Creates and saves the macroblock header(one per macroblock)
void CJOCh264encoder::create_macroblock_header ()
{
        addexpgolombunsigned(25); // mb_type (I_PCM)
}

//Creates & saves a macroblock (coded INTRA 16x16)
void CJOCh264encoder::create_macroblock(unsigned int nYpos, unsigned int nXpos)
{
        unsigned int x,y;

        create_macroblock_header();

        dobytealign();

        //Y
        unsigned int nYsize = m_frame.nYwidth * m_frame.nYheight;
        for(y = nYpos * m_frame.nYmbheight; y < (nYpos+1) * m_frame.nYmbheight; y++)
```

```
	{
		for (x = nXpos * m_frame.nYmbwidth; x < (nXpos+1) * m_frame.nYmbwidth; x++)
		{
			addbyte (m_frame.yuv420pframe.pYCbCr[(y * m_frame.nYwidth +  x)]);
		}
	}

	//Cb
	unsigned int nCsize = m_frame.nCwidth * m_frame.nCheight;
	for(y = nYpos * m_frame.nCmbheight; y < (nYpos+1) * m_frame.nCmbheight; y++)
	{
		for (x = nXpos * m_frame.nCmbwidth; x < (nXpos+1) * m_frame.nCmbwidth; x++)
		{
			addbyte(m_frame.yuv420pframe.pYCbCr[nYsize + (y * m_frame.nCwidth +
x)]);
		}
	}

	//Cr
	for(y = nYpos * m_frame.nCmbheight; y < (nYpos+1) * m_frame.nCmbheight; y++)
	{
		for (x = nXpos * m_frame.nCmbwidth; x < (nXpos+1) * m_frame.nCmbwidth; x++)
		{
			addbyte(m_frame.yuv420pframe.pYCbCr[nYsize + nCsize + (y *
m_frame.nCwidth +  x)]);
		}
	}
}


//public functions

//Initilizes the h264 coder (mini-coder)
void CJOCh264encoder::IniCoder (int nImW, int nImH, int nImFps,
CJOCh264encoder::enSampleFormat SampleFormat, int nSARw, int nSARh)
{
	m_lNumFramesAdded = 0;

	if (SampleFormat != SAMPLE_FORMAT_YUV420p)
		throw "Error: SAMPLE FORMAT not allowed. Only yuv420p is allowed in this
version";

	free_video_src_frame ();

	//Ini vars
	m_frame.sampleformat = SampleFormat;
	m_frame.nYwidth = nImW;
	m_frame.nYheight = nImH;
	if (SampleFormat == SAMPLE_FORMAT_YUV420p)
	{
		//Set macroblock Y size
		m_frame.nYmbwidth = MACROBLOCK_Y_WIDTH;
		m_frame.nYmbheight = MACROBLOCK_Y_HEIGHT;

		//Set macroblock C size (in YUV420 is 1/2 of Y)
		m_frame.nCmbwidth = MACROBLOCK_Y_WIDTH/2;
		m_frame.nCmbheight = MACROBLOCK_Y_HEIGHT/2;

		//Set C size
		m_frame.nCwidth = m_frame.nYwidth / 2;
		m_frame.nCheight = m_frame.nYheight / 2;

		//In this implementation only picture sizes multiples of macroblock size
(16x16) are allowed
```

```
		if (((nImW % MACROBLOCK_Y_WIDTH) != 0)||((nImH % MACROBLOCK_Y_HEIGHT) != 0))
			throw "Error: size not allowed. Only multiples of macroblock are
allowed (macroblock size is: 16x16)";
	}
	m_nFps = nImFps;

	//Alloc mem for 1 frame
	alloc_video_src_frame ();

	//Create h264 SPS & PPS
	create_sps (m_frame.nYwidth , m_frame.nYheight, m_frame.nYmbwidth,
m_frame.nYmbheight, nImFps , nSARw, nSARh);
	create_pps ();
}

//Returns the frame pointer to load the video frame
void* CJOCh264encoder::GetFramePtr()
{
	if (m_frame.yuv420pframe.pYCbCr == NULL)
		throw "Error: video frame is null (not initialized)";

	return (void*) m_frame.yuv420pframe.pYCbCr;
}

//Returns the the allocated size for video frame
unsigned int CJOCh264encoder::GetFrameSize()
{
	return m_frame.nyuv420pframesize;
}

//Codifies & save the video frame (it only uses 16x16 intra PCM -> NO COMPRESSION!)
void CJOCh264encoder::CodeAndSaveFrame()
{
	//The slice header is not byte aligned, so the first macroblock header is not byte
aligned
	create_slice_header (m_lNumFramesAdded);

	//Loop over macroblock size
	unsigned int y,x;
	for (y = 0; y < m_frame.nYheight / m_frame.nYmbheight; y++)
	{
		for (x = 0; x < m_frame.nYwidth / m_frame.nYmbwidth; x++)
		{
			create_macroblock(y, x);
		}
	}

	create_slice_footer();
	dobytealign();

	m_lNumFramesAdded++;
}

//Returns the number of codified frames
unsigned long CJOCh264encoder::GetSavedFrames()
{
	return m_lNumFramesAdded;
}

//Closes the h264 coder saving the last bits in the buffer
void CJOCh264encoder::CloseCoder ()
{
	close();
}
```

CJOCh264bitstream.cpp

```
#include "CJOCh264bitstream.h"

CJOCh264bitstream::CJOCh264bitstream(FILE *pOutBinaryFile)
{
	clearbuffer();

	m_pOutFile = pOutBinaryFile;
}

CJOCh264bitstream::~CJOCh264bitstream()
{
	close();
}

void CJOCh264bitstream::clearbuffer()
{
	memset (&m_buffer,0,sizeof (unsigned char)* BUFFER_SIZE_BITS);
	m_nLastbitinbuffer = 0;
	m_nStartingbyte = 0;
}

int CJOCh264bitstream::getbitnum (unsigned long lval, int nNumbit)
{
	int lrc = 0;

	unsigned long lmask = (unsigned long) pow(2.0,nNumbit);
	if ((lval & lmask) > 0)
		lrc = 1;

	return lrc;
}

void CJOCh264bitstream::addbittostream (int nVal)
{
	if (m_nLastbitinbuffer >= BUFFER_SIZE_BITS)
	{
		savebufferbyte();
	}

	//Use circular buffer of BUFFER_SIZE_BYTES
	int nBytePos = (m_nStartingbyte + (m_nLastbitinbuffer / 8)) % BUFFER_SIZE_BYTES;
	//The first bit to add is on the left
	int nBitPosInByte = 7 - m_nLastbitinbuffer % 8;

	//Get the byte value from buffer
	int nValTmp = m_buffer[nBytePos];

	//Change the bit
	if (nVal > 0)
		nValTmp = (nValTmp | (int) pow(2.0,nBitPosInByte));
	else
		nValTmp = (nValTmp & ~((int) pow(2.0,nBitPosInByte)));

	//Save the new byte value to the buffer
	m_buffer[nBytePos] = (unsigned char) nValTmp;

	m_nLastbitinbuffer++;
}

void CJOCh264bitstream::addbytetostream (int nVal)
```

```
{
	if (m_nLastbitinbuffer >= BUFFER_SIZE_BITS)
	{
		savebufferbyte();
	}

	//Used circular buffer of BUFFER_SIZE_BYTES
	int nBytePos = (m_nStartingbyte + (m_nLastbitinbuffer / 8)) % BUFFER_SIZE_BYTES;
	//The first bit to add is on the left
	int nBitPosInByte = 7 - m_nLastbitinbuffer % 8;

	//Check if it is byte aligned
	if (nBitPosInByte != 7)
		throw "Error: inserting not aligment byte";

	//Add all byte to buffer
	m_buffer[nBytePos] = (unsigned char) nVal;

	m_nLastbitinbuffer = m_nLastbitinbuffer + 8;
}

void CJOCh264bitstream::dobytealign()
{
	//Check if the last bit in buffer is multiple of 8
	int nr = m_nLastbitinbuffer % 8;
	if ((nr % 8) != 0)
		m_nLastbitinbuffer = m_nLastbitinbuffer + (8 - nr);
}

void CJOCh264bitstream::savebufferbyte(bool bemulationprevention)
{
	bool bemulationpreventionexecuted = false;

	if (m_pOutFile == NULL)
		throw "Error: out file is NULL";

	//Check if the last bit in buffer is multiple of 8
	if ((m_nLastbitinbuffer % 8)  != 0)
		throw "Error: Save to file must be byte aligned";

	if ((m_nLastbitinbuffer / 8) <= 0)
		throw "Error: NO bytes to save";

	if (bemulationprevention == true)
	{
		//Emulation prevention will be used:
		/*As per h.264 spec,
		rbsp_data shouldn't contain
				- 0x 00 00 00
				- 0x 00 00 01
				- 0x 00 00 02
				- 0x 00 00 03

		rbsp_data shall be in the following way
				- 0x 00 00 03 00
				- 0x 00 00 03 01
				- 0x 00 00 03 02
				- 0x 00 00 03 03
		*/

		//Check if emulation prevention is needed (emulation prevention is byte align
defined)
		if ((m_buffer[((m_nStartingbyte + 0) % BUFFER_SIZE_BYTES)] ==
0x00)&&(m_buffer[((m_nStartingbyte + 1) % BUFFER_SIZE_BYTES)] ==
```

```
0x00)&&((m_buffer[((m_nStartingbyte + 2) % BUFFER_SIZE_BYTES)] =
0x00)||(m_buffer[((m_nStartingbyte + 2) % BUFFER_SIZE_BYTES)] =
0x01)||(m_buffer[((m_nStartingbyte + 2) % BUFFER_SIZE_BYTES)] =
0x02)||(m_buffer[((m_nStartingbyte + 2) % BUFFER_SIZE_BYTES)] = 0x03)))
		{
			int nbuffersaved = 0;
			unsigned char cEmulationPreventionByte =
H264_EMULATION_PREVENTION_BYTE;

			//Save 1st byte
			fwrite(&m_buffer[((m_nStartingbyte + nbuffersaved) %
BUFFER_SIZE_BYTES)], 1, 1, m_pOutFile);
			nbuffersaved ++;

			//Save 2st byte
			fwrite(&m_buffer[((m_nStartingbyte + nbuffersaved) %
BUFFER_SIZE_BYTES)], 1, 1, m_pOutFile);
			nbuffersaved ++;

			//Save emulation prevention byte
			fwrite(&cEmulationPreventionByte, 1, 1, m_pOutFile);

			//Save the rest of bytes (usually 1)
			while (nbuffersaved < BUFFER_SIZE_BYTES)
			{
				fwrite(&m_buffer[((m_nStartingbyte + nbuffersaved) %
BUFFER_SIZE_BYTES)], 1, 1, m_pOutFile);
				nbuffersaved++;
			}

			//All bytes in buffer are saved, so clear the buffer
			clearbuffer();

			bemulationpreventionexecuted = true;
		}
	}

	if (bemulationpreventionexecuted == false)
	{
		//No emulation prevention was used

		//Save the oldest byte in buffer
		fwrite(&m_buffer[m_nStartingbyte], 1, 1, m_pOutFile);

		//Move the index
		m_buffer[m_nStartingbyte] = 0;
		m_nStartingbyte++;
		m_nStartingbyte = m_nStartingbyte % BUFFER_SIZE_BYTES;
		m_nLastbitinbuffer = m_nLastbitinbuffer - 8;
	}
}

//Public functions

void CJOCh264bitstream::addbits (unsigned long lval, int nNumbits)
{
	if ((nNumbits <= 0 )||(nNumbits > 64))
		throw "Error: numbits must be between 1 ... 64";

	int nBit = 0;
	int n = nNumbits-1;
	while (n >= 0)
	{
		nBit = getbitnum (lval,n);
```

```
			n--;

			addbittostream (nBit);
		}
}

void CJOCh264bitstream::addbyte (unsigned char cByte)
{
		//Byte alignment optimization
		if ((m_nLastbitinbuffer % 8)  == 0)
		{
			addbytetostream (cByte);
		}
		else
		{
			addbits (cByte, 8);
		}
}

void CJOCh264bitstream::addexpgolombunsigned (unsigned long lval)
{
		//it implements unsigned exp golomb coding

		unsigned long lvalint = lval + 1;
		int nnumbits = log2 ((double)lvalint) + 1;

		for (int n = 0; n < (nnumbits-1); n++)
			addbits(0,1);

		addbits(lvalint,nnumbits);
}

void CJOCh264bitstream::addexpgolombsigned (long lval)
{
		//it implements a signed exp golomb coding

		unsigned long lvalint = abs(lval) * 2 - 1;
		if (lval <= 0)
			lvalint =  2 * abs(lval);

		addexpgolombunsigned(lvalint);
}

void CJOCh264bitstream::add4bytesnoemulationprevention (unsigned int nVal, bool bDoAlign)
{
		//Used to add NAL header stream
		//Remember: NAL header is byte oriented

		if (bDoAlign == true)
			dobytealign();

		if ((m_nLastbitinbuffer % 8) != 0)
			throw "Error: Save to file must be byte aligned";

		while (m_nLastbitinbuffer != 0)
			savebufferbyte();

		unsigned char cbyte = (nVal & 0xFF000000)>>24;
		fwrite(&cbyte, 1, 1, m_pOutFile);

		cbyte = (nVal & 0x00FF0000)>>16;
		fwrite(&cbyte, 1, 1, m_pOutFile);

		cbyte = (nVal & 0x0000FF00)>>8;
```

```
	fwrite(&cbyte, 1, 1, m_pOutFile);

	cbyte = (nVal & 0x000000FF);
	fwrite(&cbyte, 1, 1, m_pOutFile);
}

void CJOCh264bitstream::close()
{
	//Flush the data in stream buffer

	dobytealign();

	while (m_nLastbitinbuffer != 0)
		savebufferbyte();
}
```

CJOCh264encoder.h

```
#ifndef CJOCH264ENCODER_H_
#define CJOCH264ENCODER_H_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "CJOCh264bitstream.h"

/* h264 encoder class
 It is used to create the h264 compliant stream
 */
class CJOCh264encoder : CJOCh264bitstream
{
public:

	/**
	 * Allowed sample formats
	 */
	enum enSampleFormat
	{
		SAMPLE_FORMAT_YUV420p//!< SAMPLE_FORMAT_YUV420p
	};

private:
	/*!Set the used Y macroblock size for I PCM in YUV420p */
	#define MACROBLOCK_Y_WIDTH 16
	#define MACROBLOCK_Y_HEIGHT16

	/*!Set time base in Hz */
	#define TIME_SCALE_IN_HZ   27000000

	/*!Pointer to pixels */
	typedef struct
	{
		unsigned char *pYCbCr;
	}YUV420p_frame_t;

	/*! Frame  */
	typedef struct
	{
		enSampleFormat sampleformat;	/*!< Sample format */
		unsigned int nYwidth;			/*!< Y (luminance) block width in
pixels */
```

```
		unsigned int nYheight;			/*!< Y (luminance) block height in
pixels */
		unsigned int nCwidth;			/*!< C (Crominance) block width in
pixels */
		unsigned int nCheight;			/*!< C (Crominance) block height in
pixels */

		unsigned int nYmbwidth;			/*!< Y (luminance) macroblock width in
pixels */
		unsigned int nYmbheight;		/*!< Y (luminance) macroblock height in
pixels */
		unsigned int nCmbwidth;			/*!< Y (Crominance) macroblock width
in pixels */
		unsigned int nCmbheight;		/*!< Y (Crominance) macroblock height in
pixels */

		YUV420p_frame_t yuv420pframe;	/*!< Pointer to current frame data */
		unsigned int nyuv420pframesize;	/*!< Size in bytes of yuv420pframe */
	}frame_t;

	/*! The frame var*/
	frame_t m_frame;

	/*! The frames per second var*/
	int m_nFps;

	/*! Number of frames sent to the output */
	unsigned long m_lNumFramesAdded;

	//! Frees the frame yuv420pframe allocated memory
	void free_video_src_frame ();

	//! Allocs the frame yuv420pframe memory according to the frame properties
	void alloc_video_src_frame ();

	//! Creates SPS NAL and add it to the output
	/*!
		\param nImW Frame width in pixels
		\param nImH Frame height in pixels
		\param nMbW macroblock width in pixels
		\param nMbH macroblock height in pixels
		\param nFps frames x second (tipical values are: 25, 30, 50, etc)
		\param nSARw Indicates the horizontal size of the sample aspect ratio (tipical
values are:1, 4, 16, etc)
		\param nSARh Indicates the vertical size of the sample aspect ratio (tipical
values are:1, 3, 9, etc)
	 */
	void create_sps (int nImW, int nImH, int nMbW, int nMbH, int nFps, int nSARw, int
nSARh);

	//! Creates PPS NAL and add it to the output
	void create_pps ();

	//! Creates Slice NAL and add it to the output
	/*!
		\param lFrameNum number of frame
	 */
	void create_slice_header(unsigned long lFrameNum);

	//! Creates macroblock header and add it to the output
	void create_macroblock_header ();

	//! Creates the slice footer and add it to the output
	void create_slice_footer();
```

```
        //! Creates SPS NAL and add it to the output
        /*!
                \param nYpos First vertical macroblock pixel inside the frame
                \param nYpos nXpos horizontal macroblock pixel inside the frame
         */
        void create_macroblock(unsigned int nYpos, unsigned int nXpos);

public:
        //! Constructor
        /*!
                 \param pOutFile The output file pointer
         */
        CJOCh264encoder(FILE *pOutFile);

        //! Destructor
        virtual ~CJOCh264encoder();

        //! Initializes the coder
        /*!
                \param nImW Frame width in pixels
                \param nImH Frame height in pixels
                \param nFps Desired frames per second of the output file (typical values are:
25, 30, 50, etc)
                \param SampleFormat Sample format if the input file. In this implementation
only SAMPLE_FORMAT_YUV420p is allowed
                \param nSARw Indicates the horizontal size of the sample aspect ratio (typical
values are:1, 4, 16, etc)
                \param nSARh Indicates the vertical size of the sample aspect ratio (typical
values are:1, 3, 9, etc)
        */
        void IniCoder (int nImW, int nImH, int nImFps, CJOCh264encoder::enSampleFormat
SampleFormat, int nSARw = 1, int nSARh = 1);

        //! Returns the frame pointer
        /*!
                \return Frame pointer ready to fill with frame pixels data (the format to fill
the data is indicated by SampleFormat parameter when the coder is initialized
        */
        void* GetFramePtr();

        //! Returns the allocated frame memory in bytes
        /*!
                \return The allocated memory to store the frame data
        */
        unsigned int GetFrameSize();

        //! It codes the frame that is in frame memory a it saves the coded data to disc
        void CodeAndSaveFrame();

        //! Returns number of coded frames
        /*!
                \return The number of coded frames
        */
        unsigned long GetSavedFrames();

        //! Flush all data and save the trailing bits
        void CloseCoder ();
};

#endif /* CJOCH264ENCODER_H_ */
```

CJOCh264bitstream.h

```
#ifndef CJOCH264BITSTREAM_H_
#define CJOCH264BITSTREAM_H_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

/* h264 bitstream class
 It is used to create the h264 bit oriented stream, it contains different functions that
helps you to create the h264 compliant stream (bit oriented, exp golomb coder)
 */
class CJOCh264bitstream
{
#define BUFFER_SIZE_BITS 24             /*! Buffer size in bits used for emulation
prevention */
#define BUFFER_SIZE_BYTES (24/8)  /*! Buffer size in bytes used for emulation prevention */

#define H264_EMULATION_PREVENTION_BYTE 0x03           /*! Emulation prevention byte */

private:

       /*! Buffer  */
       unsigned char m_buffer[BUFFER_SIZE_BITS];

       /*! Bit buffer index  */
       unsigned int m_nLastbitinbuffer;

       /*! Starting byte indicator  */
       unsigned int m_nStartingbyte;

       /*! Pointer to output file */
       FILE *m_pOutFile;

       //! Clears the buffer
       void clearbuffer();

       //! Returns the nNumbit value (1 or 0) of lval
       /*!
              \param lval number to extract the nNumbit value
              \param nNumbit Bit position that we want to know if its 1 or 0 (from 0 to 63)
              \return bit value (1 or 0)
        */
       static int getbitnum (unsigned long lval, int nNumbit);

       //! Adds 1 bit to the end of h264 bitstream
       /*!
              \param nVal bit to add at the end of h264 bitstream
        */
       void addbittostream (int nVal);

       //! Adds 8 bit to the end of h264 bitstream (it is optimized for byte aligned
situations)
       /*!
              \param nVal byte to add at the end of h264 bitstream (from 0 to 255)
        */
       void addbytetostream (int nVal);

       //! Save all buffer to file
       /*!
```

```
		\param bemulationprevention Indicates if it will insert the emulation
prevention byte or not (when it is needed)
	 */
	void savebufferbyte(bool bemulationprevention = true);

public:
	//! Constructor
	/*!
		\param pOutBinaryFile The output file pointer
	 */
	CJOCh264bitstream(FILE *pOutBinaryFile);

	//! Destructor
	virtual ~CJOCh264bitstream();

	//! Add 4 bytes to h264 bistream without taking into acount the emulation prevention.
Used to add the NAL header to the h264 bistream
	/*!
		\param nVal The 32b value to add
		\param bDoAlign Indicates if the function will insert 0 in order to create a
byte aligned stream before adding nVal 4 bytes to stream. If you try to call this function
and the stream is not byte aligned an exception will be thrown
	 */
	void add4bytesnoemulationprevention (unsigned int nVal, bool bDoAlign = false);

	//! Adds nNumbits of lval to the end of h264 bitstream
	/*!
		\param nVal value to add at the end of the h264 stream (only the LAST
nNumbits will be added)
		\param nNumbits number of bits of lval that will be added to h264 stream
(counting from left)
	 */
	void addbits (unsigned long lval, int nNumbits);

	//! Adds lval to the end of h264 bitstream using exp golomb coding for unsigned
values
	/*!
		\param nVal value to add at the end of the h264 stream
	 */
	void addexpgolombunsigned (unsigned long lval);

	//! Adds lval to the end of h264 bitstream using exp golomb coding for signed values
	/*!
		\param nVal value to add at the end of the h264 stream
	 */
	void addexpgolombsigned (long lval);

	//! Adds 0 to the end of h264 bistream in order to leave a byte aligned stream (It
will insert seven 0 maximum)
	void dobytealign();

	//! Adds cByte (8 bits) to the end of h264 bitstream. This function it is optimized
in byte aligned streams.
	/*!
		\param cByte value to add at the end of the h264 stream (from 0 to 255)
	 */
	void addbyte (unsigned char cByte);

	//! Close the h264 stream saving to disk the last remaing bits in buffer
	void close();
};

#endif /* CJOCH264BITSTREAM_H_ */
```