

(19) C2 (11) 125468 (13) UA

(98) а/с 78, м. Київ, 03150

(85) null

(74) Пахаренко Антоніна Павлівна, (UA)

(45) [2022-03-23]

(43) [2017-07-10]

(24) 2022-03-24

(22) 2013-04-15

(12) Патент України (на 20 р.)

(21) а201702384

(46) 2022-03-23

(86)

(30) 61/624,098 2012-04-13 US 61/666,185 2012-06-29 US

(54) КОДУВАННЯ КАРТИНКИ З МАЛОЮ ЗАТРИМКОЮ

(56) US 7782947 B2, 24.08.2010 1 EP 2351375 A1, 03.08.2011 1 US 5786858 A, 28.07.1998 1 Chi Ching Chi, Mauricio Mesa, Jan Lucas, Ben Juurlink, Thomas Schierl. Parallel HEVC Decoding on Multi- and Many-core Architectures. Journal of Signal Processing Systems . 71 (3). June 2012. pp. 1-14. 1 Thomas Wiegand, Gary J. Sullivan, Senior Member, IEEE, Gisle Bjontegaard, and Ajay Luthra. Overview of the H.264/AVC Video Coding Standard. IEEE TRANS. CIRCUITS SYST. VIDEO TECHNOL., vol. 13, no. 7, JULY 2003, IEEE 1

(71) US ДЖ.І. ВІДІЕУ КЕМПРЕШН, ЛЛСІ US GE VIDEO COMPRESSION, LLC

(72) DE Ширль Томас DE Schierl, Thomas DE Георге Валері DE George, Valeri DE Хенкель Анастасія DE Henkel, Anastasia DE Марпе Детлеф DE Marpe, Detlev DE Грюнеберг Карстен DE Grüneberg, Karsten DE Шкупін Роберт DE Skupin, Robert

(73) US ДЖ.І. ВІДІЕУ КЕМПРЕШН, ЛЛСІ US GE VIDEO COMPRESSION, LLC

Концепції паралельного кодування, така як паралельна хвильова обробка даних, реалізується з меншою наскрізною затримкою шляхом відмови від традиційної концепції вирізки, згідно з якою вирізки або повністю кодуються/декоднуються незалежно від ділянок картини зовні відповідної вирізки, або принаймні незалежно від ділянок зовні відповідної вирізки настільки, наскільки це стосується ентропійного кодування, зокрема на користь вирізок різних режимів, зокрема названих залежними вирізками, які передбачають внутрішні залежності крізь межі вирізки, та інші, які називаються, наприклад, нормальними вирізками. Об'єднана, чи ні, з аспектом концепція WPP робиться ефективнішою шляхом використання початкових частин синтаксису вирізок для вказання координат точок входу WPP.

1. Декодер для декодування інформації з потоку даних для відтворення картинки, що розбивається на вирізки, де декодер містить ентропійний декодер, сконфігурований для ентропійного декодування, з використанням процесора, де вирізки ґрунтуються на паралельній обробці хвильового фронту (WPP), в якій поточні вирізки вирізок ентропійно декодуються відповідно до одного із щонайменше двох режимів, вказаних як частина синтаксичного елемента в поточній вирізці, при цьому ентропійний декодер сконфігурований:

у відповідності з першим режимом з щонайменше двох режимів, які декодують дані, які належать до поточної вирізки з потоку даних з використанням адаптивного ентропійного декодування контексту, для одержання залишкового сигналу, який включає одержання контекстів в межах вирізки ініціалізації ймовірності символів, асоційованих з поточною вирізкою, залежно від збереженого стану ймовірності символу попередньо декодованої вирізки,

при цьому, в ініціалізації ймовірності символів, асоційованих з поточною вирізкою у відповідності з першим режимом, ентропійний декодер сконфігурований на:

перевірку, чи є перший блок кодування, асоційований з поточною вирізкою, першим блоком кодування в низці картини у відповідності з порядком растрового сканування, і

в такому випадку, ініціалізація ймовірності символів, асоційованих з поточною вирізкою, залежно від збережених ймовірностей символів, які отримують в адаптивному ентропійному декодуванні контексту попередньо декодованої вирізки до другого блока кодування в ряду, асоційованому з попередньо декодованою вирізкою у відповідності з порядком растрового сканування, і,

якщо ні, ініціалізація ймовірності символів, асоційованих з поточною вирізкою, залежно від ймовірності символів, отриманих в адаптивному ентропійному декодуванні контексту попередньо декодованої вирізки до кінця попередньо декодованої вирізки, у відповідності з другим режимом з щонайменше двох режимів, декодують дані, які належать до поточної вирізки, з потоку даних з використанням адаптивного ентропійного декодування контексту для одержання залишкового сигналу, який включає обмеження одержання контекстів за умови, що не перетинаються межі вирізки та ініціалізації ймовірностей символів незалежно від попередньо декодованої вирізки, і

збереження ймовірності символів, які отримують в адаптивному ентропійному декодуванні контексту попередньо декодованої вирізки до другого блока кодування в ряду, асоційованому з попередньо декодованою вирізкою у відповідності з порядком растрового сканування;

предиктор, сконфігурований для генерування з використанням процесора, де сигнал прогнозування базується на параметрах прогнозування, які стосуються поточної вирізки з потоку даних; і

відтворення модуля, сконфігурованого для відтворення з використанням процесора, де частина картинки, яка стосується поточної вирізки, базується на залишковому сигналі і сигналі прогнозування.

2. Декодер за п. 1, який **відрізняється** тим, що картинку розбивають на блоки кодування, які розташовані в рядках і стовпчиках і мають порядок растрового сканування, визначений серед них, і декодер, сконфігурований для зв'язування кожної вирізки з неперервною підмножиною блоків кодування в порядку растрового сканування, такими чином, що підмножини ідуть одна за одною згідно з порядком растрового сканування у відповідності з порядком вирізки.

3. Декодер за п. 1, який **відрізняється** тим, що ентропійний декодер, як сконфігуровано, дає відповідь на частину синтаксичного елемента в поточній вирізці вирізок для декодування поточної вирізки у відповідності з одним з принаймні трьох режимів, зокрема в одному з першого та третього режимів або другого режиму, при цьому декодер сконфігурований у відповідності з першим режимом для декодування поточної вирізки з використанням прогнозуючого декодування в межах вирізки у відповідності з другим режимом для декодування поточної вирізки з використанням обмеження прогнозуючого декодування за умови, що не перетинають межі вирізки, і

у відповідності з третім режимом для декодування поточної вирізки з потоку даних з використанням адаптивного ентропійного декодування контексту з обмеженням одержання контекстів за умови, що не перетинають межі вирізки, безперервного оновлення ймовірностей символів контекстів і ініціалізації ймовірностей символів незалежно від будь-якої попередньо декодованої вирізки, і прогнозуючого декодування в межі вирізки, при цьому один режим з першого і третього режимів вибирають залежно від синтаксичного елемента.

4. Декодер за п. 1, який **відрізняється** тим, що ентропійний декодер, як сконфігуровано, дає відповідь на загальний синтаксичний елемент в потоці даних, таким чином, щоб працювати в одному з принаймні двох загальних робочих режимів, з, згідно з першим загальним робочим режимом, виконанням відповіді на частину синтаксичного елемента для кожної вирізки і, згідно з другим загальним робочим режимом, для використання відмінного режиму з принаймні двох режимів, відмінних від першого режиму.

5. Декодер за п. 2, який **відрізняється** тим, що ентропійний декодер сконфігурований згідно з першим і другим режимами, що продовжують постійне оновлення ймовірностей символів

від початку до кінця поточної вирізки.

6. Декодер за п. 2, який **відрізняється** тим, що ентропійний декодер, як сконфігуровано, зберігає ймовірності символів, які отримано в адаптивному ентропійному декодуванні контексту попередньо декодованої вирізки до кінця попередньо декодованої вирізки.

7. Декодер за п. 3, який **відрізняється** тим, що ентропійний декодер, як сконфігуровано, в першому і другому режимі обмежує прогнозує декодування в мозаїках, на які підрозбивається картинка.

8. Декодер за п. 1, який **відрізняється** тим, що ентропійний декодер, як сконфігуровано, у відповідності з першим з принаймні двох режимів копіює для поточної вирізки частину синтаксису заголовка вирізки з попередньої вирізки, декодованої в другому режимі.

9. Декодер за п. 1, який **відрізняється** тим, що кожна вирізка містить початкову частину синтаксису, яка вказує положення в картинці, в якому починається ентропійне декодування відповідної вирізки, де ентропійний декодер сконфігурований для ідентифікації точок входу WPP вкладених потоків, асоційованих з вирізками з використанням початкових частин синтаксису вирізок.

10. Декодер за п. 9, який **відрізняється** тим, що ентропійний декодер, як сконфігуровано, паралельно декодує WPP вкладені потоки в шаховому порядку з послідовним початком декодування WPP вкладених потоків у відповідності з порядком вирізки.

11. Кодер для кодування картинки з одержанням потоку даних, при цьому картинка розбивається на вирізки, де кодер містить:

залишковий сигнал, сконфігурований, щоб генерувати з використанням процесора залишковий сигнал, який належить до поточної вирізки вирізок, картинок;

предиктор, сконфігурований, щоб генерувати з використанням процесора параметрів прогнозування, які базуються на сигналі прогнозування, при цьому параметри прогнозування кодуються в потік даних; і

ентропійний декодер, сконфігурований для визначення з використанням процесора, частинок синтаксичного елемента для поточної вирізки і ентропійного кодування в потоці даних, де вирізки базуються на паралельній обробці хвильового фронту (WPP), де залишковий сигнал, який належить до поточної вирізки, ентропійно кодується, відповідно до одного із щонайменше двох режимів, вказаних як частина синтаксичного елемента, при цьому ентропійний кодер сконфігурований:

якщо поточна вирізка повинна кодуватися у відповідності з першим режимом з принаймні двох режимів, кодування залишкового сигналу даних з використанням адаптивного ентропійного кодування контексту, що включає одержання контекстів в межах вирізки та

ініціалізацію ймовірності символів, асоційованих з поточною вирізкою, залежно від збереженого стану ймовірності символу попередньо кодованої вирізки, при цьому, в ініціалізації ймовірності символів, асоційованих з поточною вирізкою у відповідності з першим режимом, ентропійний кодер сконфігурований для перевірки, чи є перший блок кодування, асоційований з поточною вирізкою, першим блоком кодування в ряду картинки у відповідності з порядком растрового сканування, і, якщо так, ініціалізують ймовірності символів, асоційованих з поточною вирізкою залежно від збережених ймовірностей символів, які отримано в адаптивному ентропійному кодуванні контексту попередньо кодованої вирізки до другого блока кодування в ряду, асоційованому з попередньо кодованою вирізкою у відповідності з порядком растрового сканування, якщо ні, ініціалізують ймовірності символів, асоційованих з поточною вирізкою залежно від ймовірності символів, отриманих в адаптивному ентропійному кодуванні контексту попередньо кодованої вирізки до кінця попередньо кодованої вирізки, якщо поточна вирізка повинна кодуватися у відповідності з другим режимом з принаймні двох режимів, кодування залишкового сигналу з використанням адаптивного ентропійного кодування контексту з обмеженням одержання контекстів за умови, що не перетинають межі вирізки, і ініціалізації ймовірностей символів незалежно від будь-якої попередньо кодованої вирізки, і збереження ймовірності символів, які отримано в адаптивному ентропійному кодуванні контексту попередньо кодованої вирізки до другого блока кодування в ряду, асоційованому з попередньо кодованою вирізкою у відповідності з порядком растрового сканування.

12. Кодер за п. 11, який **відрізняється** тим, що картинку розбивають на блоки кодування, які розташовані в рядках і стовпчиках і мають порядок растрового сканування, визначений серед них, і кодер сконфігурований для зв'язування кожної вирізки з неперервною підмножиною блоків кодування в порядку растрового сканування таким чином, що підмножини ідуть одна за іншою згідно з порядком растрового сканування у відповідності з порядком вирізки.

13. Кодер за п. 11, який **відрізняється** тим, що ентропійний кодер сконфігурований для кодування частини синтаксичного елемента в поточній вирізці вирізок, таким чином, що поточна вирізка є сигнальною для кодування в ній у відповідності з одним з принаймні трьох режимів, зокрема в одному з першого і третього режиму або другого режиму, при цьому кодер сконфігурований, щоб у відповідності з першим режимом кодувати поточну вирізку з використанням прогнозуючого кодування в межах вирізки, у відповідності з другим режимом кодувати поточну вирізку з використанням прогнозуючого кодування з обмеженням прогнозуючого кодування за умови, що не перетинаються межі вирізки, у відповідності з третім режимом кодувати поточну вирізку з потоку даних з використанням

адаптивного ентропійного кодування контексту з обмеженням одержання контекстів за умови, що не перетинають межі вирізки, безперервним оновленням ймовірностей символів контекстів і ініціалізацією ймовірностей символів незалежно від будь-якої попередньо кодованої вирізки, і прогнозуючого кодування в межах вирізки, при цьому ентропійний кодер відрізняє один від одного перший і третій режими, з використанням синтаксичного елемента.

14. Кодер за п. 11, який **відрізняється** тим, що ентропійний кодер сконфігурований для визначення загального синтаксичного елемента і запису його в потік даних з роботою в одному з принаймні двох загальних робочих режимів залежно від загального синтаксичного елемента, зокрема згідно з першим загальним робочим режимом, з виконанням кодування частини синтаксичного елемента для кожної вирізки і згідно з другим загальним робочим режимом, неухильним використанням іншого режиму, вибраного з принаймні двох режимів, відмінних від першого режиму.

15. Кодер за п. 12, який **відрізняється** тим, що ентропійний кодер сконфігурований згідно з першим і другим режимом, що продовжують постійне оновлення ймовірностей символів від початку до кінця поточної вирізки.

16. Кодер за п. 12, який **відрізняється** тим, що ентропійний кодер сконфігурований для збереження ймовірностей символів, які отримано в адаптивному ентропійному кодуванні контексту попередньо кодованої вирізки до кінця попередньо кодованої вирізки.

17. Кодер за п. 13, який **відрізняється** тим, що ентропійний кодер сконфігурований для обмеження, в першому і другому режимі, прогнозуючого кодування в мозаїках, на які підрозбивається картинка.

18. Кодер за п. 11, який **відрізняється** тим, що кожна вирізка містить початкову синтаксичну частину, яка вказує положення в картинці, в якому починається ентропійне кодування відповідної вирізки, де ентропійний кодер сконфігурований для ідентифікації точок входу WPP вкладених потоків, асоційованих з вирізками з використанням початкових синтаксичних частин вирізок.

19. Спосіб декодування інформації з потоку даних для відтворення картинки, що розбивається на вирізки, де спосіб включає:

ентропійне декодування вирізок, яке базується на паралельній обробці хвильового фронту (WPP), в якій поточні вирізки вирізок ентропійно декодуються, відповідно до одного із щонайменше двох режимів вказаних як частина синтаксичного елемента в поточній вирізці, при цьому ентропійне декодування включає:

у відповідності з першим режимом з щонайменше двох режимів декодування даних, які

належать до поточної вирізки з потоку даних з використанням адаптивного ентропійного декодування контексту для одержання залишкового сигналу, що включає одержання контекстів в межах вирізки, ініціалізацію ймовірності символів, асоційованих з поточною вирізкою залежно від збереженого стану ймовірності символу попередньо декодованої вирізки,

при цьому, в ініціалізації ймовірності символів, асоційованих з поточною вирізкою, у відповідності з першим режимом, ентропійне декодування включає перевірку, чи є перший блок кодування, асоційований з поточною вирізкою, першим блоком кодування в ряду картинки у відповідності з порядком растрового сканування, і,

якщо так, ініціалізацію ймовірності символів, асоційованих з поточною вирізкою, залежно від збережених ймовірностей символів, отримуваних в адаптивному ентропійному декодуванні контексту попередньо декодованої вирізки до другого блока кодування в ряду, асоційованому з попередньо декодованою вирізкою у відповідності з порядком растрового сканування, і,

якщо ні, ініціалізацію ймовірності символів, асоційованих з поточною вирізкою, залежно від ймовірності символів, отриманих в адаптивному ентропійному декодуванні контексту попередньо декодованої вирізки до кінця попередньо декодованої вирізки, у відповідності з другим режимом з щонайменше двох режимів декодування даних, які належать до поточної вирізки, з потоку даних з використанням адаптивного ентропійного декодування контексту, для одержання залишкового сигналу, що включає обмеження одержання контекстів за умови, що не перетинаються межі вирізки, і ініціалізацію ймовірностей символів незалежно від попередньо декодованої вирізки, і

збереження ймовірності символів, які отримано в адаптивному ентропійному декодуванні контенту попередньо декодованої вирізки до другого блока кодування в ряду, асоційованому з попередньо декодованою вирізкою у відповідності з порядком растрового сканування;

генерування прогнозуючого сигналу, який базується на параметрах прогнозування, які належать до поточної вирізки з потоку даних, і

відтворення частини картинки, яка належить до поточної вирізки, що базується на залишковому сигналі і сигналі прогнозування.

20. Спосіб за п. 19, який **відрізняється** тим, що картинку розбивають на блоки кодування, які розташовані в рядках і стовпчиках і мають порядок растрового сканування, визначений серед них, і спосіб додатково включає зв'язування кожної вирізки з неперервною підмножиною блоків кодування в порядку растрового сканування таким чином, що підмножини ідуть одна за одною згідно з порядком растрового сканування у відповідності з

порядком вирізки.

21. Спосіб за п. 19, який **відрізняється** тим, що ентропійне декодування реагує на загальний синтаксичний елемент в потоці даних для роботи в одному з принаймні двох загальних робочих режимів, згідно з першим загальним робочим режимом, із здійсненням відповіді на частину синтаксичного елемента для кожної вирізки i , згідно з другим загальним робочим режимом, з використанням іншого режиму, з принаймні двох режимів, відмінних від першого режиму.

22. Спосіб за п. 20, який **відрізняється** тим, що ентропійне декодування включає у відповідності з першим і другим режимами постійне оновлення ймовірностей символів від початку до кінця поточної вирізки.

23. Спосіб за п. 20, який **відрізняється** тим, що ентропійне декодування включає збереження ймовірностей символів, які отримано в адаптивному ентропійному декодуванні контексту попередньо декодованої вирізки до кінця попередньо декодованої вирізки.

24. Спосіб за п. 19, який **відрізняється** тим, що ентропійне декодування включає у відповідності з першим з принаймні двох режимів, копіювання частини поточної вирізки синтаксису заголовка вирізки з попередньої вирізки, декодованої в другому режимі.

25. Спосіб за п. 19, який **відрізняється** тим, що кожна вирізка містить початкову синтаксичну частину, яка вказує положення в картинці, в якій починається ентропійне декодування відповідної вирізки, де ентропійне декодування включає ідентифікацію точок входу WPP вкладених потоків, асоційованих з вирізками з використанням початкових синтаксичних частин вирізок.

26. Машинозчитуваний носій для довготривалого зберігання даних, пов'язаний із відео, що включає:

потік даних, що зберігається в машинозчитуваному носії для довготривалого зберігання даних, де потік даних включає кодовану інформацію, що належить до картинки відео, де картинка розподіляється в одиницях вирізок, де кодована інформація кодується в потік даних для зберігання у машинозчитуваному носії для довготривалого зберігання даних, шляхом операції, що включають генерування залишкового сигналу, пов'язаного з поточною вирізкою вирізок картинок,

генерування прогнозуючих параметрів, які базуються на сигналі прогнозування, при цьому прогнозуючі параметри кодуються в потік даних, визначення частини синтаксичного елемента для поточної вирізки, і

ентропійне кодування в потік даних, де вирізки базуються на паралельній обробці хвильового фронту (WPP), де залишковий сигнал, який належить до поточної вирізки,

ентропійно кодується, відповідно до одного із щонайменше двох режимів вказаних як частина синтаксичного елемента, при цьому ентропійне кодування включає, якщо поточна вирізка повинна кодуватися у відповідності з першим режимом з принаймні двох режимів, кодування залишкового сигналу з використанням адаптивного ентропійного кодування контексту, що включає одержання контекстів в межах вирізки ініціалізацію ймовірності символів, асоційованих з поточною вирізкою, залежно від збереженого стану ймовірності символу попередньо кодованої вирізки, при цьому, для ініціалізації ймовірності символів, асоційованих з поточною вирізкою у відповідності з першим режимом, операції включають перевірку, чи є перший блок кодування, асоційований з поточною вирізкою, першим блоком кодування в ряду картинки у відповідності з порядком растрового сканування, і, якщо так, ініціалізацію ймовірності символів, асоційованих з поточною вирізкою, залежно від збережених ймовірностей символів, які отримано в адаптивному ентропійному кодуванні контексту попередньо кодованої вирізки до другого блока кодування в ряду, асоційованому з попередньо кодованою вирізкою у відповідності з порядком растрового сканування, і, якщо ні, ініціалізацію ймовірності символів, асоційованих з поточною вирізкою, залежно від ймовірності символів, які отримано в адаптивному ентропійному кодуванні контексту попередньо кодованої вирізки, до кінця попередньо кодованої вирізки, якщо поточна вирізка повинна кодуватися у відповідності з другим режимом з принаймні двох режимів кодування залишкового сигналу з використанням адаптивного ентропійного кодування контексту з обмеженням одержання контекстів за умови, що не перетинаються межі вирізки, і ініціалізацію ймовірностей символів незалежно від будь-якої попередньо кодованої вирізки, і збереження ймовірності символів, які отримано в адаптивному ентропійному кодуванні контексту попередньо кодованої вирізки, до другого блока кодування в ряду, асоційованому з попередньо кодованою вирізкою у відповідності з порядком растрового сканування.

27. Машинозчитуваний носій для довготривалого зберігання даних за п. 26, який **відрізняється** тим, що операції додатково включають ідентифікацію точок входу WPP вкладених потоків, асоційованих з вирізками з використанням початкових синтаксичних частин вирізок, при цьому кожна вирізка містить початкову синтаксичну частину, яка вказує положення в картинці, в якому починається ентропійне декодування відповідної вирізки.

Представлений винахід стосується кодування картинок з малою затримкою.

В поточній структурі стандарту HEVC вирізки, ентропійні вирізки (колишні вирізки малої ваги), мозаїки і WPP (паралельна хвильова обробка даних) розглядаються як засоби для розпаралелювання процесу.

Для розпаралелювання роботи відеокoderів і відеодекoderів, розбиття картинки має декілька переваг порівняно з іншими наближеннями. В попередніх відеокodeках, таких як H.264/AVC [1], розбиття картинки були можливими тільки зі стандартними вирізками та високою вартістю з точки зору ефективності кодування. Для здатного до масштабування паралельного декодування за допомогою H.264/AVC, необхідно поєднувати розпаралелювання роботи на рівні макроблоків для відтворення картинки і розпаралелювання роботи на рівні кадру для ентропійного декодування. Однак, це наближення забезпечує обмежене зменшення затримок в кодуванні картинки і використання великих об'ємів пам'яті. Для подолання цих обмежень, в кодек HEVC були включені нові стратегії розбиття картинки. Поточна еталонна версія програмного забезпечення (HM-6) містить 4 різні наближення: стандартні або нормальні вирізки, ентропійні вирізки, вкладені потоки даних паралельної хвильової обробки даних (WPP) і мозаїки. Типово, ці розбиття картинки містять множину найбільших кодувальних чарунок (LCUs) або в синонімічному формулюванні кодувальних деревоподібних чарунок (CTU), як визначено в стандарті HEVC, або навіть їх підмножини.

Фігура 1 зображає, як картинка 898 ілюстративно розбита на стандартні вирізки 900, кожна з яких утворює ряд 902 LCUs або макроблоків в картинці. Стандартні або нормальні вирізки (як визначено в стандарті H.264 [1]) мають найбільше погіршення якості кодування, оскільки вони порушують залежності ентропійного декодування і прогнозування.

Ентропійні вирізки, такі як вирізки, порушують залежності ентропійного декодування, але дозволяють прогнозування (і фільтрування) крізь межі вирізки.

В паралельній хвильовій обробці даних (WPP) розбиття картинки перемежуються в рядках і дозволяються як ентропійне декодування, так і прогнозування для використання даних з блоків в інших розбиттях. У цей спосіб втрати кодування мінімізуються з одночасною можливістю використання розпаралелювання хвильової обробки даних. Однак, перемежовування порушує причинність бітового потоку, оскільки попереднє розбиття потребує наступного розбиття для декодування.

Фігура 2 ілюстративно зображає картинку 898, поділену на два рядки 904, 904b горизонтально подільних мозаїк 906. Мозаїки визначають горизонтальну 908 і вертикальну межу 910, які розбивають картинку 898 на стовпчики 912a, b, c мозаїк і рядки 904a, b мозаїк. Подібно до стандартних вирізок 900, мозаїки 906 порушують залежності ентропійного декодування і прогнозування, але не вимагають заголовка для кожної мозаїки.

Для кожної з цих технологій кодером може вільно вибиратися ряд розбиттів. Головним чином, більше розбиттів приводить до більших втрат стиснення. Однак в WPP поширення втрат не є таким великим і, тому, ряд розбиттів картинки навіть може фіксуватися по одному на рядок. Це також приводить до кількох переваг. По-перше, гарантується причинність WPP бітового потоку. По-друге, втілення декодера можуть припускати, що доступне певне розпаралелювання роботи, що також підвищується роздільною здатністю. І, нарешті, жодна із залежностей вибору контексту і прогнозування не повинна порушуватися при декодуванні у хвильовому порядку, призводячи до відносно малих втрат кодування.

Однак, до цього часу усе паралельне кодування в концепціях перетворення не забезпечує досягання високої ефективності стиснення в поєднанні із збереженням малої затримки. Це також вірно для концепції WPP. Вирізки є найменшими одиницями транспортування в кодуванні каналу інформації і декілька WPP вкладених потоків все ще повинні передаватися послідовно.

Відповідно, задачею представленого винаходу є надання концепції кодування картинки, яка передбачає паралельне декодування згідно з, наприклад, паралельною хвильовою обробкою даних з підвищеною ефективністю, як, наприклад, навіть з подальшим зменшенням наскрізної затримки або підвищенням ефективності кодування шляхом зменшення використовуюваного заголовку кодування.

Ця задача вирішується об'єктом незалежних пунктів формули винаходу.

Основне відкриття представленого винаходу полягає в тому, що концепції паралельної обробки даних, такі як паралельна хвильова обробка даних, можуть реалізовуватися з меншою наскрізною затримкою, якщо відмовитися від звичайної концепції вирізки, згідно з якою вирізки або кодуються/декодуються повністю незалежно від ділянок картини зовні відповідної вирізки, або принаймні незалежно від ділянок зовні відповідної вирізки настільки, наскільки це стосується ентропійного кодування, на користь вирізок різних режимів, а саме: так званих залежних вирізок, які передбачають внутрішні залежності крізь границі вирізки, та інших, які, наприклад, називаються нормальними вирізками.

Подальше основне відкриття представленого винаходу, яке може поєднуватися з першим відкриттям або використовуватися індивідуально, полягає в тому, що концепція WPP може робитися ефективнішою, якщо для визначення координат WPP точок входу використовуються початкові частини синтаксису вирізок.

Переважні варіанти виконання представленого винаходу описуються нижче стосовно фігур, при цьому переважні варіанти виконання є об'єктами залежних пунктів формули винаходу. На фігурах:

Фіг. 1 зображає картинку, ілюстративно розбиту на стандартні вирізки, які формують рядки LCUs або макроблоків в картинці;

Фіг. 2 зображає картинку, ілюстративно поділену на два рядки горизонтально розбитих мозаїк;

Фіг. 3 ілюстративно зображає присвоєння паралельно кодованих розбиттів вирізці або транспортувальному сегменту мережі;

Фіг. 4 зображає схематичну діаграму, яка показує загальну фрагментацію кадру з наближенням мозаїкового кодування для мінімальної наскрізної затримки;

Фіг. 5 зображає схематичну діаграму, яка показує ілюстративну фрагментацію кадру з наближенням WPP кодування для мінімальної наскрізної затримки;

Фіг. 6 зображає схему, яка показує сценарій розмови з використанням відеопослуг;

Фіг. 7 схематично зображає можливу годинну диспетчеризацію кодування, передачі даних і декодування для мозаїк з спільними підмножинами з мінімальною наскрізною затримкою;

Фіг. 8 схематично зображає годинну диспетчеризацію, яка зазвичай досягає наскрізної затримки;

Фіг. 9 зображає картинку, яка ілюстративно має 11×9 кодувальних деревовидних блоків, яка ділиться на дві вирізки;

Фіг. 10 зображає картинку, яка ілюстративно має 13×8 кодувальних деревовидних блоків, яка розбивається на три мозаїки;

Фіг. 11 зображає приклад для синтаксису множини значень параметра послідовності;

Фіг. 12 зображає приклад для синтаксису множини значень параметра послідовності;

Фіг. 13 зображає приклад для синтаксису заголовка вирізки;

Фіг. 14 ілюструє розбиття картинку для WPP на стандартну вирізку і для обробки з малою затримкою - на залежні вирізки;

Фіг. 15 зображає приклад для частини в синтаксисі множини значень параметра картинки;

Фіг. 16 зображає можливий синтаксис заголовка вирізки;

Фіг. 17 схематично зображає внутрішні залежності кодування для нормальних вирізок (і залежних вирізок);

Фіг. 18 зображає схематичну діаграму, яка порівнює кодування для передачі мозаїк з малою затримкою (паралельна хвильова обробка даних з використанням залежних вирізок);

Фіг. 19 зображає годинну диспетчеризацію, яка показує ілюстративне WPP кодування з передачею даних з малою затримкою по каналу зв'язку при використанні паралельної хвильової обробки даних з використанням залежних вирізок, як зображено справа на Фіг. 18;

Фіг. 20 зображає схематичну діаграму, яка показує покращення надійності шляхом використання стандартних вирізок як якорів;

Фіг. 21 зображає інший варіант виконання для синтаксису заголовка вирізки;

Фіг. 22 зображає інший варіант виконання для синтаксису множини значень параметра картинки;

Фіг. 23 зображає схематичну діаграму, яка показує спосіб ініціалізації ймовірності символу для залежної вирізки у випадку початку на лівій межі картинки;

Фіг. 24 зображає схему декодера;

Фіг. 25 схематично зображає схему декодера разом з схематичним зображенням розбиття картинку на кодувальні блоки і вирізки;

Фіг. 26 схематично зображає схему кодера;

Фіг. 27 схематично зображає картинку, розбиту на нормальні і залежні вирізки (тут названі сегментами вирізки);

Фіг. 28a і 28b схематично зображають картинку, розбиту на нормальні і залежні вирізки (тут названі сегментами вирізки), з одного боку, і мозаїки, з іншого боку;

Фіг. 29 зображає блок-схему, яка показує процес ініціалізації контексту з використанням залежних вирізок;

Фіг. 30 зображає блок-схему, яка показує процес зберігання контексту для використання залежних вирізок; і

Фіг. 31 схематично зображає різні можливості сигналізації WPP точок входу.

Далі, опис починається з опису сьогоднішніх концепцій для надання можливості паралельної обробки картинку і, відповідно, кодування з малою затримкою. Окреслюються проблеми, які з'являються при бажанні мати обидві можливості. Зокрема, як виявиться з наступного обговорення, концепція WPP вкладеного потоку, як зазначалося до цих пір, деяким чином конфліктує з бажанням мати малу затримку внаслідок необхідності передавати WPP вкладені потоки шляхом групування їх в одну вирізку. Наступні варіанти виконання реалізують концепції паралельної обробки даних, такі як концепція WPP, застосовувані для випадків, які потребують навіть меншу затримку шляхом розширення концепції вирізки, зокрема шляхом введення іншого типу вирізки, далі названими залежними вирізками.

Мінімізація наскрізної відеозатримки від уловлювання зображення до відображення його є однією з

головних цілей у застосуваннях, таких як відеоконференція і подібне.

Контур обробки сигналу для передачі цифрових відеоданих складається з відеокамери, уловлювального пристрою, кодера, інкапсулятора, передавача, демультіплексора, декодера, рендерера і відображального пристрою. Кожен з цих етапів робить внесок в наскрізну затримку шляхом буферизації даних зображення перед їх послідовною передачею на наступний етап.

Деякі застосування вимагають мінімізації такої затримки, наприклад дистанційного оперування об'єктами на небезпечних ділянках, без прямого огляду об'єкта, яким оперують, або мінімального докучливого втручання. Кожна мала затримка може приводити до серйозних труднощів, пов'язаних з належним оперуванням, або навіть приводити до катастрофічних помилок.

В багатьох випадках, на етапі обробки буферизується увесь відеокадр, наприклад, для забезпечення внутрішньокадрової обробки даних. Деякі етапи збирають дані для формування пакетів, які подаються на наступний етап. Загалом, існує нижня межа для затримки, яка впливає з вимог локальної обробки даних. Нижче детальніше аналізується цей факт для кожного окремого етапу.

Обробка даних всередині відеокамери обов'язково не вимагає внутрішньокадрової обробки сигналу таким чином, що мінімальна затримка забезпечується періодом інтегрування датчика, який обмежується частотою кадрів, і деякими виборами виробником структури апаратного засобу. Вихідні дані відеокамери типово зв'язуються з порядком проходження, згідно з яким обробка даних зазвичай починається у верхньому лівому куті, далі здійснюється до верхнього правого кута і продовжується рядок за рядком до нижнього правого кута. Тому, це займає приблизно період проходження одного кадру до тих пір, доки усі дані не будуть передані від датчика до виходу відеокамери.

Уловлювальний пристрій може передавати дані відеокамери зразу ж після їх прийому; однак, він буде типово буферизувати деяку кількість даних і генерувати пакети для оптимізації доступу даних до запам'ятовуючого пристрою або зберігального пристрою. Окрім того, з'єднання між відеокамерою/уловлювальним пристроєм і запам'ятовуючим пристроєм комп'ютера типово обмежує швидкість передачі бітів для передачі уловлених даних зображення до запам'ятовуючого пристрою для подальшої обробки (кодування). Типово, відеокамери з'єднані між собою за допомогою USB 2.0 або скоріше за допомогою USB 3.0, що будуть завжди частково передавати дані зображення до кодера. Це обмежує здатність до розпаралелювання процесу на кодувальній стороні з надзвичайно малою затримкою, тобто, кодер буде намагатися розпочинати кодування якомога скоріше, коли дані з відеокамери стануть доступними, наприклад в порядку проходження растру: зверху донизу зображення.

В кодері існує декілька ступенів свободи, які дозволяють узгоджувати ефективність кодування з точки зору швидкості передачі даних, необхідної для точності передачі певних відеоданих для зменшення затримки в обробці даних.

Кодер використовує дані, які вже були надіслані, для прогнозування зображення, яке потім кодується. Загалом, різниця між реальним зображенням і прогнозом може кодуватися меншою кількістю бітів, ніж, як це повинно потребуватися у випадку без прогнозування. Величини цього прогнозування повинні бути доступними для декодера, таким чином прогноз базується на попередньо декодованих частинах одного і того ж зображення (внутрішньокадрове прогнозування) або на інших зображеннях (міжкадрове прогнозування), які були оброблені раніше. В попередньому кодуванні відеоданих стандарти HEVC використовують тільки частину зображення зверху або в тому ж рядку, але ліву, яка була попередньо кодована, для внутрішньокадрового прогнозування, прогнозування вектора руху і ентропійного кодування (CABAC).

На додаток до оптимізації структури прогнозування, може розглядатися вплив паралельної обробки даних. Паралельна обробка даних вимагає ідентифікації ділянок картини, які можуть оброблятися незалежно. З практичних причин, вибирають суміжні ділянки, такі як горизонтальні або вертикальні прямокутники, які часто називають "мозаїками". У випадку обмеження малої затримки, такі ділянки повинні дозволяти розпаралелене кодування даних, які якомога скоріше надходять від уловлювального пристрою до запам'ятовуючого пристрою. Припускаючи передачу даних пам'яті в порядку проходження растру, мають сенс вертикальні розбиття необроблених даних для негайного початку кодування. Всередині таких мозаїк, які ділять картинку на вертикальні рядки (порівняти фігуру нижче), інтрапрогнозування, прогнозування вектора руху і ентропійне кодування (CABAC) можуть приводити до раціональної ефективності кодування. Для мінімізації затримки, тільки частина картини, починаючи зверху, повинна передаватися до пристрою для запам'ятовування кадру кодера і у вертикальних мозаїках повинна розпочинатися паралельна обробка даних.

Іншим способом забезпечення паралельної обробки даних є використання WPP в стандартній вирізці, що повинна порівнюватися з мозаїками, "рядком" мозаїк, включених в єдину вирізку. Дані у такій вирізці можна також паралельно обробляти з використанням WPP вкладених потоків у вирізці. Розбиття картини на вирізки 900 і мозаїки/WPP вкладені потоки 914, зображені на Фіг. 3/1, є прикладами.

Фіг. 3, таким чином, зображає присвоєння паралельно кодованих частин, таких як 906 або 914, вирізці або мережевому транспортному сегменту (єдиний мережевий пакет або множина мережевих

пакетів 900).

Інкапсуляція кодованих даних в блоки мережевого рівня абстракції (NAL), як визначено в стандарті H.264 або HEVC, перед передачею даних або під час процесу кодування додає деякий заголовок до блоків даних, який дозволяє ідентифікацію кожного блока і повторне упорядковування блоків, якщо це застосовується. В стандартному випадку, не вимагається додаткової сигналізації, оскільки порядок кодувальних елементів завжди відповідає порядку декодування, тобто, надається неявне присвоєння положення мозаїки або загального кодувального фрагмента.

Якщо паралельна обробка розглядається з додатковим транспортувальним шаром для паралельного транспортування з малою затримкою, тобто, транспортувальний шар може переупорядковувати ділянки картинки для мозаїк для забезпечення передачі даних з малою затримкою, означаючи передачу фрагментів, як зображено на Фіг. 4, коли вони кодуються. Такі фрагменти можуть також бути не повністю кодованими вирізками, вони можуть бути підмножиною вирізки або можуть міститися у залежній вирізці.

У випадку створення додаткових фрагментів, існує узгодження між ефективністю, яка повинна бути найвищою з великими блоками даних, оскільки інформація заголовку додає сталу кількість байтів, і затримкою, оскільки великі блоки даних паралельних кодерів повинні потребувати буферизації перед передачею. Загальна затримка може зменшуватися, якщо кодоване представлення вертикальних мозаїк 906 розбивається на ряд фрагментів 916, які передаються, як тільки фрагмент стає повністю кодованим. Розмір кожного фрагмента може визначатися в термінах фіксованої ділянки зображення, такої як макроблоки, LCUs, або в термінах максимальної кількості даних, як зображено на Фіг. 4.

Фіг. 4, таким чином, зображає загальну фрагментацію кадру з наближенням мозаїкового кодування для мінімальної наскрізної затримки.

Подібним чином, Фіг. 5 зображає фрагментацію кадру з WPP кодувальним наближенням для мінімальної наскрізної затримки.

Передача даних може додавати додаткову затримку, наприклад, якщо застосовується додаткова орієнтована на блоки обробка даних, така як коди прямого виправлення помилок, яка підвищує надійність передачі даних. Окрім того, інфраструктура мережі (маршрутизатори і так далі) або фізичний канал передачі даних можуть вносити затримку, для з'єднання це типово відоме як латентність. На додаток до латентності, швидкість передачі бітів визначає тривалість (затримку) передачі даних від сторони а до сторони б у спілкуванні, як зображено на Фіг. 6, яке використовує відеопослуги.

Якщо кодовані блоки даних передаються без порядку, то потрібно розглядати затримку у переупорядкуванні. Декодування може розпочинатися зразу з при надходженні блока даних, припускаючи, що доступні інші блоки даних, які потрібно декодувати перед цим.

У випадку мозаїк, між мозаїками відсутні залежності, таким чином, мозаїка може зразу ж декодуватися. Якщо фрагменти сформовані з мозаїки, такі як окремі вирізки на кожен фрагмент, як зображено на Фіг 4, то фрагменти можуть безпосередньо транспортуватися, як тільки вони відповідно стануть кодованими або їх LCUs або CUs стануть кодованими.

Рендерер збирає вихідні дані паралельних декодерів і направляє комбіновану картинку рядком за рядком до відображального пристрою.

Відображальний пристрій обов'язково не додає якусь затримку, але, на практиці, може виконувати деяку внутрішньокадрову обробку даних перед реальним відображенням даних зображення. Це залежить від виборів виробником структури апаратного засобу.

Підсумовуючи, для досягання мінімальної наскрізної затримки, можна впливати на етапи кодування, інкапсуляції, передачу даних та декодування. Якщо використовується паралельна обробка даних, мозаїки і фрагментація в мозаїках, то загальна затримка може значно зменшуватися, як зображено на Фіг. 7, порівняно із зазвичай використовуваним контуром обробки даних, який додає приблизно одну кадрову затримку на кожному з цих етапів, як зображено на Фіг. 8.

Зокрема, хоча Фіг. 7 зображає кодування, передачу даних і декодування для мозаїк із загальними підмножинами з мінімальною наскрізною затримкою, Фіг. 8 зображає загалом одержану наскрізну затримку.

Стандарт HEVC дозволяє використання розбиття вирізки, розбиття мозаїки наступним чином.

мозаїка: ціла кількість деревоподібних блоків, які сумісно розташовані в одному стовпчику і одному рядку, послідовно упорядковані в порядку проходження растру деревоподібного блоку мозаїки. Поділ кожної картини на мозаїки є розбиттям. Мозаїки в картинці послідовно упорядковані в порядку проходження растру картини. Хоча вирізка містить деревоподібні блоки, які розташовані послідовно в порядку проходження растру деревоподібного блоку мозаїки, ці деревоподібні блоки обов'язково не послідовно розташовані в порядку проходження растру деревоподібного блоку картини.

вирізка: ціла кількість деревоподібних блоків послідовно упорядковані в порядку проходження растру. Поділ кожної картини на вирізки є розбиттям. Адреси деревоподібних блоків одержуються з адреси першого деревоподібного блока у вирізці (як представлено у заголовку вирізки).

проходження растру: перетворення прямокутної двовимірної структури на одновимірну структуру

так, що перші вхідні дані в одновимірній структурі одержуються з першого верхнього рядка двовимірної структури, який проходиться зліва направо, подібним чином зліва направо проходиться другий, третій і так далі, рядки структури (донизу).

деревоподібний блок: блок $N \times N$ зразків яскравості і два відповідних блоки зразків кольоровості картини, яка має три масиви зразків або блок $N \times N$ зразків монохромної картини або картини, яка кодується з використанням трьох окремих кольорових площин. Поділ вирізки на деревоподібні блоки є розбиттям.

розбиття: поділ множини на підмножини так, що кожен елемент множини знаходиться точно в одній з підмножин.

квадрадерево: дерево, у якому батьківський вузол може розбиватися на чотири вузли-нащадки. Вузол-нащадок може стати батьківським вузлом для іншого розбиття на чотири вузла-нащадки.

Далі, пояснюється просторове підрозбиття картинок, вирізок і мозаїк. Зокрема, подальший опис уточнює як картинка розбивається на вирізки, мозаїки і кодувальні деревоподібні блоки. Картинки діляться на вирізки і мозаїки. Вирізка є послідовністю кодувальних деревоподібних блоків. Подібним чином, мозаїка є послідовністю кодувальних деревоподібних блоків.

Зразки обробляються в одиницях кодувальних деревоподібних блоків. Розмір масиву яскравості для кожного деревоподібного блока в зразках як по ширині, так і по висоті становить $CtbSize$. Ширина і висота масивів кольоровості для кожного кодувального деревоподібного блока становлять $CtbWidthC$ і, відповідно, $CtbHeightC$. Наприклад, картинка може ділитися на дві вирізки, як зображено на наступній фігурі. Як інший приклад, картинка може ділитися на три мозаїки, як зображено на другій наступній фігурі.

На відміну від вирізок, мозаїки є завжди прямокутними і завжди містять цілу кількість кодувальних деревоподібних блоків в порядку проходження растру кодувального деревоподібного блока. Мозаїка може складатися з кодувальних деревоподібних блоків, які містяться у більше ніж одній вирізці. Подібним чином, вирізка може містити кодувальні деревоподібні блоки, які містяться у більше ніж одній мозаїці.

Фіг. 9 зображає картинку 898 з 11×9 кодувальними деревоподібними блоками 918, яка розбивається на дві вирізки 900a, b.

Фіг. 10 зображає картинку з 13×8 кодувальними деревоподібними блоками 918, яка розбивається на три вирізки.

Кожен кодувальний деревоподібний блок 918 картини 898 присвоюється сигналу розбиття для ідентифікації розмірів блока для інтра- або інтерпрогнозування і для кодування з перетворенням. Розбиття є рекурсивним підрозбиттям квадрадерева. Корінь квадрадерева зв'язаний з кодувальним деревоподібним блоком. Квадрадерево розбивається до досягання листка, який називається кодувальним блоком. Кодувальний блок є кореневим вузлом двох дерев, дерева прогнозу і дерева перетворення.

Дерево прогнозу вказує положення і розмір прогнозувальних блоків. Прогнозувальні блоки і відповідні дані прогнозування названі прогнозувальною одиницею.

Фіг. 11 зображає ілюстративний RBSP (Дані Необробленої Послідовності Байтів) синтаксис множини значень параметра послідовності.

Дерево перетворення вказує положення і розмір блоків перетворення. Блоки перетворення і відповідні дані перетворення називаються одиницею перетворення.

Інформація про розбиття для яскравості і кольоровості ідентична для дерева прогнозування і може або може не бути ідентичною для дерева перетворення.

Кодувальний блок, відповідні дані кодування і відповідні одиниці прогнозування і перетворення формують разом кодувальну одиницю.

Спосіб перетворення адреси кодувального деревоподібного блока в порядку проходження растру кодувального деревоподібного блока на порядок проходження мозаїки може бути наступним:

Вихідними даними для цього способу є

- масив $CtbAddrTS[ctbAddrRS]$, з $ctbAddrRS$ в інтервалі $0-PicHeightInCtbs * PicWidthInCtbs-1$, включно.

- масив $TileId[ctbAddrTS]$, з $ctbAddrTS$ в інтервалі $0-PicHeightInCtbs * PicWidthInCtbs-1$, включно.

Масив $CtbAddrTS[]$ одержується наступним чином:

```

for( ctbAddrRS = 0; ctbAddrRS < PicHeightInCtbs * PicWidthInCtbs; ctbAddrRS++ ) {
    tbX = ctbAddrRS % PicWidthInCtbs
    tbY = ctbAddrRS / PicWidthInCtbs
    for( j = 0; j <= num_tile_columns_minus1; j++ )
        if( tbX < ColBd[ j + 1 ] )
            tileX = j
    for( i = 0; i <= num_tile_rows_minus1; i++ )
        if( tbY < RowBd[ i + 1 ] )
            tileY = i
    CtbAddrTS[ ctbAddrRS ] = ctbAddrRS - tbX
    for( j = 0; j < tileX; j++ )
        ctbAddrTS += RowHeight[ tileY ] * ColumnWidth[ j ]
    CtbAddrTS[ ctbAddrRS ] += ( tbY - RowBd[ tileY ] ) * ColumnWidth[ tileY ] + tbX - ColBd[ tileX ]
}

```

Масив TileId [] одержується наступним чином:

```

for( j = 0; tileId = 0; j <= num_tile_columns_minus1; j++ )
    for( i = 0; i <= num_tile_rows_minus1; i++, tileId++ )
        for( y = RowBd[ j ], y < RowBd[ j + 1 ]; y++ )
            for( x = ColBd[ i ], x < ColBd[ i + 1 ]; x++ )
                TileId[ CtbAddrTS[ y*PicWidthInCtbs + x ] ] = tileId

```

Відповідний ілюстративний синтаксис зображений на Фіг. 11, 12 і 13, при цьому Фіг. 12 має ілюстративний RBSP синтаксис множини значень параметра картинки. Фіг. 13 зображає ілюстративний синтаксис заголовка вирізки.

В прикладі синтаксису може застосовуватися наступна семантика:

entropy_slice_flag equal to 1 вказує, що відсутнє значення синтаксичних елементів заголовку вирізки дорівнює значенню синтаксичних елементів заголовку вирізки в попередній вирізці, де попередня вирізка визначається як вирізка, яка містить кодувальний деревоподібний блок з координатами (SliceCtbAddrRS-1). entropy_slice_flag повинен дорівнювати 0, коли SliceCtbAddrRS дорівнює 0.

tiles_or_entropy_coding_sync_idc equal to 0 вказує, що існує тільки одна мозаїка в кожній картинці в послідовності кодованих відеоданих і не викликається спеціальний процес синхронізації для змінних контексту перед декодуванням першого кодувального деревоподібного блока рядка кодувальних деревоподібних блоків.

tiles_or_entropy_coding_sync_idc equal to 1 вказує, що тут в кожній картинці може бути більше, ніж одна мозаїка в послідовності кодованих відеоданих, і не викликається спеціальний процес синхронізації змінних контексту перед декодуванням першого кодувального деревоподібного блока рядка кодувальних деревоподібних блоків.

tiles_or_entropy_coding_sync_idc equal to 2 вказує, що в кожній картинці існує тільки одна мозаїка в послідовності кодованих відеоданих, викликається спеціальний процес синхронізації для змінних контексту перед декодуванням першого кодувального деревоподібного блока рядка кодувальних деревоподібних блоків і викликається спеціальний процес збереження змінних контексту в запам'ятовуючому пристрої після декодування двох кодувальних деревоподібних блоків рядка кодувальних деревоподібних блоків.

Значення tiles_or_entropy_coding_sync_idc повинно становити 0-2, включно.

num_tile_columns_minus1 plus 1 вказує кількість стовпчиків мозаїк, які розбивають картинку.

num_tile_rows_minus1 plus 1 вказує кількість рядків мозаїк, які розбивають картинку.

Коли num_tile_columns_minus1 дорівнює 0, то num_tile_rows_minus1 не повинен бути рівним 0.

Одна або обидві з наступних умов повинні задовольнятися для кожної вирізки і мозаїки:

- Усі кодовані блоки у вирізці належать одній і тій же мозаїці.
- Усі кодовані блоки в мозаїці належать одній і тій же вирізці.

ПРИМІТКА - В одній і тій же картинці можуть бути присутніми обидві вирізки, які містять множину мозаїк, і мозаїки, які містять множину вирізок.

uniform_spacing_flag equal to 1 вказує, що межі стовпчика і, подібним чином, межі рядка однорідно розподілені по картинці. uniform_spacing_flag equal to 0 вказує, що межі стовпчика і, подібним чином, межі рядка однорідно не розподілені по картинці, але явно сигналізовані з використанням синтаксичних елементів column_width [i] і row_height [i].

column_width [i] вказує ширину i-го стовпчика мозаїки в одиницях кодувальних деревоподібних блоків.

row_height [i] вказує висоту i-го рядка мозаїки в одиницях кодувальних деревоподібних блоків. Значення ColumnWidth [i], який вказує ширину i-го стовпчика мозаїки в одиницях кодувальних

деревоподібних блоків, і значення ColumnWidthInLumaSamples [i], які вказують ширину i-го стовпчика мозаїки в одиницях зразків яскравості, одержуються наступним чином:

```
for( i = 0; i <= num_tile_columns_minus1; i++ ) {  
    if( uniform_spacing_flag )  
        ColumnWidth[ i ] = ( ( i + 1 ) * PicWidthInCtbs ) / ( num_tile_columns_minus1 + 1 ) -  
            ( i * PicWidthInCtbs ) / ( num_tile_columns_minus1 + 1 )  
    else  
        ColumnWidth[ i ] = column_width[ i ]  
    ColumnWidthInLumaSamples[ i ] = ColumnWidth[ i ] << Log2Ctbsize  
}
```

Значення RowHeight [i], які вказують висоту i-го рядка мозаїки в одиницях кодувальних деревоподібних блоків, одержуються наступним чином:

```
for( i = 0; i <= num_tile_rows_minus1; i++ )  
    if( uniform_spacing_flag )  
        RowHeight[ i ] = ( ( i + 1 ) * PicHeightInCtbs ) / ( num_tile_rows_minus1 + 1 ) -  
            ( i * PicHeightInCtbs ) / ( num_tile_rows_minus1 + 1 )  
    else  
        RowHeight[ i ] = row_height[ i ]
```

Значення ColBd [i], які вказують координати лівої межі i-го стовпчика мозаїки в одиницях кодувальних деревоподібних блоків, одержуються наступним чином:

```
for( ColBd[ 0 ] = 0, i = 0; i <= num_tile_columns_minus1; i++ )  
    ColBd[ i + 1 ] = ColBd[ i ] + ColumnWidth[ i ]
```

Значення RowBd [i], які вказують координати верхньої межі i-го рядка мозаїки в одиницях кодувальних деревоподібних блоків, одержуються наступним чином:

```
for( RowBd[ 0 ] = 0, i = 0; i <= num_tile_rows_minus1; i++ )  
    RowBd[ i + 1 ] = RowBd[ i ] + RowHeight[ i ]
```

num_substreams_minus1 plus 1 вказує максимальну кількість підмножин, включених у вирізку, коли tiles_or_entropy_coding_sync_idc дорівнює 2. Коли він відсутній, то величина num_substreams_minus1 дорівнює 0.

num_entry_point_offsets вказує кількість синтаксичних елементів entry_point_offset [i] у заголовку вирізки. Коли tiles_or_entropy_coding_sync_idc дорівнює 1, то величина num_entry_point_offsets повинна становити 0-(num_tile_columns_minus1+1) * (num_tile_rows_minus1+1)-1, включно. Коли tiles_or_entropy_coding_sync_idc дорівнює 2, то величина num_entry_point_offsets повинна становити 0-num_substreams_minus1, включно. Коли він відсутній, то величина num_entry_point_offsets дорівнює 0.

offset_len_minus1 plus 1 вказує довжину синтаксичних елементів entry_point_offset [i] в бітах.

entry_point_offset [i] вказує зміщення i-ї точки входу в байтах і повинна представлятися бітами offset_len_minus1 plus 1. Кодований блок NAL вирізки складається з підмножин num_entry_point_offsets+1, при цьому значення індексу яких становлять від 0 до num_entry_point_offsets, включно. Підмножина 0 складається з байтів 0-entry_point_offset [0] - 1 кодованого блока NAL вирізки, включно, при цьому підмножина k з k, яке становить 1-num_entry_point_offsets-1, включно, складається з байтів entry_point_offset [k-1] - entry_point_offset [k] + entry_point_offset [k-1] - 1 кодованого блока NAL вирізки, включно, і остання підмножина (з індексом підмножини, рівним num_entry_point_offsets) складається з решти байтів кодованого блока NAL вирізки.

ПРИМІТКА - Заголовок блока NAL і заголовок вирізки кодованого блока NAL вирізки завжди включаються в підмножину 0.

Коли tiles_or_entropy_coding_sync_idc дорівнює 1 і num_entry_point_offsets більший за 0, то кожна підмножина повинна містити усі кодовані біти однієї або множин повних мозаїк, а кількість підмножин повинна дорівнювати або бути меншою за кількість мозаїк у вирізці.

Коли tiles_or_entropy_coding_sync_idc дорівнює 2, а num_entry_point_offsets більша за 0, то підмножина k для кожної з усіх можливих величин k повинна містити усі біти, які використовуються під час процесу ініціалізації для поточного вказівника k бітового потоку.

Розглядаючи семантику даних вирізки, можна застосувати наступне.

end_of_slice_flag equal to 0 вказує, що інший макроблок є наступним у вирізці. end_of_slice_flag equal to 1 вказує кінець вирізки і що далі відсутній додатковий макроблок.

entry_point_marker_two_3bytes є фіксованою послідовністю з 3 байтів, яка дорівнює 0 × 000002. Цей синтаксичний елемент називається вхідним префіксом маркера.

tile_idx_minus_1 вказує TileID в порядку проходження растру. Перша мозаїка в картинці повинна мати TileID, рівний 0. Величина tile_idx_minus_1 повинна становити $0 - (\text{num_tile_columns_minus1} + 1) * (\text{num_tile_rows_minus1} + 1) - 1$.

Процес CABAC синтаксичного аналізу для даних вирізки може бути наступним:

Цей процес може вимагатися при виконанні синтаксичного аналізу синтаксичних елементів дескриптором ae(v).

Вхідними даними для цього процесу є запит на надання величини синтаксичного елемента і величин попередньо синтаксично проаналізованих синтаксичних елементів.

Виходом цього процесу є величина синтаксичного елемента.

При розпочинанні синтаксичного аналізу даних вирізки, вимагається процес ініціалізації CABAC процесу синтаксичного аналізу. Коли tiles_or_entropy_coding_sync_idc дорівнює 2 і num_substreams_minus1 більший за 0, то таблиця перетворення BitStreamTable з вхідними даними num_substreams_minus1+1, які визначають таблицю вказівників бітового потоку для використання для пізнішого одержання вказівника поточного бітового потоку, одержується наступним чином.

- BitStreamTable [0] ініціалізується для включення вказівника бітового потоку.

- Для усіх індексів i, більших за 0 і менших за num_substreams_minus1+1, BitStreamTable [i] містить вказівник бітового потоку для entry_point_offset [i] bytes після BitStreamTable [i - 1].

Поточний вказівник бітового потоку встановлюють рівним BitStreamTable [0].

Адреса мінімального кодувального деревоподібного блока, який містить просторово сусідній блок T, ctbMinCbAddrT, одержується з використанням координат (x0, y0) верхнього лівого зразка яскравості поточного кодувального деревоподібного блока, наприклад, наступним чином:

$$x = x0 + 2 \ll \text{Log2CtbSize} - 1,$$
$$y = y0 - 1,$$
$$\text{CtbMinCbAddrT} = \text{MinCbAddrZS} [x \gg \text{Log2MinCbSize}] [y \gg \text{Log2MinCbSize}].$$

Змінна availableFlagT одержується запитанням відповідного процесу одержання доступу до кодувального блока з CtbMinCbAddrT, як вхідною величиною.

При розпочинанні процесу синтаксичного аналізу кодувального дерева і tiles_or_entropy_coding_sync_idc is equal to 2 та num_substreams_minus1 is greater than 0, використовується наступне.

- Якщо CtbAddrRS % PicWidthInCtbs дорівнює 0, то застосовується наступне;

- Коли availableFlagT дорівнює 1, то вимагається процес синхронізації процесу CABAC синтаксичного аналізу, як це вказано в підпункті "Synchronization process for context variables".

- Процес декодування для бінарних рішень перед вимогою завершення слідує за процесом ініціалізації для арифметичного декодера.

- Вказівник поточного бітового потоку вказує BitStreamTable [i] з індексом i, одержаним наступним чином.

$$i = (\text{CtbAddrRS} / \text{PicWidthInCtbs}) \% (\text{num_substreams_minus1} + 1)$$

- Інакше, якщо CtbAddrRS % PicWidthInCtbs дорівнює 2, то вимагається процес введення даних в пам'ятовуючий пристрій в процесі CABAC синтаксичного аналізу, як це вказано в підпункті "Memorization process for context variables".

Процес ініціалізації може бути наступним:

Вихідні дані цього процесу є ініціалізованими CABAC внутрішніми змінними.

Спеціальні обробки його даних вимагаються при розпочинанні синтаксичного аналізу даних вирізки або при розпочинанні синтаксичного аналізу даних кодувального дерева і кодувальне дерево є першим кодувальним деревом в мозаїці.

Процес введення змінних контексту в пам'ятовуючий пристрій може бути наступним:

Вхідними даними цього процесу є CABAC змінні контексту, позначені індексом ctxldx.

Вихідні дані цього процесу є змінні TableStateSync і TableMPSSync, які містять значення змінних m і p, використовуваних в процесі ініціалізації змінних контексту, які присвоюються синтаксичним елементам, за виключенням прапорця кінця вирізки.

Для кожної змінної контексту відповідні вхідні дані n і m таблиць TableStateSync і TableMPSSync ініціалізуються для відповідних pStateldx і valMPS.

Процес синхронізації змінних контексту може бути наступним:

Вхідні дані цього процесу є змінними TableStateSync і TableMPSSync, які містять значення змінних n і m, використовуваних в процесі введення змінних контексту в пам'ятовуючий пристрій, які присвоюються синтаксичним елементам, за виключенням прапорця кінця вирізки.

Вихідні дані цього процесу є CABAC змінні контексту, позначені індексом ctxldx.

Для кожної змінної контексту відповідні змінні контексту pStateldx і valMPS ініціалізуються для відповідних вхідних величин n і m таблиць TableStateSync і TableMPSSync.

Далі пояснюється кодування з малою затримкою і транспортування з використанням WPP. Зокрема, нижченаведене обговорення вказує як транспортування з малою затримкою, як описано на Фіг. 7, може також застосовуватися до WPP.

Перш за все, важливо, що може надсилатися підмножина картинки перед завершенням усієї картинки. Зазвичай, це може досягатися з використанням вирізок, як вже показано на Фіг. 5.

Для зменшення затримки порівняно з мозаїками, як зображено на наступних фігурах, існує потреба у застосуванні єдиного WPP вкладеного потоку на рядок LCUs і, окрім того, дозволяє окрему передачу кожного з таких рядків. Для збереження високої ефективності кодування, не можна використовувати вирізки на кожен рядок/вкладений потік. Тому, нижче вводиться так звана залежна вирізка, як визначено в наступному розділі. Ця вирізка, наприклад, не має усіх полів повного заголовка вирізки стандарту HEVC, але має поля, використовувані для ентропійних вирізок. Окрім того, може бути присутнім перемикач для усунення розриву CABAC між рядками. У випадку WPP, використання CABAC контексту (стрілки на Фіг. 14) і прогнозування рядків повинно дозволятися для збереження виграшу ефективності кодування WPP по усіх мозаїках.

Зокрема, Фіг. 14 ілюструє картинку 10 для WPP у стандартній вирізці 900 (ділянка SL) і для обробки даних з малою затримкою у залежних вирізках (OS) 920.

На даний момент, стандарт HEVC, що розробляється, пропонує два типи розбиття з точки зору вирізок. Існує стандартна (нормальна) вирізка і ентропійна вирізка. Стандартна вирізка є повністю незалежним розбиттям картинки, за виключенням деяких залежностей, які можуть бути доступними внаслідок деблокування процесу фільтрування на межах вирізки. Ентропійна вирізка також незалежна, але тільки з точки зору ентропійного кодування. Ідеєю з Фіг. 14 є узагальнення концепції розбиття на вирізки. Таким чином, стандарт HEVC, що розробляється, повинен пропонувати два загальні типи вирізок: незалежна (стандартна) або залежна. Тому, вводиться новий тип вирізки: залежна вирізка.

Залежна вирізка є вирізкою, яка має залежності для попередньої вирізки. Залежності є спеціальними даними, які можуть використовуватися між вирізками в процесі ентропійного декодування і/або процесі відтворення елемента зображення.

На Фігурі 14 ілюстративно представлена концепція залежних вирізок. Картинка починається, наприклад, завжди зі стандартної вирізки. Відзначається, що у цій концепції поведінка стандартної вирізки трохи змінюється. Типово, в стандартах, таких як H264/AVC або HEVC, стандартна вирізка є повністю незалежним розбиттям і не повинна зберігати жодних даних після декодування, за виключенням деяких даних для деблокування процесу фільтрування. Проте обробка майбутньої залежної вирізки 920 можлива тільки завдяки посиланню на дані верхньої вирізки (тут в першому рядку): стандартна вирізка 900. Для встановлення цього, стандартні вирізки 900 повинні зберігати дані останнього рядка CU (кодувальна чарунка). Ці дані включають:

- CABAC дані кодера (стани моделі контексту однієї CU, з яких може ініціалізуватися процес ентропійного декодування залежної вирізки),
- усі декодовані синтаксичні елементи CUs для стандартного CABAC процесу декодування залежних CUs,
- дані інтрапрогнозування і прогнозування вектора руху.

Тому, кожна залежна вирізка 920 повинна виконувати ту ж процедуру - зберігати дані для майбутньої залежної вирізки в тій же картинці.

На практиці, ці додаткові етапи не повинні бути проблемою, оскільки процес декодування загалом зберігає деякі дані, такі як синтаксичні елементи.

В нижченаведених розділах представлені можливі зміни для синтаксису стандарту HEVC, які вимагаються для реалізації концепції залежних вирізок.

Фіг. 5, наприклад, зображає можливі зміни в синтаксисі RBSP множини значень параметра картинки.

Семантика множини значень параметра картинки для залежних вирізок може бути наступною:

`dependent_slices_present_flag` equal to 1 вказує, що картинка містить залежні вирізки і процес декодування кожної (стандартної або залежної) вирізки повинен зберігати стани ентропійного декодування і дані інтрапрогнозування і прогнозування вектора руху для наступної вирізки, яка може бути залежною вирізкою, яка може також відповідати стандартній вирізці. Наступна залежна вирізка може посилатися на такі збережені дані.

Фіг. 16 зображає можливий `slice_header syntax` із змінами відносно поточного статусу HEVC.

`dependent_slice_flag` equal to 1 вказує, що значення відсутніх синтаксичних елементів заголовка вирізки дорівнює значенню синтаксичних елементів заголовка вирізки в попередній (стандартній) вирізці, де попередня вирізка визначається як вирізка, яка містить кодувальний деревоподібний блок з координатами (`SliceCtbAddrRS-1`). `dependent_slice_flag` повинен дорівнювати 0, коли `SliceCtbAddrRS` дорівнює 0.

`no_cabac_reset_flag` equal to 1 вказує CABAC ініціалізацію із збереженого стану попередньо декодованої вирізки (і не початковими значеннями). Інакше, тобто, якщо 0, то CABAC ініціалізація не залежить від жодного стану попередньо декодованої вирізки, тобто з початковими значеннями.

`last_ctb_cabac_init_flag` equal to 1 вказує CABAC ініціалізацію із збереженого стану останнього кодованого деревоподібного блока попередньо декодованої вирізки (наприклад, для мозаїк завжди

дорівнює 1). Інакше (дорівнює 0), дані ініціалізації вказуються із збереженого стану другого кодованого деревоподібного блока останнього (сусіднього) ctb-рядка попередньо декодованої вирізки, якщо перший кодований деревоподібний блок поточної вирізки є першим кодованим деревоподібним блоком в рядку (тобто, WPP режим), інакше, CABAC ініціалізація виконується із збереженого стану останнього кодованого деревоподібного блока попередньо декодованої вирізки.

Нижче надається порівняння залежних вирізок та інших схем розбиття (інформативних).

На Фіг. 17 зображується відмінність між нормальною і залежною вирізкою.

Можливе кодування і передачу WPP вкладених потоків в залежних вирізках (DS), як описано стосовно Фіг. 18, порівнюють з кодуванням для транспортування мозаїк (ліві) з малою затримкою і WPP/DS (праві). Товсті хрестики, зображені товстими суцільними лініями, на Фіг. 18 зображають однаковий момент часу для двох способів, припускаючи, що кодування WPP рядка займає той же час що й кодування єдиної мозаїки. Завдяки кодувальним залежностям, готовим є тільки перший рядок WPP після кодування усіх мозаїк. Проте використання наближення залежної вирізки дозволяє WPP наближенню передавати перший рядок зразу ж після його кодування. Це відрізняється від більш ранніх присвоєнь WPP вкладеного потоку, "вкладений потік" визначається для WPP як конкатенація рядків CU вирізки, які WPP декодуються тим же потоком декодера, тобто, тим же ядром/процесором. Хоча перед цим повинен також бути можливим вкладений потік на рядок і на ентропійну вирізку, ентропійна вирізка порушує залежності ентропійного кодування і, тому, має низьку ефективність кодування, тобто втрачається виграш WPP ефективності.

Окрім того, відмінність у затримці між обома наближеннями може бути реально малою, припускаючи передачу даних, як це зображено на Фіг. 19. Зокрема, Фіг. 19 зображає WPP кодування з передачею даних по каналу інформації з малою затримкою.

Припускаючи, що кодування останніх двох CUs DS #1.1 в WPP наближенні на Фіг. 18 не займає більше часу, ніж передача першого рядка SL #1, між мозаїками і WPP відсутня відмінність у випадку малої затримки. Проте ефективність WP/DS перевищує ефективність концепції мозаїки.

Для підвищення надійності для WPP режиму з малою затримкою, Фіг. 20 вказує, що покращення надійності досягається використанням стандартних вирізок (RS) як якорів. На картинці, зображеній на Фіг. 20, за (стандартною) вирізкою (RS) слідує залежні вирізки (DS). Тут, (стандартна) вирізка функціонує як якорь для порушення залежностей для попередніх вирізок, тому, забезпечується більша надійність в таких точках вставлення (стандартною) вирізки. В принципі, це жодним чином не відрізняється від вставлення (стандартних) вирізок.

Концепція залежних вирізок може також втілюватися наступним чином.

Тут, Фіг. 21 зображає можливий синтаксис заголовка вирізки.

Семантика заголовка вирізки є наступною:

`dependent_slice_flag` equal to 1 вказує, що значення кожного відсутнього синтаксичного елемента заголовка вирізки дорівнює значенню відповідного синтаксичного елемента заголовка вирізки в попередній вирізці, яка містить кодувальний деревоподібний блок, адресою якого є `SliceCtbAddrRS-1`. Коли він відсутній, то значення `dependent_slice_flag` дорівнює 0. Значення `dependent_slice_flag` повинно дорівнювати 0, коли `SliceCtbAddrRS` дорівнює 0.

`slice_address` вказує адресу в роздільній здатності вирізки, у якій починається вирізка. Довжина синтаксичного елемента `slice_address` становить $(\text{Ceil}(\log_2(\text{PicWidthInCtbs} * \text{PicHeightInCtbs})) + \text{SliceGranularity})$ біт.

Змінна `SliceCtbAddrRS`, яка вказує кодувальний деревоподібний блок, у якому починається вирізка в порядку проходження растру кодувального деревоподібного блока, одержується наступним чином:

$\text{SliceCtbAddrRS} = (\text{slice_address} \gg \text{SliceGranularity})$.

Змінна `SliceCbAddrZS`, яка вказує адресу першого кодувального блока у вирізці в гранулярності мінімального кодувального блока в порядку z-проходження, одержується наступним чином:

$\text{SliceCbAddrZS} = \text{slice_address}$

$\ll ((\log_2(\text{diff_max_min_coding_block_size} - \text{SliceGranularity})) \ll 1)$.

Декодування вирізки починається з найбільш можливого кодувального блока або в інших термінах з CTU в початковій координаті вирізки.

`first_slice_in_pic_flag` вказує на те, чи є вирізка першою вирізкою картинки. Якщо `first_slice_in_pic_flag` дорівнює 1, то змінні `SliceCbAddrZS` і `SliceCtbAddrRS` обидві встановлюються в 0 і декодування починається з першого кодувального деревоподібного блока в картинці.

`pic_parameter_set_id` вказує застосовану множину значень параметра картинки. Значення `pic_parameter_set_id` повинно становити 0-255 включно.

`num_entry_point_offsets` вказує кількість синтаксичних елементів `entry_point_offset[i]` в заголовку вирізки. Коли `tiles_or_entropy_coding_sync_idc` дорівнює 1, то значення `num_entry_point_offsets` повинно становити $0 - (\text{num_tile_columns_minus1} + 1) * (\text{num_tile_rows_minus1} + 1) - 1$, включно. Коли `tiles_or_entropy_coding_sync_idc` дорівнює 2, то значення `num_entry_point_offsets` повинно становити $0 - \text{PicHeightInCtbs} - 1$, включно. Коли він відсутній, то значення `num_entry_point_offsets` дорівнює 0.

`offset_len_minus1` plus 1 вказує довжину синтаксичних елементів `entry_point_offset[i]` в бітах.

entry_point_offset [i] вказує зміщення i-ї точки входу в байтах і повинне представлятися offset_len_minus1 plus 1 бітами. Кодовані дані вирізки після заголовка вирізки складаються з підмножин num_entry_point_offsets+1, при цьому значення індексу підмножини становлять від 0 до num_entry_point_offsets, включно. Підмножина 0 складається з байтів від 0 до entry_point_offset [0] - 1 кодованих даних вирізки, включно, при цьому підмножина k, де k становить 1-num_entry_point_offsets-1, включно, складається з байтів entry_point_offset [k-1] - entry_point_offset [k] + entry_point_offset [k-1] - 1 кодованих даних вирізки, включно, а остання підмножина (з індексом підмножини, рівним num_entry_point_offsets) складається з решти байтів кодованих даних вирізки.

Коли tiles_or_entropy_coding_sync_idc дорівнює 1 і num_entry_point_offsets більший за 0, то кожна підмножина повинна містити усі кодовані біти саме однієї мозаїки, а кількість підмножин (тобто, значення num_entry_point_offsets+1) повинна дорівнювати або бути меншою за кількість мозаїк у вирізці.

ПРИМІТКА - Коли tiles_or_entropy_coding_sync_idc дорівнює 1, то кожна вирізка повинна містити або підмножину однієї мозаїки (у цьому випадку не потрібна сигналізація точок входу) або цілу кількість повних мозаїк.

Коли tiles_or_entropy_coding_sync_idc дорівнює 2 і num_entry_point_offsets більший за 0, то кожна підмножина k з k, яке становить 0-num_entry_point_offsets-1, включно, повинна містити усі кодовані біти точно одного рядка кодувальних деревоподібних блоків, остання підмножина (з індексом, рівним num_entry_point_offsets) повинна містити усі кодовані біти решти кодувальних блоків, включених у вирізку, при цьому решта кодувальних блоків складаються або з точно одного рядка кодувальних деревоподібних блоків або підмножини одного рядка кодувальних деревоподібних блоків, а кількість підмножин (тобто, значення num_entry_point_offsets+1) повинна дорівнювати кількості рядків кодувальних деревоподібних блоків у вирізці, при цьому також підраховується підмножина одного рядка кодувальних деревоподібних блоків у вирізці.

ПРИМІТКА - Коли tiles_or_entropy_coding_sync_idc дорівнює 2, то вирізка може містити певну кількість рядків кодувальних деревоподібних блоків і підмножину рядка кодувальних деревоподібних блоків. Наприклад, якщо вирізка містить два з половиною рядка кодувальних деревоподібних блоків, то кількість підмножин (тобто, значення num_entry_point_offsets+1) повинна дорівнювати 3.

Синтаксис Rbsp відповідної множини значень параметра картинки може вибиратися, як зображено на Фіг. 22.

Семантика Rbsp множини значень параметра картинки може бути наступною:

dependent_slice_enabled_flag equal to 1 вказує присутність синтаксичного елемента dependent_slice_flag у заголовку вирізки для кодованих картинок, які посилаються на множину значень параметра картинки. dependent_slice_enabled_flag equal to 0 вказує відсутність синтаксичного елемента dependent_slice_flag у заголовку вирізки для кодованих картинок, які посилаються на множину значень параметра картинки. Коли tiles_or_entropy_coding_sync_idc дорівнює 3, то величина dependent_slice_enabled_flag повинна дорівнювати 1.

tiles_or_entropy_coding_sync_idc equal to 0 вказує, що повинна бути тільки одна мозаїка в кожній картинці, яка посилається на множину значень параметра картинки, не повинен викликатися спеціальний процес синхронізації для змінних контексту перед декодуванням першого кодувального деревоподібного блока рядка кодувальних деревоподібних блоків в кожній картинці, які посилаються на множину значень параметра картинки, і величини cabac_independent_flag і dependent_slice_flag для кодованих картинок, які посилаються на множину значень параметра картинки, не повинні обое дорівнювати 1.

Відзначається, що коли cabac_independent_flag і dependent_slice_flag обидва дорівнюють 1 для вирізки, то вирізка є ентропійною вирізкою.

tiles_or_entropy_coding_sync_idc equal to 1 вказує, що може бути більше, ніж одна мозаїка в кожній картинці, яка посилається на множину значень параметра картинки, не повинен викликатися спеціальний процес синхронізації для змінних контексту перед декодуванням першого кодувального деревоподібного блока рядка кодувальних деревоподібних блоків в кожній картинці, які посилаються на множину значень параметра картинки, і значення cabac_independent_flag і dependent_slice_flag для кодованих картинок, які посилаються на множину значень параметра картинки, не повинні обое дорівнювати 1.

tiles_or_entropy_coding_sync_idc equal to 2 вказує, що повинна бути тільки одна мозаїка в кожній картинці, яка посилається на множину значень параметра картинки, спеціальний процес синхронізації для змінних контексту повинен викликатися перед декодуванням першого кодувального деревоподібного блока рядка кодувальних деревоподібних блоків в кожній картинці, які посилаються на множину значень параметра картинки, і спеціальний процес введення змінних контексту в запам'ятовуючий пристрій повинен викликатися після декодування двох кодувальних деревоподібних блоків рядка кодувальних деревоподібних блоків в кожній картинці, які посилаються на множину значень параметра картинки, і значення cabac_independent_flag і dependent_slice_flag для кодованих картинок, які посилаються на множину значень параметра картинки, не повинні обидва дорівнювати 1.

`tiles_or_entropy_coding_sync_idc` equal to 3 вказує, що повинна бути тільки одна мозаїка в кожній картинці, яка посилається на множину значень параметра картинки, не повинен викликатися спеціальний процес синхронізації для змінних контексту перед декодуванням першого кодувального деревоподібного блока рядка кодувальних деревоподібних блоків в кожній картинці, які посилаються на множину значень параметра картинки, і величини `cabac_independent_flag` і `dependent_slice_flag` для кодованих картинок, які посилаються на множину значень параметра картинки, можуть обидва дорівнювати 1.

Коли `dependent_slice_enabled_flag` повинен дорівнювати 0, то `tiles_or_entropy_coding_sync_idc` не повинен дорівнювати 3.

Вимога узгодженості бітового потоку полягає в тому, що величина `tiles_or_entropy_coding_sync_idc` повинна бути однаковою для множин значень параметра картинки, які активуються в кодованій послідовності відеоданих.

Для кожної вирізки, яка посилається на множину значень параметра картинки, коли `tiles_or_entropy_coding_sync_idc` дорівнює 2 і перший кодувальний блок у вирізці не є першим кодувальним блоком у першому кодувальному деревоподібному блоці рядка кодувальних деревоподібних блоків, то останній кодувальний блок у вирізці повинен належати тому ж рядку кодувальних деревоподібних блоків що й перший кодувальний блок у вирізці.

`num_tile_columns_minus1` plus 1 вказує кількість стовпчиків мозаїки, які розбивають картинку.

`num_tile_rows_minus1` plus 1 вказує кількість рядків мозаїки, які розбивають картинку.

Коли `num_tile_columns_minus1` дорівнює 0, то `num_tile_rows_minus1` не повинен дорівнювати 0. `uniform_spacing_flag` equal to 1 вказує, що межі стовпчика *i*, подібним чином, межі рядка розподілені однорідно по картинці. `uniform_spacing_flag` equal to 0 вказує, що межі стовпчика *i*, подібним чином, межі рядка не розподілені однорідно по картинці, але явно сигналізуються з використанням синтаксичних елементів `column_width[i]` і `row_height[i]`.

`column_width[i]` вказує ширину *i*-го стовпчика мозаїки в одиницях кодувальних деревоподібних блоків.

`row_height[i]` вказує висоту *i*-го рядка мозаїки в одиницях кодувальних деревоподібних блоків.

Вектор `colWidth[i]` вказує ширину *i*-го стовпчика мозаїки в одиницях CTBs зі стовпчиком *i*, який становить від 0 до `num_tile_columns_minus1`, включно.

Вектор `CtbAddrRStoTS[ctbAddrRS]` вказує перехід від CTB адреси в порядку проходження растру до CTB адреси в порядку проходження мозаїки з індексом `ctbAddrRS`, який становить від 0 до $(\text{picHeightInCtbs} * \text{picWidthInCtbs}) - 1$, включно.

Вектор `CtbAddrTstoRS[ctbAddrTS]` вказує перехід від CTB адреси в порядку проходження мозаїки до CTB адреси в порядку проходження растру з індексом `ctbAddrTS`, який становить від 0 до $(\text{picHeightInCtbs} * \text{picWidthInCtbs}) - 1$, включно.

Вектор `TileId[ctbAddrTS]` вказує перехід від CTB адреси в порядку проходження мозаїки до мозаїки з індексом `ctbAddrTS`, який становить від 0 до $(\text{picHeightInCtbs} * \text{picWidthInCtbs}) - 1$, включно.

Величини `colWidth`, `CtbAddrRStoTS`, `CtbAddrTstoRS` і `TileId` одержуються шляхом викликання CTB растру і процесу проходження вирізки з переходом з `PicHeightInCtbs` і `PicWidthInCtbs` як вхідними величинами, а вихідні дані присвоюються `colWidth`, `CtbAddrRStoTS` і `TileId`.

Значення `ColumnWidthInLumaSamples[i]`, який вказує ширину *i*-го стовпчика мозаїки в одиницях зразків яскравості, встановлюються рівними `colWidth[i] << Log2CtbSize`.

Масив `MinCbAddrZS[x][y]`, який вказує перехід від координат (*x*, *y*) в одиницях мінімальних CBs до адреси мінімального CB в порядку *z*-проходження з *x*, яке становить від 0 до `picWidthInMinCbs-1`, включно, і *y*, який становить від 0 до `picHeightInMinCbs-1`, включно, одержують шляхом викликання процесу ініціалізації масиву з порядком *z*-проходження з використанням `Log2MinCbSize`, `Log2CtbSize`, `PicHeightInCtbs`, `PicWidthInCtbs` і вектора `CtbAddrRStoTS` як вхідних величин, а вихідна величина присвоюється `MinCbAddrZS`.

`loop_filter_across_tiles_enabled_flag` equal to 1 вказує, що операції внутрішньоконтурного фільтрування виконуються крізь межі мозаїки. `loop_filter_across_tiles_enabled_flag` equal to 0 вказує, що операції внутрішньоконтурного фільтрування не виконуються крізь межі мозаїки. Операції внутрішньоконтурного фільтрування включають деблокування фільтра, адаптивний до зразка зсув і адаптивні операції фільтрування контуру. Коли він відсутній, то величина `loop_filter_across_tiles_enabled_flag` дорівнює 1.

`cabac_independent_flag` equal to 1 вказує, що CABAC декодування кодувальних блоків у вирізці не залежить від жодного стану попередньо декодованої вирізки. `cabac_independent_flag` equal to 0 вказує, що CABAC декодування кодувальних блоків у вирізці залежить від станів попередньо декодованої вирізки. Коли він відсутній, то величина `cabac_independent_flag` дорівнює 0.

Процес одержання для доступності кодувального блока з адресою мінімального кодувального блока може бути наступним:

Вхідними величинами для цього процесу є

- адреса мінімального кодувального блока `minCbAddrZS` в порядку *z*-проходження,

- поточна адреса мінімального кодувального блока currMinCbAddrZS в порядку z-проходження.

Виходом цього процесу є доступність кодувального блока з адресою мінімального кодувального блока cbAddrZS в порядку cbAvailable z-проходження.

ПРИМІТКА 1 - Значення доступності визначається, коли викликається цей процес.

ПРИМІТКА 2 - Будь-який кодувальний блок, не зважаючи на його розмір, зв'язаний з адресою мінімального кодувального блока, яка є адресою кодувального блока з мінімальним розміром в порядку z-проходження.

- Якщо одна або більша кількість наступних умов є вірними, то cbAvailable встановлюється в положення FALSE;

- inCbAddrZS менший, ніж 0;

- inCbAddrZS більший за currMinCbAddrZS;

- кодувальний блок з адресою мінімального кодувального блока minCbAddrZS належить іншій вирізці, ніж кодувальний блок з поточною адресою мінімального кодувального блока currMinCbAddrZS і dependent_slice_flag вирізки, яка містить кодувальний блок з поточною адресою мінімального кодувального блока currMinCbAddrZS, дорівнює 0;

- кодувальний блок з адресою мінімального кодувального блока minCbAddrZS міститься в іншій вирізці, ніж кодувальний блок з поточною адресою мінімального кодувального блока currMinCbAddrZS;

- інакше, cbAvailable встановлюється в положення TRUE.

Процес CABAC синтаксичного аналізу для даних вирізки може бути наступним:

Цей процес вимагається при проведенні синтаксичного аналізу певних синтаксичних елементів дескриптором ae(v).

Вхідними даними для цього процесу є запит на надання величини синтаксичного елемента і величин попередньо синтаксично проаналізованих синтаксичних елементів.

Виходом цього процесу є величина синтаксичного елемента.

При розпочинанні синтаксичного аналізу даних вирізки вимагається процес ініціалізації CABAC процесу синтаксичного аналізу.

Фіг. 23 зображає, як просторово сусідній блок T використовується для запитування процесу одержання доступності кодувального деревоподібного блока відносно поточного кодувального деревоподібного блока (інформативного).

Адресу мінімального кодувального блока кодувального деревоподібного блока, який містить просторово сусідній блок T (Фіг. 23) (ctbMinCbAddrT), одержують з використанням координат (x0, y0) верхнього лівого зразка яскравості поточного кодувального деревоподібного блока наступним чином:

$x = x0 + 2 \ll \text{Log2CtbSize} - 1,$

$y = y0 - 1,$

$\text{ctbMinCbAddrT} = \text{MinCbAddrZS} [x \gg \text{Log2MinCbSize}] [y \gg \text{Log2MinCbSize}].$

Змінна availableFlagT одержується запитуванням процесу одержання доступності кодувального блока з CtbMinCbAddrT як вхідною величиною.

При розпочинанні синтаксичного аналізу кодувального дерева, як вказано, застосовуються наступні упорядковані етапи.

Арифметичний декодер ініціалізується наступним чином.

Якщо CtbAddrRS дорівнює slice_address, dependent_slice_flag дорівнює 1 і entropy_coding_reset_flag дорівнює 0, то застосовується наступне.

Процес синхронізації CABAC процесу синтаксичного аналізу викликається з TableStateldxDS і TableMPSValDS як вхідними величинами.

Викликається процес декодування для бінарних рішень перед завершенням, за яким слідує процес ініціалізації для арифметичного декодера.

Інакше, якщо tiles_or_entropy_coding_sync_idc дорівнює 2 і CtbAddrRS % PicWidthInCtbs дорівнює 0, то застосовується наступне.

Коли availableFlagT дорівнює 1, то процес синхронізації CABAC процесу синтаксичного аналізу викликається з TableStateldxWPP і TableMPSValWPP як вхідними величинами.

Викликається процес декодування для бінарних рішень перед завершенням, за яким слідує процес для арифметичного декодера.

Коли cabac_independent_flag дорівнює 0 і dependent_slice_flag дорівнює 1, або коли tiles_or_entropy_coding_sync_idc дорівнює 2, то процес введення даних в запам'ятовуючий пристрій використовується наступним чином.

Коли tiles_or_entropy_coding_sync_idc дорівнює 2 і CtbAddrRS % PicWidthInCtbs дорівнює 2, то викликається процес введення в запам'ятовуючий пристрій даних CABAC процесу синтаксичного аналізу з TableStateldxWPP і TableMPSValWPP як вихідними величинами.

Коли cabac_independent_flag дорівнює 0, dependent_slice_flag дорівнює 1 і end_of_slice_flag дорівнює 1, то вимагається процес введення в запам'ятовуючий пристрій даних CABAC процесу синтаксичного аналізу з TableStateldxDS і TableMPSValDS як вихідними величинами.

Синтаксичний аналіз синтаксичних елементів виконується наступним чином:

Для кожної запитаної величини синтаксичного елемента одержують двійкове представлення.

Бінаризація для синтаксичного елемента і послідовність синтаксично проаналізованих інформаційних величин визначає послідовність операцій декодування.

Для кожної інформаційної величини бінаризації синтаксичного елемента, яка індексується змінним `binIdx`, одержують індекс контексту `ctxIdx`.

Для кожного індексу `ctxIdx` викликається процес арифметичного декодування.

Одержувана послідовність (`b0...bbinIdx`) синтаксично проаналізованих інформаційних величин порівнюється з множиною послідовностей інформаційних величин, які надаються процесом бінаризації після декодування кожної інформаційної величини. Коли послідовність відповідає послідовності інформаційних величин у заданій множині, то синтаксичному елементу присвоюється відповідна величина.

У випадку, коли запит на надання величини синтаксичного елемента обробляється для синтаксичного елемента `pcm-flag` і декодована величина `pcm_flag` дорівнює 1, то декодер ініціалізується після декодування будь-якого `pcm_alignment_zero_bit`, `num_subsequent_pcm` і усіх даних `pcm_sample_luma` і `pcm_sample_chroma`.

Таким чином, вищенаведений опис розкриває декодер, як це зображено на Фіг. 24. Цей декодер, який головним чином вказаний позицією 5, відтворює картинку 10 з потоку даних 12, у якому картинка 10 кодується в одиницях вірсівок 14, на які розбивається картинка 10, при цьому декодер 5 конфігурується для декодування вірсівок 14 з потоку даних 12 у відповідності з порядком 16 вірсівок. Зазвичай, декодер 5 не обмежується послідовним декодуванням вірсівок 14. Скоріше, декодер 5 може використовувати паралельну хвильову обробку даних для декодування вірсівок 14 за умови, що картинка 10, які розбиваються на вірсівки 14, підходять для паралельної хвильової обробки даних. Відповідно, декодер 5 може, наприклад, бути декодером, який здатен декодувати вірсівки 14 паралельно в шаховому порядку, починаючи з декодування вірсівок 14, беручи до уваги порядок 16 вірсівок, для надання можливості хвильової обробки даних, як це було описано вище і буде також описуватися далі.

Декодер 5 реагує на частину 18 синтаксичного елемента в поточній вірсізці вірсівок 14 для декодування поточної вірсізки у відповідності з одним з принаймні двох режимів 20 і 22. У відповідності з першим з принаймні двох режимів, зокрема режимом 20, поточна вірсізка декодується з потоку даних 12 з використанням адаптивного до контексту ентропійного декодування, яке включає одержання контексту крізь межі вірсізки, тобто, крізь пунктирні лінії на Фіг. 24, тобто шляхом використання інформації, яка одержується з кодування/декодування інших, "згідно з порядком 16 вірсівок", "попередніх вірсівок". Окрім того, декодування поточної вірсізки з потоку даних 12 з використанням першого режиму 20 включає безперервне оновлення ймовірностей символів кодера і ініціалізацію ймовірностей символів на початку декодування поточної вірсізки, яка залежить від збережених станів ймовірностей символів попередньо декодованої вірсізки. Така залежність була описана вище, наприклад, у зв'язку з "процесом синхронізації для змінних кодера". Нарешті, перший режим 20 також включає прогнозувальне декодування крізь межі вірсізки. Таке прогнозувальне декодування крізь межі вірсізки може, наприклад, включати інтрапрогнозування крізь межі вірсізки, тобто, прогнозування величин зразка в поточній вірсізці на основі вже відновлених величин зразка "в порядку 16 вірсівок", попередньої вірсізки або прогнозування кодувальних параметрів крізь межі вірсізки, таке як прогнозування векторів руху, режими прогнозування, режими кодування або подібне.

У відповідності з другим режимом 22 декодер 5 декодує поточну вірсізку, тобто, вірсізку, яка на даний момент декодується, з потоку даних 12 з використанням адаптивного до контексту ентропійного декодування з обмеженням, однак, одержання контекстів для не перетинання меж вірсізки. Якщо у будь-якому випадку, наприклад, шаблон сусідніх положень, використовуваний для одержання контексту для певного синтаксичного елемента, який належить до блока в поточній вірсізці, входить у сусідню вірсізку, таким чином, перетинаючи межу поточної вірсізки, то відповідний атрибут відповідної частини сусідньої вірсізки, такий як величина відповідного синтаксичного елемента цієї сусідньої частини сусідньої вірсізки, встановлюється у значення за замовчуванням для усунення внутрішніх залежностей між поточною вірсізкою і сусідніми вірсізками. Хоча безперервне оновлення ймовірностей символів контекстів може відбуватися саме як у випадку першого режиму 20, ініціалізація ймовірностей символів у другому режимі 22 не залежить від жодної попередньо декодованої вірсізки. Окрім того, прогнозувальне декодування виконується з обмеженням прогнозувального декодування для не перетинання меж вірсізки.

Для полегшення розуміння опису Фіг. 24 і наступного опису, посилання робиться на Фіг. 25, яка показує можливе втілення декодера 5 в більш конструктивному сенсі, ніж на Фіг. 24. Як і у випадку з Фіг. 24, декодер 5 є прогнозувальним декодером, який використовує адаптивне до контексту ентропійне декодування для декодування потоку даних для одержання, наприклад, залишку прогнозування і параметрів прогнозування.

Як зображено на Фіг. 25, декодер 5 може містити ентропійний декодер 24, модуль 26 деквантизації

і оберненого перетворення, об'єднувальний модуль 28, втілений, як зображено на Фіг. 25, наприклад, як суматор і предиктор 28. Ентропійний декодер 24, модуль 26 і суматор 27 послідовно з'єднані між входом і виходом декодера 5 в порядку їх згадування, а предиктор 28 з'єднаний між виходом суматора 28 і додатковим його входом для формування прогнозувального контуру разом з об'єднувальним модулем 27. Тому, вихід декодера 24 додатково з'єднаний з входом предиктора 28 для кодувального параметра.

Хоча Фіг. 25 надає враження, що декодер послідовно декодує поточну картинку, декодер 5 може, наприклад, втілюватися для паралельного декодування картинки 10. Декодер 5 може, наприклад, містити багато ядер, кожне з яких працює згідно з елементами 24-28 на Фіг. 25. Однак, паралельна обробка є необов'язковою і послідовно працюючий декодер 5 також здатен декодувати потік даних, який надходить на вхід ентропійного декодера 24.

Для ефективного досягання тільки що згаданої здатності або послідовного або паралельного декодування поточної картинки 10, декодер 5 працює в одиницях кодувальних блоків 30 для декодування картинки 10. Кодувальні блоки 30 є, наприклад, листяними блоками, на які розбиваються кодувальні деревоподібні блоки або найбільші кодувальні блоки 32 за допомогою рекурсивного мультидеревного розбиття, такого як квадрандеревного розбиття. Кодувальні деревоподібні блоки 32, у свою чергу, можуть рівномірно розташовуватися у стовпчиках і рядках для формування рівномірного розбиття картинки 10 на ці кодувальні деревоподібні блоки 32. На Фіг. 25 кодувальні деревоподібні блоки 32 зображені суцільними лініями, тоді як кодувальні блоки 30 зображені пунктирними лініями. Для ілюстрації, зображено просто один кодувальний деревоподібний блок 32, який додатково розбивається на кодувальні блоки 30, тоді як інші кодувальні деревоподібні блоки 32 зображені, замість цього, без додаткового розбиття для безпосереднього формування кодувального блока. Потік даних 12 може містити синтаксичну частину, яка сигналізує, як картинка 10 розбивається на кодувальні блоки 30.

Потік даних 12 переносить для кожного кодувального блока 30 синтаксичні елементи, які розкривають як модулі 24-28 повинні відновлювати зміст картинки в такому кодувальному блоці 30. Наприклад, ці синтаксичні елементи містять:

1) необов'язково дані розбиття для подальшого розбиття кодувального блока 30 на прогнозувальні блоки,

2) необов'язково дані розбиття для додаткового розбиття кодувального блока 30 на залишкові блоки і/або блоки перетворення,

3) режим прогнозування, який сигналізує про те, який режим прогнозування необхідно використовувати для одержання сигналу прогнозування для кодувального блока 30, при цьому гранулярність, з якою цей режим прогнозування сигналізується, може залежати від кодувальних блоків 30 і/або прогнозувального блока.

4) параметри прогнозування можуть сигналізуватися на кодувальний блок або, якщо присутній, на прогнозувальний блок з типом надісланих параметрів прогнозування в залежності, наприклад, від режиму прогнозування. Можливі режими прогнозування можуть, наприклад, включати інтрапрогнозування і/або інтерпрогнозування,

5) можуть також бути присутніми інші синтаксичні елементи, такі як фільтрувальна інформація для фільтрування картинки 10 в кодувальному блоці 30 для одержання сигналу прогнозування і/або реконструйованого сигналу, який відтворюється,

6) нарешті, залишкова інформація у формі, між тим, коефіцієнтів перетворення може міститися у потоці даних для кодувального блока 30; залишкові дані можуть сигналізуватися в одиницях залишкових блоків; за допомогою залишкового блока спектральний розклад може, наприклад, виконуватися в одиницях вищезгаданих блоків перетворення, якщо вони присутні.

Ентропійний декодер 24 відповідає за одержання тільки що згаданих синтаксичних елементів з потоку даних. Для цього, ентропійний декодер 24 використовує адаптивне до контексту ентропійне декодування. Тобто, ентропійний декодер 24 надає декілька контекстів. Для одержання певного синтаксичного елемента з потоку даних 12, ентропійний декодер 24 вибирає певний контекст серед можливих контекстів. Вибір серед можливих контекстів виконується в залежності від атрибуту сусіднього блока частини картинки 10, якій належить поточний синтаксичний елемент. Для кожного з можливих контекстів ентропійний декодер 24 керує ймовірностями символів, тобто, оцінкою ймовірності для кожного можливого символу символного алфавіту, на основі якого працює ентропійний декодер 24. "Керування" включає вищезгадані безперервні оновлення ймовірностей символів контекстів для адаптації ймовірностей символів, зв'язаних кожним контекстом з реальним змістом картинки. Завдяки цьому заходу ймовірності символів адаптуються до реальної статистики ймовірності символів.

Іншою обставиною, де атрибути сусіднього блока впливають на відтворення поточної частини картинки 10, такої як поточний кодувальний блок 30, є прогнозувальне декодування в предикторі 28. Прогнозування обмежується не тільки змістом прогнозування в поточному кодувальному блоці 30, а може також охоплювати прогнозування параметрів, які містяться в потоці даних 12, для поточного

кодувального блока 30, таких як параметри прогнозування, дані розбиття або навіть коефіцієнти перетворення. Тобто, предиктор 28 може прогнозувати зміст картинки або такі параметри з вищезгаданого сусіднього блока для одержання записаного сигналу, який потім об'єднується із залишком прогнозування, як він одержаний модулем 26 від потоку даних 12. У випадку параметрів прогнозування, предиктор 28 може використовувати синтаксичні елементи, які містяться в потоці даних як залишки прогнозування для одержання реальної величини параметра прогнозування. Предиктор 28 використовує останнє значення параметра прогнозування для одержання тільки що згаданого сигналу прогнозування, який об'єднується із залишком прогнозування в об'єднувальному модулі 27.

Вищезгаданий "сусідній блок" головним чином покриває верхню ліву частину периметра поточної частини, якій належить синтаксичний елемент, який на даний момент ентропійно декодується, або синтаксичний елемент, який на даний момент прогнозується. На Фіг. 25 такий сусідній блок ілюстративно зображується в позиції 34 для одного кодувального блока 30.

Серед кодувальних блоків 30 визначається порядок кодування/декодування: на найкрупнішому рівні кодувальні деревоподібні блоки 32 картинки 10 проходяться в порядку 36 проходження (тут зображений як проходження растру, який рядок за рядком веде від верху до низу). В кожному кодувальному деревоподібному блоці кодувальні блоки 30 проходяться в глибину в першому порядку проходження так, що, на кожному ієрархічному рівні кодувальний деревоподібний блок 32 по суті проходиться також в порядку проходження растру, який веде рядок за рядком від верху до низу.

Порядок кодування, визначений серед кодувальних блоків 30, гармонізується з визначенням сусіднього блока 34, використовуваного для одержання атрибуту в сусідньому блоці для вибору контекстів і/або виконання просторового прогнозування в такому сусідньому блоці 34, в більшості випадків, покриває частини картинки 10, які вже були декодовані у відповідності з порядком кодування. Усяк випадок, коли частина сусіднього блока 34 покриває недоступні частини картинки 10, замість цього, використовуються дані за умовчанням. Наприклад, сусідній шаблон 34 може виходити за межі картинки 10. Однак, іншою можливістю є те, що сусідній блок 34 входить в сусідню вирізку.

Вирізки ділять, наприклад, картинку 10 в порядку кодування/декодування, визначеному серед кодувальних блоків 30, тобто, кожна вирізка є неперервною послідовністю кодувальних блоків 30 у вищезгаданому порядку кодувальних блоків. На Фіг. 25 вирізки вказані штрих-пунктирними лініями 14. Порядок, визначений серед вирізок 14, одержується з їх структури проходів послідовних кодувальних блоків 30, як окреслено вище. Якщо частина 18 синтаксичного елемента певної вирізки 14 вказує, що вона повинна декодуватися в першому режимі, то потім ентропійний декодер 24 дозволяє адаптивне до контексту ентропійне декодування для одержання контекстів крізь межі вирізки. Тобто, просторово сусідній блок 34 використовується для вибору контекстів в даних ентропійного декодування, які стосуються поточної вирізки 14. У випадку Фіг. 25, наприклад, вирізка 3 може бути поточно декодованою вирізкою і в ентропійному декодуванні синтаксичного елемента, який стосується кодувального блока 30 або деякої частини, яка міститься в ньому, ентропійний декодер 24 може використовувати атрибути, які одержуються з декодувальних частин в сусідній вирізці, такий як вирізка 1. Предиктор 28 поводить себе таким же чином: для вирізок, які належать першому режиму 20, предиктор 28 використовує просторове прогнозування крізь межу вирізки, яка охоплює поточну вирізку.

Однак, для вирізок, які мають другий режим 22, зв'язаний з ними, тобто, для якого частина 18 синтаксичного елемента вказує другий режим 22, ентропійний декодер 24 і предиктор 28 обмежують одержання ентропійних контекстів і прогнозувальне декодування залежністю від атрибутів, які належать до частин, які лежать тільки в поточній вирізці. Очевидно, ефективність кодування страждає від цього обмеження. З іншого боку, вирізки другого режиму 22 передбачають порушення внутрішніх залежностей між послідовністю вирізок. Відповідно, вирізки другого режиму 22 можуть розкидатися в картинці 10 або у відеоданих, яким належить картинка 10, для передбачення точок повторної синхронізації. Однак, не потрібно, щоб кожна картинка 10 мала принаймні одну вирізку в другому режимі 22.

Як вже згадувалося вище, перший і другий режим 20 і 22 також відрізняються у своїй ініціалізації ймовірностей символів. Вирізки, кодовані у другому режимі 22, надають ентропійний декодер 24, який повторно ініціалізує ймовірності незалежно від будь-якої попередньо декодованої вирізки, тобто, попередньо декодованої в сенсі порядку, визначеного серед вирізок. Ймовірності символів, наприклад, встановлюються рівними величинам за умовчанням, відомим як кодеру, так і декодеру, або величини ініціалізації містяться у вирізках, кодованих в другому режимі 22.

Тобто, для вирізок, які кодуються/декоднуються в другому режимі 22, адаптація ймовірностей символів завжди починається зразу ж на початку цих вирізок. Відповідно, точність адаптації є поганою для цих вирізок на початку цих вирізок.

Сутності відмінні у вирізках, кодованих/декодованих в першому режимі 20. Для останніх вирізок ініціалізація ймовірностей символів, виконувана ентропійним декодером 24, залежить від збережених

станів ймовірностей символів попередньо декодованої вирізки. Усяк випадок, коли вирізка, кодована/декодована в першому режимі 20, має свій початок, наприклад, розташований в іншому місці, ніж ліва сторона картини 10, тобто, не на стороні, звідки починається проходження 36 растру, яке здійснюється по рядку перед переходом на наступний рядок вниз, приймаються ймовірності символів, які одержуються в кінці ентропійного декодування безпосередньо попередньої вирізки. Це зображено, наприклад, на Фіг. 2 стрілкою 38 для вирізки 4. Вирізка 4 має свій початок десь між правою стороною і лівою стороною картини 10 і, відповідно, при ініціалізації ймовірностей символів, ентропійний декодер 24 приймає, при ініціалізації ймовірностей символів, ймовірності символів, які були одержані в ентропійному декодуванні безпосередньо попередньої вирізки, тобто вирізки 3, до її кінця, тобто включаючи безперервне оновлення ймовірностей символів під час ентропійного декодування вирізки 3 до її кінця.

Вирізки, які мають другий режим 22, зв'язаний з ними, однак, які мають початок на лівій стороні картини 10, такі як, наприклад, вирізка 5, не адаптують ймовірності символів, як вони одержані після закінчення ентропійного декодування безпосередньо попередньої вирізки 4, оскільки це повинно перешкоджати декодеру 5 паралельно декодувати картинку 10 шляхом використання хвильової обробки даних. Скоріше, як окреслено вище, ентропійний декодер 24 адаптує ймовірності символів, як вони одержані після завершення ентропійного декодування другого в порядку 36 кодування/декодування кодувального деревоподібного блоку 32 в безпосередньо попередньому в порядку 36 кодування/декодування рядку деревоподібних блоків, як вказано стрілкою 40.

На Фіг. 25, наприклад, картинка 10 була ілюстративно розбита на три рядки кодованих деревоподібних блоків і чотири стовпчики кодувальних деревоподібних кореневих блоків 32, і кожен рядок кодувальних деревоподібних блоків підрозбивався на дві вирізки 14 таким чином, що початок кожної другої вирізки співпадає з першою кодувальною одиницею в порядку кодувальних одиниць відповідного рядка кодувальних деревоподібних кореневих блоків. Ентропійний декодер 24 повинен, відповідно, бути здатним використовувати хвильову обробку даних в декодуванні картини 10 шляхом паралельного декодування кожного рядка кодувальних деревоподібних кореневих блоків з початком декодування цих рядків кодувальних деревоподібних кореневих блоків в шаховому порядку, починаючи з першого або найвищого рядка кодувальних деревоподібних кореневих блоків, потім другого, а потім третього.

Зазвичай, розбиття блоків 32 рекурсивним чином на додаткові кодувальні блоки 30 є необов'язковим і, відповідно, в більш загальному сенсі, блоки 32 можуть називатися також "кодувальними блоками". Тобто, загалом кажучи, картинка 10 може розбиватися на кодувальні блоки 32, які розташовані в рядках і стовпчиках, і мають порядок 36 проходження растру, визначений серед них, і декодер 5 може вважатися таким, що зв'язує кожну вирізку 14 з неперервною підмножиною кодувальних блоків 32 в порядку 36 проходження растру таким чином, що підмножини слідує одна за іншою згідно з порядком 36 проходження растру у відповідності з порядком вирізки.

Як також стало зрозуміло з вищенаведеного обговорення, декодер 5 або, точніше, ентропійний декодер 24 може конфігуруватися для зберігання ймовірностей символів, як вони одержані в адаптивному до контексту ентропійному декодуванні будь-якої вирізки до другого кодувального блоку в рядку кодувальних блоків у відповідності з порядком 36 проходження растру. При ініціалізації ймовірностей символів для адаптивного до контексту ентропійного декодування поточної вирізки, яка має перший режим 20, зв'язаний з нею, декодер 5 або, точніше, ентропійний декодер 24 перевіряє, чи є перший кодувальний блок 32 неперервної підмножини кодувальних блоків 32, зв'язаної з поточною вирізкою, першим кодувальним блоком 32 в рядку кодувальних блоків у відповідності з порядком 36 проходження растру. Якщо це так, то ймовірності символів для адаптивного до контексту ентропійного декодування поточної вирізки ініціалізуються, як пояснюється стосовно стрілки 40, зокрема в залежності від збережених ймовірностей символів, як вони одержані в адаптивному до контексту ентропійному декодуванні попередньо декодованої вирізки до другого кодувального блоку в рядку кодувальних блоків у відповідності з порядком 36 проходження растру. Якщо ні, то ініціалізація ймовірностей символів для адаптивного до контексту ентропійного декодування поточної вирізки виконується в залежності від ймовірностей символів, як вони одержані в адаптивному до контексту ентропійному декодуванні попередньо декодованої вирізки до кінця попередньо декодованої вирізки, тобто згідно зі стрілкою 38. Знову, у випадку ініціалізації згідно з 38, мається на увазі збережений стан в кінці ентропійного декодування безпосередньо попередньої вирізки в порядку 36 вирізки, тоді як, у випадку ініціалізації 40, це є попередньо декодована вирізка, яка містить кінець другого блоку безпосередньо попереднього рядка блоків 32 в порядку 36 блоків.

Як зображено пунктирними лініями на Фіг. 24, декодер може конфігуруватися для реагування на частину 18 синтаксичного елемента в поточній вирізці вирізок 14 для декодування поточної вирізки у відповідності з одним з принаймні трьох режимів. Тобто, окрім інших режимів 20 і 22 може бути третій режим 42. Третій режим 42 може відрізнятися від другого режиму 22 тим, що допускається прогнозування крізь межі вирізки, тоді як ентропійне кодування/декодування все ще обмежується для не перетинання меж вирізки.

Вище були представлені два варіанти виконання, які стосуються частини 18 синтаксичного елемента. Нижче таблиця наводить ці два варіанти виконання.

	Варіант виконання 1	Варіант виконання 2
Частина синтаксичного елемента	dependent_slice_flag, no_cabac_reset_flag	dependent_slice_flag
Режим 1	dependent_slice_flag=1, no_cabac_reset_flag=1	dependent_slice_flag=1,
Режим 2	dependent_slice_flag=0	dependent_slice_flag=0
Режим 3	dependent_slice_flag=1, no_cabac_reset_flag=0	dependent_slice_flag=1, cabac_independent_flag=1, tiles_or_entropy_coding_sync_idc=3

В одному варіанті виконання частина 18 синтаксичного елемента окремо формується dependent_slice_flag, тоді як в іншому варіанті виконання комбінація dependent_slice_flag і no_cabac_reset_flag формує частину синтаксичного елемента. Посилання робиться на процес синхронізації для змінних контексту настільки, наскільки це стосується ініціалізації ймовірностей символів в залежності від збережених станів ймовірностей символів попередньо декодованої вирізки. Зокрема, якщо last_ctb_cabac_init_flag=0 і tiles_or_entropy_coding_sync_idc=2, то декодер може конфігуруватися для збереження ймовірностей символів, як вони одержані в адаптивному до контексту ентропійному декодуванні попередньо декодованої вирізки до другого кодувального блока в рядку у відповідності з порядком проходження растру, і, при ініціалізації ймовірностей символів для адаптивного до контексту ентропійного декодування поточної вирізки у відповідності з першим режимом, перевіряють, чи є перший кодувальний блок неперервної підмножини кодувальних блоків, зв'язаної з поточною вирізкою, першим кодувальним блоком в рядку у відповідності з порядком проходження растру і, якщо це так, то ініціалізують ймовірності символів для адаптивного до контексту ентропійного декодування поточної вирізки в залежності від збережених ймовірностей символів, як вони одержані в адаптивному до контексту ентропійному декодуванні попередньо декодованої вирізки до другого кодувального блока в рядку у відповідності з порядком проходження растру, і, якщо ні, то ініціалізують ймовірності символів для адаптивного до контексту ентропійного декодування поточної вирізки в залежності від ймовірностей символів, як вони одержані в адаптивному до контексту ентропійному декодуванні попередньо декодованої вирізки до кінця попередньо декодованої вирізки.

Таким чином, іншими словами, у відповідності з другим варіантом виконання, для синтаксису декодер повинен відтворювати картинку 10 з потоку даних 12, у який картинка кодується в одиницях вирізки 14, на які розбивається картинка (10), при цьому декодер конфігурується для декодування вирізок 14 з потоку даних 12 у відповідності з порядком 16 вирізки і декодер реагує на частину 18 синтаксичного елемента, зокрема dependent_slice_flag, в поточній вирізці вирізку для декодування поточної вирізки у відповідності з одним з принаймні двох режимів 20, 22. У відповідності з першим 20 з принаймні двох режимів, зокрема, якщо dependent_slice_flag=1, декодер декодує поточну вирізку з потоку даних 12 з використанням адаптивного до контексту ентропійного декодування 24, яке включає одержання контекстів крізь межі вирізки, безперервне оновлення ймовірностей символів контекстів і ініціалізацію 38, 40 ймовірностей символів в залежності від збережених станів ймовірностей символів попередньо декодованої вирізки і прогнозувальне декодування крізь межі вирізки, і у відповідності з другим 22 з принаймні двох режимів, зокрема, якщо dependent_slice_flag=0, декодер декодує поточну вирізку з потоку даних 12 з використанням адаптивного до контексту ентропійного декодування з обмеженням одержання контекстів для не перетинання меж вирізки, безперервного оновлення ймовірностей символів контекстів і ініціалізації ймовірностей символів незалежно від будь-якої попередньо декодованої вирізки, і прогнозувального декодування з обмеженням прогнозувального декодування для не перетинання меж вирізки. Картинка 10 може розбиватися на кодувальні блоки 32, які розташовані в рядках і стовпчиках, і мають порядок 36 проходження растру, визначений серед них, і декодер конфігурується для зв'язування кожної вирізки 14 з неперервною підмножиною кодувальних блоків 32 в порядку 36 проходження растру таким чином, що підмножини слідує одна за іншою згідно з порядком 36 проходження растру у відповідності з порядком вирізки. Декодер, зокрема, який реагує на tiles_or_entropy_coding_sync_idc=2, може конфігуруватися для збереження ймовірностей символів, як вони одержані в адаптивному до контексту ентропійному декодуванні попередньо декодованої вирізки до другого кодувального блока 32 в рядку у відповідності з порядком 36 проходження растру, і, при ініціалізації ймовірностей символів для адаптивного до контексту ентропійного декодування поточної вирізки у відповідності з першим режимом, перевіряють, чи є перший кодувальний блок неперервної підмножини кодувальних блоків 32, зв'язаної з поточною вирізкою, першим кодувальним блоком 32 в рядку у відповідності з порядком проходження растру, і, якщо так, то ініціалізують 40 ймовірності символів для адаптивного до контексту ентропійного

декодування поточної вирізки в залежності від збережених ймовірностей символів, як вони одержані в адаптивному до контексту ентропійному декодуванні попередньо декодованої вирізки до другого кодувального блока в рядку у відповідності з порядком 36 проходження растру, і, якщо ні, то ініціалізують 38 ймовірностей символів для адаптивного до контексту ентропійного декодування поточної вирізки в залежності від ймовірностей символів, як вони одержані в адаптивному до контексту ентропійному декодуванні попередньо декодованої вирізки до кінця попередньо декодованої вирізки. Декодер може конфігуруватися для реагування на частину (18) синтаксичного елемента в поточній вирізці вирізок 14 для декодування поточної вирізки у відповідності з одним з принаймні трьох режимів, зокрема в одному режимі, вибраному серед першого 20 і третього режиму 42 або другого режиму 22, при цьому у відповідності з третім режимом 42, зокрема, якщо `dependent_slice_flag=1` і `tiles_or_entropy_coding_sync_idc=3`, декодер сконфігурований для декодування поточної вирізки з потоку даних з використанням адаптивного до контексту ентропійного декодування з обмеженням одержання контекстів для не перетинання меж вирізки, неперервного оновлення ймовірностей символів контекстів і ініціалізації ймовірностей символів незалежно від будь-якої попередньо декодованої вирізки і прогнозувального декодування крізь межі вирізки, при цьому один режим, вибраний серед першого і третього режиму, вибирається в залежності від синтаксичного елемента, зокрема `cabac_independent_flag`. Декодер може додатково конфігуруватися, зокрема, якщо `tiles_or_entropy_coding_sync_idc=0,1`, і 3 ("3" коли `cabac_independent_flag=0`), для збереження ймовірностей символів, як вони одержані в адаптивному до контексту ентропійному декодуванні попередньо декодованої вирізки до кінця попередньо декодованої вирізки, і, при ініціалізації ймовірностей символів для адаптивного до контексту ентропійного декодування поточної вирізки у відповідності з першим режимом, для ініціалізації ймовірностей символів для адаптивного до контексту ентропійного декодування поточної вирізки в залежності від збережених ймовірностей символів. Декодер може конфігуруватися, зокрема, якщо `tiles_or_entropy_coding_sync_idc=1` в першому і другому режимі, для обмеження прогнозувального декодування в мозаїках, на які підрозбивається картинка.

Зазвичай, кодер здатен встановлювати вищенаведений синтаксис, відповідно, для надання декодеру можливості отримувати вищевказані переваги. Кодер може паралельно обробляти дані, як багатоядерний кодер, але не потребує бути їм. Для кодування картинки 10 з одержанням потоку даних 12 в одиницях вирізок 14, кодер повинен конфігуруватися для кодування вирізок 14 з одержанням потоку даних 12 у відповідності з порядком 16 вирізки. Кодер повинен визначати частину 18 синтаксичного елемента для кодування її з одержанням поточної вирізки вирізок таким чином, що частина синтаксичного елемента сигналізує поточну вирізку, яка кодується у відповідності з одним з принаймні двох режимів 20, 22, і якщо поточна вирізка повинна кодуватися у відповідності з першим 20 з принаймні двох режимів, то кодують поточну вирізку з одержанням потоку даних 12 з використанням адаптивного до контексту ентропійного кодування 24, яке включає одержання контекстів крізь межі вирізки, безперервне оновлення ймовірностей символів контекстів і ініціалізацію 38, 40 ймовірностей символів в залежності від збережених станів ймовірностей символів попередньо кодованої вирізки і прогнозувальне кодування крізь межі вирізки, і якщо поточна вирізка повинна кодуватися у відповідності з другим 22 з принаймні двох режимів, то кодують поточну вирізку з одержанням потоку даних 12 з використанням адаптивного до контексту ентропійного кодування з обмеженням одержання контекстів для не перетинання меж вирізки, безперервного оновлення ймовірностей символів контекстів і ініціалізації ймовірностей символів незалежно від будь-якої попередньо кодованої вирізки, і прогнозувального кодування з обмеженням прогнозувального кодування для не перетинання меж вирізки. Хоча картинка 10 може розбиватися на кодувальні блоки 32, які розташовані в рядках і стовпчиках, і мають порядок 36 проходження растру, визначений серед них, кодер може конфігуруватися для зв'язування кожної вирізки 14 з неперервною підмножиною кодувальних блоків 32 в порядку 36 проходження растру таким чином, що підмножини слідує одна за іншою згідно з порядком 36 проходження растру у відповідності з порядком вирізки. Кодер може конфігуруватися для збереження ймовірностей символів, як вони одержані в адаптивному до контексту ентропійному кодуванні попередньо кодованої вирізки до другого кодувального блока 32 в рядку у відповідності з порядком 36 проходження растру, і, при ініціалізації ймовірностей символів для адаптивного до контексту ентропійного кодування поточної вирізки у відповідності з першим режимом, для перевірки, чи є перший кодувальний блок неперервної підмножини кодувальних блоків 32, зв'язаної з поточною вирізкою, першим кодувальним блоком 32 в рядку у відповідності з порядком проходження растру і, якщо так, то для ініціалізації 40 ймовірностей символів для адаптивного до контексту ентропійного кодування поточної вирізки в залежності від збережених ймовірностей символів, як вони одержані в адаптивному до контексту ентропійному кодуванні попередньо кодованої вирізки до другого кодувального блока в рядку у відповідності з порядком 36 проходження растру, і, якщо ні, то для ініціалізації 38 ймовірностей символів для адаптивного до контексту ентропійного кодування поточної вирізки в залежності від ймовірностей символів, як вони одержані в адаптивному до контексту ентропійному кодуванні попередньо декодованої вирізки до кінця попередньо кодованої

вирізки. Кодер може конфігуруватися для кодування частини (18) синтаксичного елемента з одержанням поточної вирізки вирізок (14) таким чином, що поточна вирізка сигналізується для кодування з одержанням неї у відповідності з одним з принаймні трьох режимів, зокрема в одному з режимів, вибраного серед першого (20) і третього режиму (42) або другого режиму (22), при цьому кодер конфігурується у відповідності з третім режимом (42) для кодування поточної вирізки з одержанням потоку даних з використанням адаптивного до контексту ентропійного кодування з обмеженням одержання контекстів для не перетинання меж вирізки, безперервного оновлення ймовірностей символів контекстів і ініціалізації ймовірностей символів незалежно від будь-якої попередньо кодованої вирізки, і прогнозувального кодування крізь межі вирізки, при цьому кодер розрізняє один режим, вибраний серед першого і третього режиму, з використанням синтаксичного елемента, зокрема, наприклад, `cabac_independent_flag`. Кодер може конфігуруватися для визначення загального синтаксичного елемента, такого як `dependent_slices_present_flag`, і записувати його в потік даних, працюючи в одному з принаймні двох загальних робочих режимів в залежності від загального синтаксичного елемента, зокрема згідно з першим загальним робочим режимом, виконуючи кодування частини синтаксичного елемента для кожної вирізки, і згідно з другим загальним робочим режимом, неухильно використовуючи інший режим, вибраний серед принаймні двох режимів, відмінний від першого режиму. Кодер може конфігуруватися згідно з першим і другим режимом для неухильного і безперервного продовження оновлення ймовірностей символів від початку до кінця поточної вирізки. Кодер може конфігуруватися для збереження ймовірностей символів, як вони одержані в адаптивному до контексту ентропійному кодуванні попередньо кодованої вирізки до кінця попередньо кодованої вирізки, і, при ініціалізації ймовірностей символів для адаптивного до контексту ентропійного кодування поточної вирізки у відповідності з першим режимом, для ініціалізації ймовірності символів для адаптивного до контексту ентропійного кодування поточної вирізки в залежності від збережених ймовірностей символів. І кодер може в першому і другому режимі обмежувати прогнозувальне кодування в мозаїках, на які підрозбивається картинка.

Можлива структура кодера зображена на Фіг. 26 для повноти. Предиктор 70 працює майже так само як і предиктор 28, тобто, виконує прогнозування, і також визначає шляхом оптимізації, наприклад, параметри кодування, які включають параметри прогнозування і режими. Модулі 26 і 27 також присутні в декодері. Модуль віднімання 72 визначає залишок прогнозування без втрат, який потім з втратами шляхом використання дискретизації і, необов'язково, з використанням перетворення спектрального розкладу, кодується в модулі 74 перетворення і дискретизації. Ентропійний кодер 76 виконує адаптивне до контексту ентропійне кодування.

На додаток до вищенаведених конкретних прикладів синтаксису, інший приклад наводиться нижче із забезпеченням узгодження між термінами, використовуваними далі, і термінами, використовуваними вище.

Зокрема, без особливого зазначення вище, залежні вирізки є не тільки "залежними" в тому, що вони передбачають використання інформації, відомої з ділянок зовні їх межі, як описано вище, а мають швидше адаптовані ентропійні контексти або досягають кращого просторового прогнозування завдяки змозі перетину їх межі. Скоріше, для збереження темпу, який повинен використовуватися для визначення заголовків вирізок шляхом поділу картинки на вирізки, залежні вирізки приймають частину синтаксису заголовка вирізки з попередніх вирізок, тобто, ця частина заголовку синтаксису вирізки не передається повторно для залежних вирізок. Це, наприклад, зображено на Фіг. 16 в позиції 100 і на Фіг. 21 в позиції 102, згідно з чим тип вирізки, наприклад, приймається від попередньої вирізки. Завдяки цьому заходу підрозбиття картинок на вирізки, такі як незалежна вирізка і залежні вирізки, є менш дорогими з точки зору використання бітів.

Саме тільки що згадана залежність приводить у нижченаведеному прикладі до трохи іншого формулювання: вирізки визначаються як одиничні частини картинки, у яких здатен встановлюватися синтаксис заголовка вирізки. Відповідно, вирізки складаються з однієї - використовуючи вищенаведену номенклатуру - незалежної/стандартної/нормальної вирізки, тепер названої сегментом незалежної вирізки, і однієї або більшої кількості - використовуючи вищенаведену номенклатуру - залежних вирізок, тепер названих сегментами залежної вирізки.

Фіг. 27, наприклад, зображає картинку, яка розбивається на дві вирізки, одна з яких сформована сегментами 14₁-14₃, а інша сформована виключно сегментом 14₄. Індeksi 1-4 зображають порядок вирізки в порядку кодування. Фіг. 28a і b зображають інший приклад у випадку підрозбиття картинки 10 на дві мозаїки, при цьому у випадку Фіг. 28a, одна вирізка, сформована усіма п'ятьма сегментами 14, які покривають обидві мозаїки 50₁ і 50₂ - індeksi, які повторно ростуть в порядку кодування - і, у випадку з Фіг. 28a, дві вирізки, сформовані сегментами 14₁ і 14₂ та 14₃ і 14₄, відповідно, які підрозбивають мозаїку 50₁, та іншою вирізкою, сформованою сегментами 14₅-14₆, які покривають мозаїку 50₂.

Визначення можуть бути наступними:

сегмент залежної вирізки: Сегмент вирізки, для якого величини деяких синтаксичних елементів заголовка сегмента вирізки одержуються з величин для сегмента попередньої незалежної вирізки в

порядку декодування: раніше, у вищенаведених варіантах виконання названа залежною вирізкою.

сегмент незалежної вирізки: Сегмент вирізки, для якого величини синтаксичних елементів заголовка сегмента вирізки не одержується з величин для сегмента попередньої вирізки: раніше, у вищенаведених варіантах виконання названої нормальною вирізкою.

вирізка: Ціла кількість кодувальних деревоподібних одиниць, які містяться в одному сегменті незалежної вирізки і усіх наступних сегментах залежної вирізки (якщо такі є), які передують наступному сегменту незалежної вирізки (якщо така є) в тій же одиниці доступу/картинці.

заголовок вирізки: Заголовок сегмента вирізки сегмента незалежної вирізки, який є сегментом поточної вирізки або є сегментом незалежної вирізки, яка передує сегменту поточної залежної вирізки.

сегмент вирізки: Ціла кількість кодувальних деревоподібних блоків, які розташовані послідовно в порядку проходженні мозаїки і містяться в єдиному блоці NAL; поділ кожної картинки на сегменти вирізки є розбиттям.

заголовок сегмента вирізки: частина сегмента кодуваної вирізки, яка містить елементи даних, які належать до першої або усіх кодувальних деревоподібних одиниць, представлених в сегменті вирізки.

Сигналізація "режимів" 20 і 22, тобто "сегмента залежної вирізки" і "сегмента незалежної вирізки" може бути наступною:

В деяких екстраблоках NAL, таких як PPS, синтаксичний елемент може використовуватися для сигналізації того, чи використовуються, чи ні залежні вирізки для певної картинки однорідної послідовності для певних картинок:

`dependent_slice_segments_enabled_flag` equal to 1 вказує присутність синтаксичного елемента `dependent_slice_segment_flag` в заголовках сегментів вирізки. `dependent_slice_segments_enabled_flag` equal to 0 вказує відсутність синтаксичного елемента `dependent_slice_segment_flag` в заголовках сегментів вирізки.

`dependent_slice_segments_enabled_flag` подібний по об'єму до раніше описаного `dependent_slices_present_flag`

Подібним чином, `dependent_slice_flag` може називатися `dependent_slice_segment_flag` для взяття до уваги іншої номенклатури стосовно вирізок.

`dependent_slice_segment_flag` equal to 1 вказує, що величина синтаксичного елемента заголовка сегмента кожної вирізки, який відсутній в заголовку сегмента поточної вирізки, дорівнює величині синтаксичного елемента заголовка сегмента відповідної вирізки в заголовці вирізки, тобто заголовка сегмента попередньої незалежної вирізки.

На тому ж рівні, такому як рівень картинки, може включатися наступний синтаксичний елемент:

`entropy_coding_sync_enabled_flag` equal to 1 вказує, що викликається спеціальний процес сигналізації для змінних контексту перед декодуванням кодувальної деревоподібної одиниці, яка містить перший кодувальний деревоподібний блок рядка кодувальних деревоподібних блоків в кожній мозаїці в кожній картинці, яка посилається на PPS, і спеціальний процес зберігання для змінних контексту викликається після декодування кодувальної деревоподібної одиниці, яка містить другий кодувальний деревоподібний блок рядка кодувальних деревоподібних блоків в кожній мозаїці в кожній картинці, яка посилається на PPS. `entropy_coding_sync_enabled_flag` equal to 0 вказує, що не вимагається викликання спеціального процесу синхронізації для змінних контексту перед декодуванням кодувальної деревоподібної одиниці, яка містить перший кодувальний деревоподібний блок рядка кодувальних деревоподібних блоків в кожній мозаїці в кожній картинці, яка посилається на PPS, і не вимагається викликання спеціального процесу зберігання для змінних контексту після декодування кодувальної деревоподібної одиниці, яка містить другий кодувальний деревоподібний блок рядка кодувальних деревоподібних блоків в кожній мозаїці в кожній картинці, яка посилається на PPS.

Вимога узгодженості бітового потоку полягає в тому, що величина `entropy_coding_sync_enabled_flag` повинна бути тією же для усіх PPSs, які активуються в CVS.

Коли `entropy_coding_sync_enabled_flag` дорівнює 1 і перший кодувальний деревоподібний блок у вирізці не є першим кодувальним деревоподібним блоком рядка кодувальних деревоподібних блоків в мозаїці, то вимога узгодженості бітового потоку полягає в тому, що останній кодувальний деревоподібний блок у вирізці повинен належати тому ж рядку кодувальних деревоподібних блоків, що й перший кодувальний деревоподібний блок у вирізці. Коли `entropy_coding_sync_enabled_flag` дорівнює 1 і перший кодувальний деревоподібний блок в сегменті вирізки не є першим кодувальним деревоподібним блоком рядка кодувальних деревоподібних блоків в мозаїці, то вимога узгодженості бітового потоку полягає в тому, що останній кодувальний деревоподібний блок в сегменті вирізки повинен належати до того ж рядка кодувальних деревоподібних блоків, що й перший кодувальний деревоподібний блок в сегменті вирізки.

Як вже описано, порядок кодування/декодування серед CTBs 30 здійснюється в растровому порядку по рядкам від верхньої частини до нижньої, починаючи з проходження першої мозаїки з переходом потім до наступної мозаїки, якщо в картинці присутня більше, ніж одна мозаїка.

Декодер 5 і, відповідно, кодер функціонує наступним чином в ентропійному декодуванні

(кодуванні) сегментів вирізки 14 картинки:

A1) Усяк випадок, коли поточно декодований/кодований синтаксичний елемент `synEI` є першим синтаксичним елементом мозаїки 50, сегментом 14 вирізки або рядком СТВ, розпочинається процес ініціалізації з Фіг. 29.

A2) Інакше, відбувається декодування цього синтаксичного елемента з використанням поточних ентропійних контекстів.

A3) Якщо поточний синтаксичний елемент був останнім синтаксичним елементом в СТВ 30, то потім розпочинається процес збереження ентропійного контексту, як зображено на Фіг. 30.

A4) Процес передувє етапу A1) з наступним синтаксичним елементом.

В процесі ініціалізації перевіряється 200, чи є `synEI` першим синтаксичним елементом сегмента вирізки 14 або мозаїки 50. Якщо так, то контексти ініціалізуються незалежно від будь-якого сегмента попередньої вирізки на етапі 202. Якщо ні, то перевіряють 204, чи є `synEI` першим синтаксичним елементом рядка СТБ 30 і `entropy_coding_sync_enabled_flag` дорівнює одиниці. Якщо так, то перевіряють 206, чи у попередньому рядку СТБ 30 однакової мозаїки доступний другий СТВ 30 (дивіться Фіг. 23). Якщо так, то адаптація контексту згідно з 40 виконується на етапі 210 з використанням поточно збережених ймовірностей контексту для адаптацій типу 40. Якщо ні, то контексти ініціалізуються незалежно від будь-якого сегмента попередньої вирізки на етапі 202. Якщо перевірка 204 вказує ні, то потім перевіряють на етапі 212, чи `synEI` є першим синтаксичним елементом в першому СТВ сегмента залежної вирізки 14, і чи `dependent_slice_segement_flag` дорівнює одиниці, і якщо так, то приймання контексту згідно з 38 виконують на етапі 214 з використанням поточно збережених ймовірностей контексту для адаптацій типу 38. Після будь-яких з етапів 214, 212, 210 і 202, реально починається декодування/кодування.

Сегменти залежної вирізки з `dependent_slice_segement_flag`, який дорівнює одиниці, таким чином, допомагають додатково зменшувати затримку кодування/декодування майже без погіршення ефективності кодування.

В процесі збереження з Фіг. 30 на етапі 300 перевіряють, чи кодований/декодований `synEI` є останнім синтаксичним елементом другого СТВ 30 рядка СТБ 30, і `entropy_coding_sync_enabled_flag` дорівнює одиниці. Якщо так, то поточні ентропійні контексти на етапі 302, тобто, ймовірності ентропійного кодування контекстів, зберігаються в запам'ятовуючому пристрої, який є спеціальним для процесів прийняття величин згідно з 40. Подібним чином, на етапі 304 перевіряють, на додаток до етапу 300 або 302, чи кодований/декодований `synEI` є останнім синтаксичним елементом сегмента 14 вирізки, і `dependent_slice_segement_flag` дорівнює одиниці. Якщо так, то поточні ентропійні контексти на етапі 306, тобто, ймовірності ентропійного кодування контекстів, зберігають у запам'ятовуючому пристрої, який є спеціальним для процесів прийняття величин згідно з 38.

Відзначається, що будь-яка перевірка того, чи є синтаксичний елемент першим `synEI` рядка СТВ, використовує, наприклад, синтаксичний елемент `slice_adress` 400 в заголовках сегментів вирізки, тобто, початковий синтаксичний елемент, який вказує положення початку відповідного сегмента вирізки згідно з порядком декодування.

При відтворенні картинки 10 з потоку даних 12 з використанням WPP, декодер здатен використовувати саме останню частину 400 початку синтаксису для пошуку точок входу WPP вкладеного потоку. Оскільки кожен сегмент вирізки містить початкову частину 400 синтаксису, яка вказує положення початку декодування відповідного сегмента вирізки в картинці 10, то декодер здатне ідентифікувати точки входу WPP вкладених потоків, у які групуються сегменти вирізки, шляхом ідентифікації з використанням початкових частин 400 синтаксису сегментів вирізки, сегментів вирізки, які починаються на лівій стороні картинки. Декодер може потім паралельно декодувати WPP вкладені потоки в шаховому порядку з послідовним початком декодування WPP вкладених потоків у відповідності з порядком вирізки. Сегменти вирізки можуть навіть бути меншими за ширину однієї картинки, тобто за один рядок СТБ, таким чином, що їх передача може перемежовуватися серед WPP вкладених потоків для додаткового зменшення загальної наскрізної затримки передачі даних. Кодер забезпечує кожен вирізку (14) початковою частиною (400) синтаксису, яка вказує положення початку кодування відповідної вирізки в картинці (10) і повинен групувати вирізки у WPP вкладені потоки таким чином, що для кожного WPP вкладеного потоку перша вирізка згідно з порядком вирізки починається на лівій стороні картинки. Кодер може навіть сам по собі використовувати WPP у кодуванні картинки: кодер паралельно кодує WPP вкладені потоки в шаховому порядку з послідовним початком кодування WPP вкладених потоків у відповідності з порядком вирізки.

Між іншим, останній аспект використання початкових частин синтаксису сегмента вирізки як засобу для визначення координат точок входу WPP вкладеного потоку може використовуватися без концепції залежної вирізки.

Судячи з усього, повинна бути можливою паралельна обробка картинки 10 шляхом встановлення вищезгаданих змінних наступним чином:

	tiles_enabled_flag==0	tiles_enabled_flag==1	entropy_coding_sync_enable_flag==1	tiles_enabled_flag==1	entropy_coding_sync_enable_flag==1
dependent_slice_segment_flag	0/1	0/1	0/1	0/1	0/1
Підрозбиття картини на мозаїки	Hi	Так	Hi	Так	Hi
Підрозбиття вирізки з точками входу для вказання початку WPP вкладеного потоку або мозаїки.	Hi	Так	Так	Hi	Hi
Можлива паралельна обробка даних	у вирізках	в мозаїках	WPP	в мозаїках	WPP

Повинно навіть бути можливим поєднувати WPP з розбиттям мозаїки. У такому випадку, можна обробляти мозаїки як окремі картини: кожна з яких, яка використовує WPP, повинна складатися з вирізки, яка має один або більшу кількість сегментів залежної вирізки, і перевірка на етапі 300 і 208 повинна посилатися на другий СТВ у верхньому рядку СТВ в тій же вирізці, так само як етапи 204 і A1 повинні посилатися на перший СТВ 30 в рядку СТВ 30 поточної мозаїки. У такому випадку, вищенаведена таблиця може розширюватися:

	tiles_enabled_flag	tiles_enabled_flag==1	entropy_coding_sync_enable_flag==1	tiles_enabled_flag==1	entropy_coding_sync_enable_flag==1	entropy_coding_sync_enable_flag==1	entropy_coding_sync_enable_flag==1
dependent_slice_segment_flag	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Підрозбиття картини на мозаїки	Hi	Так	Hi	Так	Hi	Так	Так
Підрозбиття вирізки з точками входу для вказання початку WPP вкладеного потоку або мозаїки.	Hi	Так	Так	Hi	Hi	Hi (кожен вкладений потік мозаїки у його власній залежній вирізці)	Так (кожна мозаїка в окремій вирізці і точки входу для WPP вкладених потоків у вирізці мозаїки)
можлива паралельна обробка даних	у вирізках	в мозаїках	WPP	в мозаїках	WPP	WPP вкладених потоків в мозаїках	WPP вкладених потоків в мозаїках

Як коротка примітка, останнє подовження повинно було також бути можливим з варіантом виконання 2. Варіант виконання 2 дозволяє наступну обробку:

	Рівень синтаксису				
tiles_or_entropy_coding_sync_idc	на картинку	0	1	2	3
cabac_independent_flag	на вирізку	відсутнє	відсутнє	відсутнє	0/1
dependent_slice_flag	на вирізку	0/1	0/1	0/1	0/1
Підрозбиття картини на мозаїки		Hi	Так	Hi	Hi
Підрозбиття вирізки з точками входу для вказання початку WPP вкладеного потоку або мозаїки		Hi	Так	Так	Hi
Можлива паралельна обробка		У вирізках	в мозаїках	WPP	Ентропійне

даних						кодування вирізків
-------	--	--	--	--	--	-----------------------

Але з наступними подовженнями таблиця нижче повинна надавати:

Додавання до семантики множини значень параметра картинки:

Якщо `tiles_or_entropy_coding_sync_idc` дорівнює 4, то перший рядок CTBs повинен міститися в іншій вирізці із прапорцем залежної вирізки, встановленим в 1. CTBs різних рядків не повинні бути присутніми в одній і тій же вирізці. Може бути більше, ніж одна вирізка, присутня на рядок CTB.

Якщо `tiles_or_entropy_coding_sync_idc` дорівнює 5, то CTBs першої мозаїки повинні міститися в іншій вирізці. CTBs різних мозаїк не повинні бути присутніми в одній і тій же вирізці. Може бути більше ніж одна вирізка, присутня на мозаїку.

Дивіться Фіг. 31 для подальшого пояснення.

Тобто, вищенаведена таблиця може подовжуватися:

	Рівень синтаксису								
<code>tiles_or_entropy_coding_sync_idc</code>	на картинку	0	1	2	3	5	4	6	7
<code>cabac_independent_flag</code>	На вирізку	відсутнє	відсутнє	відсутнє	0/1	відсутнє	відсутнє	відсутнє	відсутнє
<code>dependent_slice_flag</code>	На вирізку	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Підрозбиття картини на мозаїки		Ні	Так	Ні	Ні	Так	Ні	Так	Так
Підрозбиття вирізки з точками входу для вказання початку WPP вложеного потоку або мозаїки.		Ні	Так	Так	Ні	Ні	Ні	Ні (кожен вкладений потік мозаїки у його власній залежній вирізці)	Так (кожна мозаїка в окремі й вирізці і точки входу для WPP вкладених потоків у вирізці мозаїки)
Можлива паралельна обробка		у вирізках	в мозаїках	WPP	Ентропійне кодування вирізків	в мозаїках	WPP	WPP вкладені потоки в мозаїках	WPP вкладені потоки в мозаїках

Що стосується вищевказаних варіантів виконання, слід відзначити, що декодер може конфігуруватися для, наприклад, реагування на `tiles_or_entropy_coding_sync_idc=1,2` у першому і другому режимі, зчитувати інформацію з поточної вирізки, яка вказує підрозбиття поточної вирізки на паралельні підсекції, при цьому паралельні підсекції можуть бути WPP вкладеними потоками або мозаїками, припиняти адаптивне до контексту ентропійне декодування в кінці першої паралельної підсекції і відновлювати адаптивне до контексту ентропійне декодування на початку будь-якої наступної паралельної підсекції, яке включає в першому режимі ініціалізацію ймовірностей символів в залежності від збережених станів ймовірностей символів попередньої паралельної підсекції і, в другому режимі, ініціалізацію ймовірностей символів незалежно від будь-якої попередньо декодованої вирізки і будь-якої попередньо декодованої паралельної підсекції.

Таким чином, вищевказаний опис розкрив способи кодування з малою затримкою, декодування, інкапсуляції і передачі структурованих відеоданих, як вони надаються новим стандартом кодування HEVC, таких як структуровані в мозаїках дані, вкладені потоки паралельної хвильової обробки даних (WPP), вирізки або ентропійні вирізки.

Особливо, було визначено як транспортувати паралельно кодовані дані в сценарії діалогу для досягання мінімальної латентності в процесі кодування, декодування і передачі даних. Тому, було описано паралельне кодування каналу інформації, передача даних і наближення декодування для забезпечення мінімальної затримки у застосуваннях, таких як ігри, дистанційна хірургія і так далі.

Окрім того, вищевказані варіанти виконання заповнили пробіл паралельної хвильової обробки даних (WPP) для можливості її застосування у передачах даних з малою затримкою. Тому, був представлений новий формат інкапсуляції для WPP вкладених потоків даних 0: залежна вирізка. Ця залежна вирізка може містити дані ентропійної вирізки, WPP вкладений потік, увесь рядок LCUs, тільки фрагмент вирізки, де заголовок попередньо переданої вирізки також застосовується до вміщених даних фрагмента. Вміщені дані сигналізуються в заголовку субвирізки.

Нарешті, відзначається, що назвою для нових вирізок може також бути "Підмножина /Вирізки малої ваги", але назва "Залежна Вирізка" вважається кращою.

Була представлена сигналізація, яка описує рівень розпаралелювання кодування і транспортування.

Хоча деякі аспекти були описані в контексті пристрою, зрозуміло, що ці аспекти також представляють опис відповідного способу, де блок або пристрій відповідає етапу способу або ознаці етапу способу. Аналогічно, аспекти, описані в контексті етапу способу, також представляють опис відповідного блока або елемента, або ознаки відповідного пристрою. Деякі або усі етапи способу можуть виконуватися (або використовувати) апаратним засобом, таким як, наприклад, мікропроцесор, програмований комп'ютер або електронний контур. В деяких варіантах виконання деякий один або більша кількість найважливіших етапів способу можуть виконуватися таким пристроєм.

В залежності від певних вимог до втілення, варіанти виконання винаходу можуть реалізовуватися в апаратному засобі або в програмному засобі. Реалізація може здійснюватися з використанням середовища для зберігання цифрової інформації, наприклад дискети, DVD, Blu-Ray, CD, ROM, PROM, EPROM, EEPROM або флеш-пам'яті, яке має збережені на собі здатні до електронного читання керувальні сигнали, які взаємодіють (або здатні взаємодіяти) з програмованою комп'ютерною системою так, що виконується відповідний спосіб. Тому, середовище для збереження цифрової інформації може читатися комп'ютером.

Деякі варіанти виконання згідно з винаходом містять носій даних, який має здатні до читання електронними засобами керувальні сигнали, які здатні взаємодіяти з програмованою комп'ютерною системою так, що виконується один з описаних тут способів.

Загалом, варіанти виконання представленого винаходу можуть втілюватися як комп'ютерна програма з програмним кодом, який виконується для виконання одного із способів, коли комп'ютерна програма виконується на комп'ютері. Програмний код може, наприклад, зберігатися на здатному до читання машиною носії.

Інші варіанти виконання містять комп'ютерну програму для виконання одного з описаних тут способів, збережену на здатному до читання машиною носії.

Іншими словами, варіант виконання винаходу є, тому, комп'ютерною програмою, яка має програмний код для виконання одного з описаних тут способів, коли комп'ютерна програма виконується на комп'ютері.

Подальший варіант виконання винаходу є, тому, носієм даних (або середовищем для зберігання цифрової інформації або здатним до читання комп'ютером середовищем), яке містить записану на ньому комп'ютерну програму для виконання одного з описаних тут способів. Носій даних, середовище для зберігання цифрової інформації або середовище для запису типowo є матеріальними і/або неперехідними.

Подальший варіант виконання винаходу є, тому, потоком даних або послідовністю сигналів, яка представляє комп'ютерну програму для виконання одного з описаних тут способів. Потік даних або послідовність сигналів може, наприклад, конфігуруватися для передачі по з'єднанню цифрового зв'язку, наприклад по інтернету.

Подальший варіант виконання включає засіб обробки даних, наприклад комп'ютер, або програмований логічний пристрій, сконфігурований або адаптований для виконання одного з описаних тут способів.

Подальший варіант виконання включає комп'ютер, який має інстальовану на ньому комп'ютерну програму для виконання одного з описаних тут способів.

Подальший варіант виконання згідно з винаходом включає пристрій або систему, сконфігуровану для передачі (наприклад, електронними засобами або оптичними засобами) комп'ютерної програми для виконання одного з описаних тут способів до приймача. Приймач може, наприклад, бути комп'ютером, мобільним пристроєм, запам'ятовуючим пристроєм або подібним. Пристрій або система може, наприклад, включати файловий сервер для передачі комп'ютерної програми до приймача.

В деяких варіантах виконання програмований логічний пристрій (наприклад, програмована логічна інтегральна схема) може використовуватися для виконання деяких або усіх функцій описаних тут способів. В деяких варіантах виконання програмована логічна інтегральна схема може взаємодіяти з мікропроцесором для виконання одного з описаних тут способів. Загалом, способи переважно виконуються будь-яким апаратним засобом.

Вищеописані варіанти виконання є просто ілюстративними для принципів представленого винаходу. Зрозуміло, що описані тут модифікації і варіанти розташувань та деталі будуть очевидними для фахівців у цій галузі. Тому, наміром є обмеження тільки правовими рамками наступної формули винаходу, а не спеціальними деталями, представленими описом і поясненням варіантів виконання.

Джерела інформації:

[1] Thomas Wiegand, Gary J. Sullivan, Gisle Bjontegaard, Ajay Luthra, "Overview of the H.264/AVC Video Coding Standard", IEEE Trans. Circuits Syst. Video Technol., vol. 13, N7, July 2003.

[2] JCT-VC, "High-Efficiency Video Coding (HEVC) text specification Working Draft 6", JCTVC-H1003,

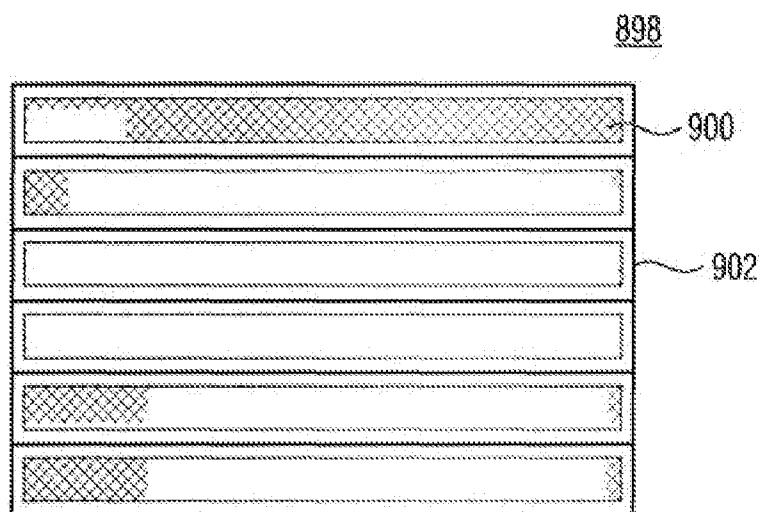


Fig. 1

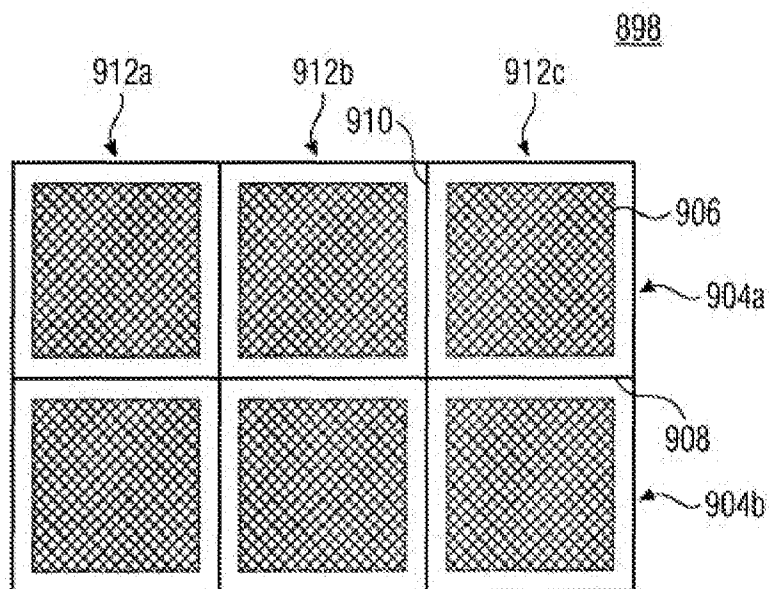


Fig. 2

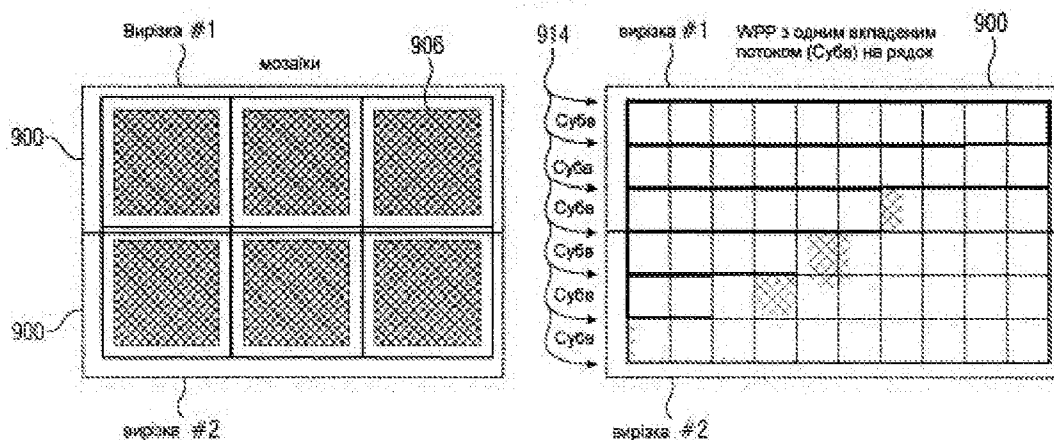


Fig. 3

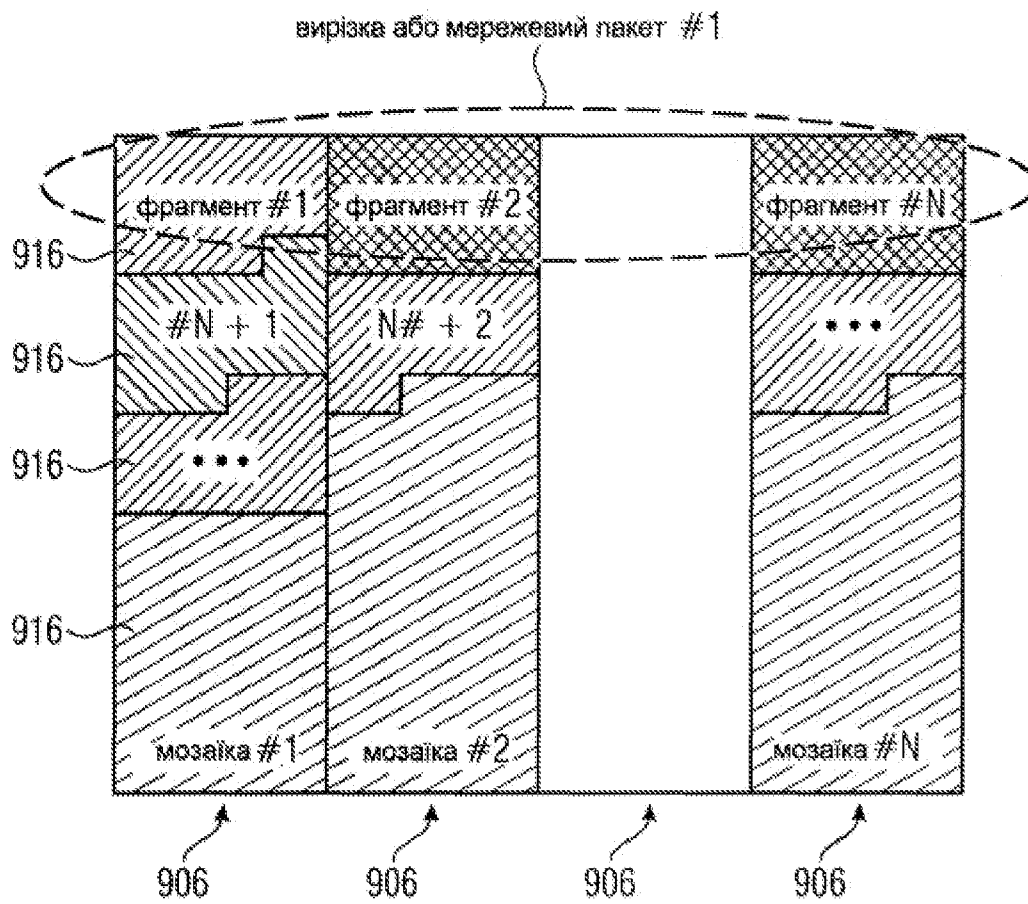


Fig. 4

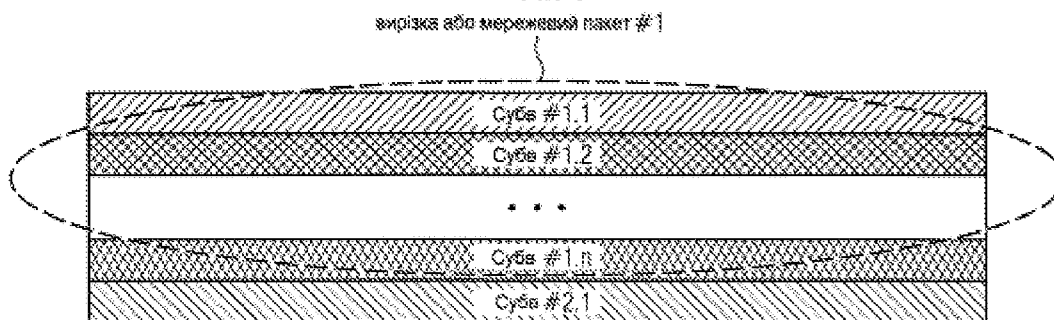
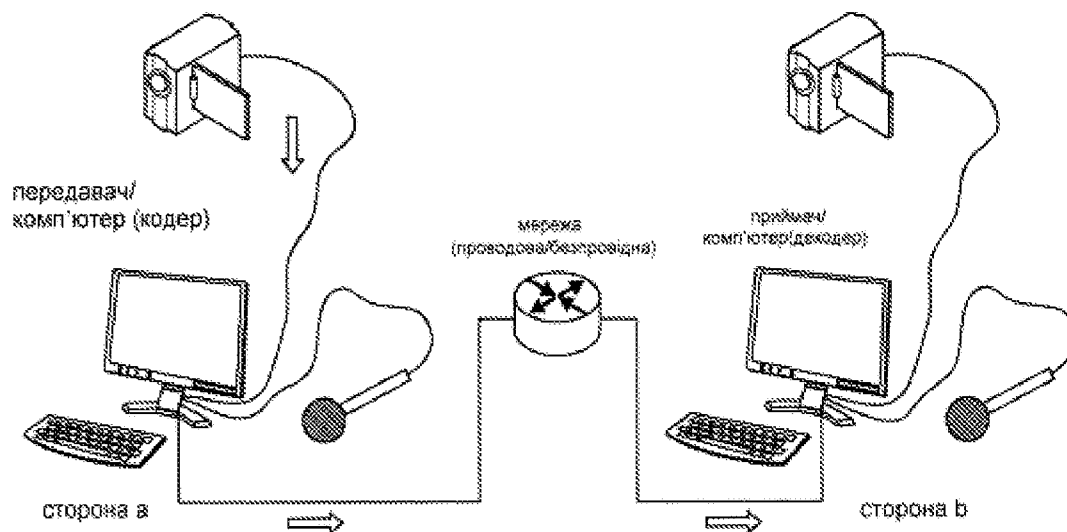
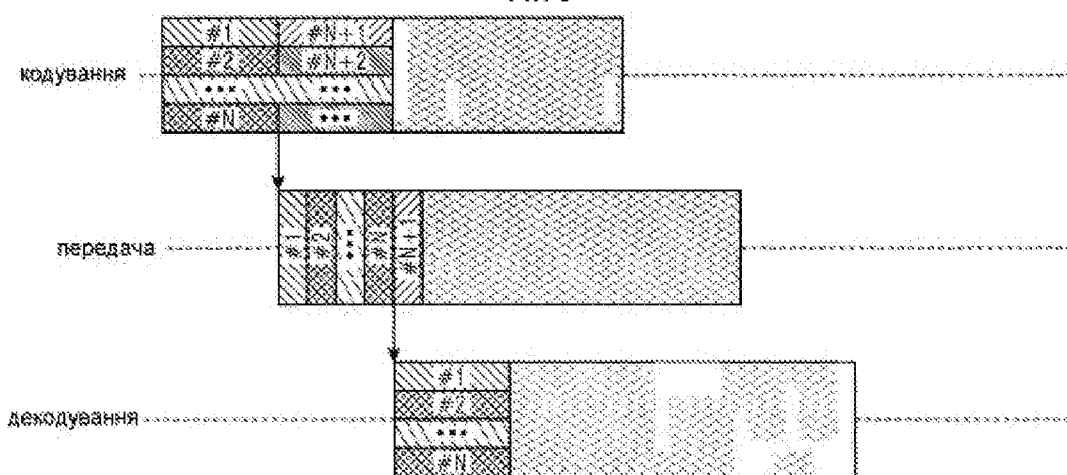


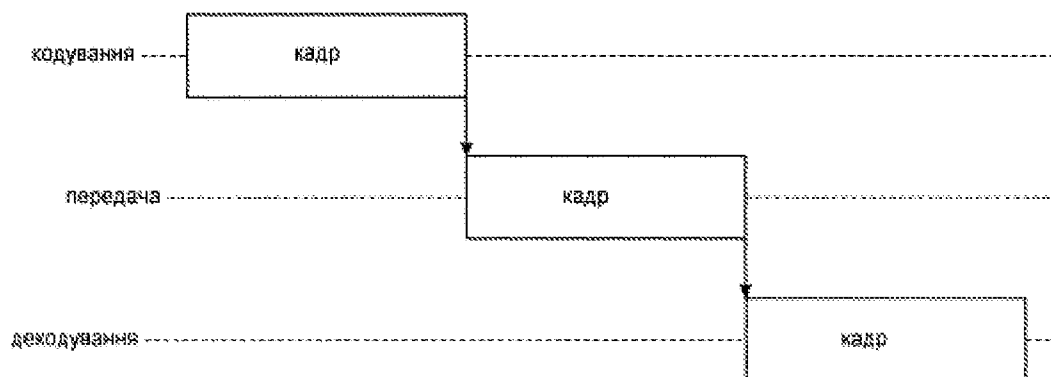
Fig. 5



Фіг. 6



Фіг. 7



Фіг. 8

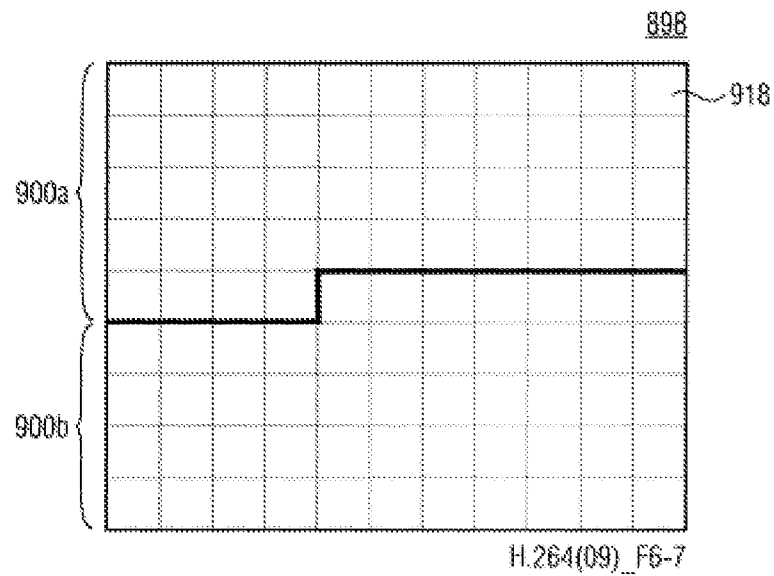


Fig. 9

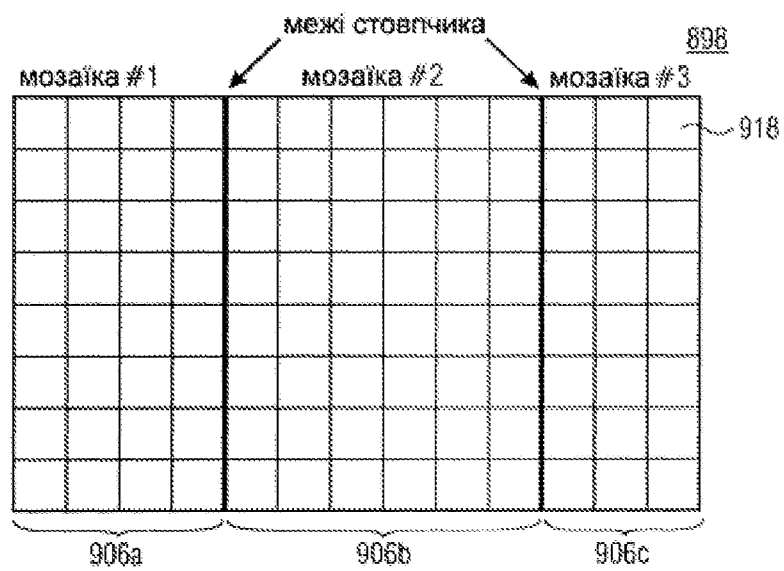


Fig. 10

seq_parameter_set_rbsp{ } {	Descriptor
profile_idc	u(8)
reserved_zero_8bits /* equal to 0 */	u(8)
level_idc	u(8)
seq_parameter_set_id	se(v)
chroma_format_idc	se(v)
if(chroma_format_idc == 3)	
separate_colour_plane_flag	u(1)
max_temporal_layers_minus1	u(3)
pic_width_in_luma_samples	se(v)
pic_height_in_luma_samples	se(v)
pic_cropping_flag	u(1)
if(pic_cropping_flag) {	
pic_crop_left_offset	se(v)
pic_crop_right_offset	se(v)
pic_crop_top_offset	se(v)
pic_crop_bottom_offset	se(v)
}	
bit_depth_luma_minus8	se(v)
bit_depth_chroma_minus8	se(v)
pcm_enabled_flag	u(1)
if(pcm_enabled_flag) {	
pcm_bit_depth_luma_minus1	u(4)
pcm_bit_depth_chroma_minus1	u(4)
}	
qpprime_y_zero_transquant_bypass_flag	u(1)
log2_max_pic_order_cnt_lsb_minus4	se(v)
for(i = 0; i <= max_temporal_layers_minus1; i++) {	
max_dec_pic_buffering[i]	se(v)

ΦΓ. 11	ΦΓ. 11A
	ΦΓ. 11B
	ΦΓ. 11C

ΦΓ. 11A

num_reorder_pics[i]	ue(v)
max_latency_increase[i]	ue(v)
}	
restricted_ref_pic_lists_flag	u(1)
if (restricted_ref_pic_lists_flag)	
lists_modification_present_flag	u(1)
log2_min_coding_block_size_minus3	ue(v)
log2_diff_max_min_coding_block_size	ue(v)
log2_min_transform_block_size_minus2	ue(v)
log2_diff_max_min_transform_block_size	ue(v)
if (pcm_enabled_flag) {	
log2_min_pcm_coding_block_size_minus3	ue(v)
log2_diff_max_min_pcm_coding_block_size	ue(v)
}	
max_transform_hierarchy_depth_inter	ue(v)
max_transform_hierarchy_depth_intra	ue(v)
scaling_list_enable_flag	
chroma_pred_from_luma_enabled_flag	u(1)
deblocking_filter_in_aps_enabled_flag	u(1)
seq_loop_filter_across_slices_enabled_flag	u(1)
asymmetric_motion_partitions_enabled_flag	u(1)
non_square_quadtrees_enabled_flag	u(1)
sample_adaptive_offset_enabled_flag	u(1)
adaptive_loop_filter_enabled_flag	u(1)
if (adaptive_loop_filter_enabled_flag)	
all_coef_in_slice_flag	u(1)
if (pcm_enabled_flag)	
pcm_loop_filter_disable_flag	u(1)
temporal_id_nesting_flag	u(1)

Фиг. 11	Фиг. 11А
	Фиг. 11В
	Фиг. 11С

Фиг. 11В

if(log2_min_coding_block_size_minus3 == 0)	
inter_4x4_enabled_flag	u(1)
num_short_term_ref_pic_sets	ue(v)
for(i = 0; i < num_short_term_ref_pic_sets; i++)	
short_term_ref_pic_set(i)	
long_term_ref_pic_present_flag	u(1)
tiles_or_entropy_coding_sync_idc	u(2)
if(tiles_or_entropy_coding_sync_idc == 1) {	
num_tile_columns_minus1	ue(v)
num_tile_rows_minus1	ue(v)
uniform_spacing_flag	u(1)
if(uniform_spacing_flag) {	
for(i = 0; i < num_tile_columns_minus1; i++)	
column_width[i]	ue(v)
for(i = 0; i < num_tile_rows_minus1; i++)	
row_height[i]	ue(v)
}	
loop_filter_across_tiles_enabled_flag	u(1)
}	
vui_parameters_present_flag	u(1)
if(vui_parameters_present_flag)	
vui_parameters()	
sps_extension_flag	u(1)
if(sps_extension_flag)	
while(more_rbsp_data())	
sps_extension_data_flag	u(1)
rbsp_trailing_bits()	
}	

Фиг. 11	Фиг. 11А
	Фиг. 11В
	Фиг. 11С

Фиг. 11С

pic_parameter_set_rbsp() {	Descriptor
pic_parameter_set_id	ue(v)
seq_parameter_set_id	ue(v)
sign_data_hiding_flag	u(1)
if(sign_data_hiding_flag)	
sign_hiding_threshold	u(4)
cabac_init_present_flag	u(1)
num_ref_idx_l0_default_active_minus1	ue(v)
num_ref_idx_l1_default_active_minus1	ue(v)
pic_init_qp_minus26	se(v)
constrained_intra_pred_flag	u(1)
enable_temporal_mvp_flag	u(1)
slice_granularity	u(2)
max_cu_qp_delta_depth	ue(v)
cb_qp_offset	se(v)
cr_qp_offset	se(v)
weighted_pred_flag	u(1)
weighted_bipred_idc	u(2)
output_flag_present_flag	u(1)
if(tiles_or_entropy_coding_sync_idc == 1) {	
tile_info_present_flag	u(1)
tile_control_present_flag	u(1)
if(tile_info_present_flag) {	
num_tile_columns_minus1	ue(v)

ΦΓ. 12	ΦΓ. 12A
	ΦΓ. 12B

ΦΓ. 12A

num_tile_rows_minus1	ue(v)
uniform_spacing_flag	u(1)
if(uniform_spacing_flag) {	
for(i = 0; i < num_tile_columns_minus1; i++)	
column_width[i]	ue(v)
for(i = 0; i < num_tile_rows_minus1; i++)	
row_height[i]	ue(v)
}	
}	
if(tile_control_present_flag)	
loop_filter_across_tiles_enabled_flag	u(1)
} else if(tiles_or_entropy_coding_sync_idc == 2)	
num_substreams_minus1	ue(v)
deblocking_filter_control_present_flag	u(1)
if(slice_type == P slice_type == B)	
log2_parallel_merge_level_minus2	ue(v)
pps_extension_flag	u(1)
if(pps_extension_flag)	
while(more_rbsp_data())	
pps_extension_data_flag	u(1)
rbsp_trailing_bits()	
}	

Фиг. 12	Фиг. 12А
	Фиг. 12В

Фиг. 12В

slice_header() {	Descriptor
first_slice_in_pic_flag	u(1)
if (first_slice_in_pic_flag == 0)	
slice_address	u(v)
slice_type	se(v)
entropy_slice_flag	u(1)
if (entropy_slice_flag) {	
pic_parameter_set_id	se(v)
if (output_flag_present_flag)	
pic_output_flag	u(1)
if (separate_colour_plane_flag == 1)	
colour_plane_id	u(2)
if (irPicFlag) {	
idr_pic_id	se(v)
no_output_of_prior_pics_flag	u(1)
} else {	
pic_order_cnt_lsb	u(v)
short_term_ref_pic_set_sps_flag	u(1)
if (short_term_ref_pic_set_sps_flag)	
short_term_ref_pic_set(num_short_term_ref_pic_sets)	
else	
short_term_ref_pic_set_idx	u(v)
if (long_term_ref_pics_present_flag) {	
num_long_term_pics	se(v)
for (i = 0; i < num_long_term_pics; i++) {	
delta_poc_lsb_R[i]	se(v)
delta_poc_msb_present_flag[i]	u(1)
if (delta_poc_msb_present_flag[i])	
delta_poc_msb_cycle_R_minus1[i]	se(v)
used_by_curr_pic_R_flag[i]	u(1)
}	
}	
}	
}	
if (sample_adaptive_offset_enabled_flag) {	
slice_sao_interleaving_flag	u(1)
slice_sample_adaptive_offset_flag	u(1)

ΦΓ. 13	ΦΓ. 13A
	ΦΓ. 13B
	ΦΓ. 13C

ΦΓ. 13A

if(slice_sao_interleaving_flag && slice_sample_adaptive_offset_flag) {	
sao_cb_enable_flag	u(1)
sao_cr_enable_flag	u(1)
}	
if(scaling_list_enable_flag & deblocking_filter_in_aps_enabled_flag & (sample_adaptive_offset_enabled_flag && !slice_sao_interleaving_flag) & adaptive_loop_filter_enabled_flag)	
aps_id	se(v)
if(slice_type == P & slice_type == B) {	
num_ref_idx_active_override_flag	u(1)
if(num_ref_idx_active_override_flag) {	
num_ref_idx_l0_active_minus1	se(v)
if(slice_type == B)	
num_ref_idx_l1_active_minus1	se(v)
}	
if(lists_modification_present_flag) {	
ref_pic_list_modification()	
ref_pic_list_combination()	
}	
if(slice_type == B)	
mvd_l1_zero_flag	u(1)
}	
if(cabac_init_present_flag && slice_type != I)	
cabac_init_flag	u(1)
if(entropy_slice_flag) {	
slice_qp_delta	se(v)
if(deblocking_filter_control_present_flag) {	
if(deblocking_filter_in_aps_enabled_flag)	
inherit_dbl_params_from_aps_flag	u(1)
if(inherit_dbl_params_from_aps_flag) {	
disable_deblocking_filter_flag	u(1)
if(disable_deblocking_filter_flag) {	
bata_offset_div2	se(v)

Фиг. 13	Фиг. 13A
	Фиг. 13B
	Фиг. 13C

Фиг. 13B

ic_offset_div2	se(v)
}	
}	
}	
if(slice_type == 0)	
collocated_from_id_flag	u(1)
if(slice_type != 1 &&	
((collocated_from_id_flag && num_ref_idx_id_active_minus1 > 0)	
(collocated_from_id_flag && num_ref_idx_l1_active_minus1 > 0))	
collocated_ref_idx	se(v)
if((weighted_pred_flag && slice_type == P)	
(weighted_bipred_flag == 1 && slice_type == B))	
pred_weight_table()	
}	
if(slice_type == P slice_type == B)	
five_minus_max_num_merge_cand	se(v)
if(adaptive_loop_filter_enabled_flag) {	
slice_adaptive_loop_filter_flag	u(1)
if(slice_adaptive_loop_filter_flag && all_cscf_in_slice_flag)	
all_param()	
if(slice_adaptive_loop_filter_flag && half_cscf_in_slice_flag)	
all_cu_control_param()	
}	
if(seq_loop_filter_across_slices_enabled_flag &&	
(slice_adaptive_loop_filter_flag slice_sample_adaptive_offset_flag	
disable_deblocking_filter_flag))	
slice_loop_filter_across_slices_enabled_flag	u(1)
if(hlas_or_entropy_coding_sync_idc > 0) {	
num_entry_point_offsets	se(v)
if(num_entry_point_offsets > 0) {	
offset_ten_minus1	se(v)
for(i = 0; i < num_entry_point_offsets; i++)	
entry_point_offset[i]	u(v)
}	
}	
}	

Фиг. 13	Фиг. 13A
	Фиг. 13B
	Фиг. 13C

Фиг. 13C

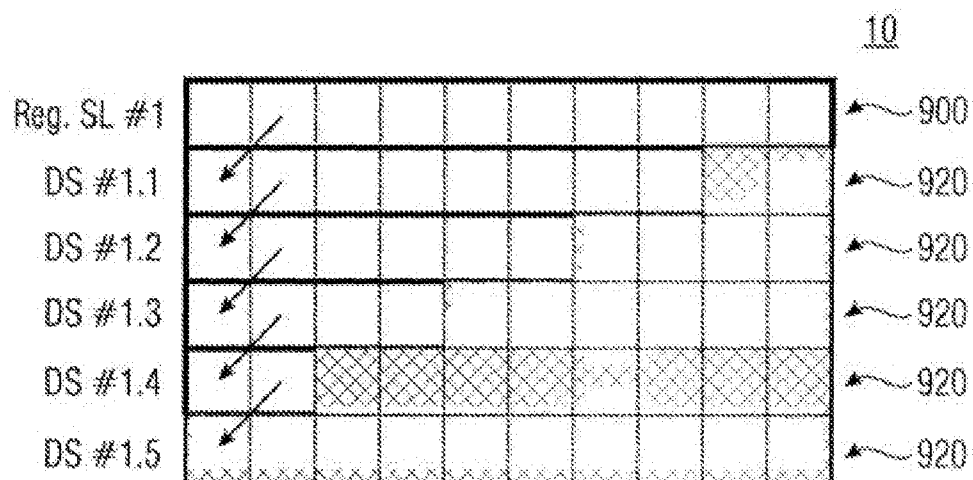


Fig. 14

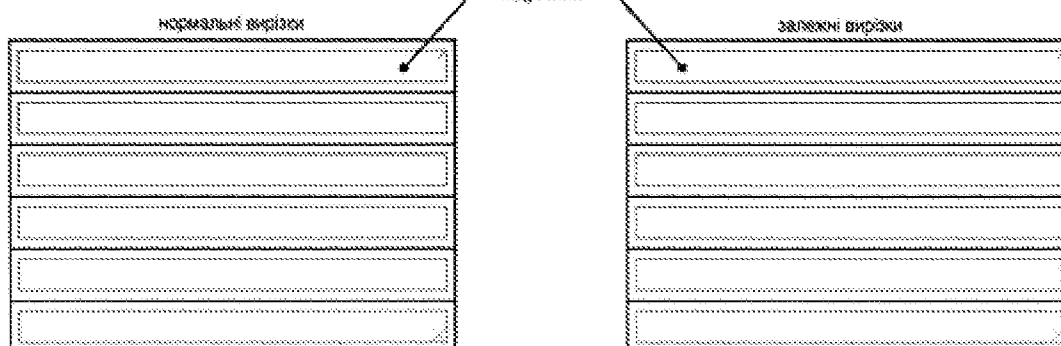
pic_parameter_set_rbsp() {	Descriptor
...	
dependent_slices_present_flag	u(1)
last_ctb_cabac_init_flg	u(1)
}	

Fig. 15

slice_header() {	Descriptor
first_slice_in_pic_flag	u(1)
if (first_slice_in_pic_flag == 0) {	
slice_address	u(v)
slice_type	ue(v)
entropy_slice_flag	u(1)
dependent_slice_flag	u(1)
if (entropy_slice_flag) {	
if (dependent_slice_flag) {	
* * *	
}	
}	
no_cabac_reset_flag	u(1)
}	
* * *	
if (cabac_init_present_flag && slice_type != I && !no_cabac_reset_flag) {	
cabac_init_flag	u(1)
* * *	
if (slice_type == P slice_type == B) {	
five_minus_max_num_merge_cand	ue(v)
if (adaptive_loop_filter_enabled_flag) {	
slice_adaptive_loop_filter_flag	u(1)
if (slice_adaptive_loop_filter_flag && all_coef_in_slice_flag) {	
all_param()	
if (slice_adaptive_loop_filter_flag && half_coef_in_slice_flag) {	
all_cu_control_param()	
}	
if (seq_loop_filter_across_slices_enabled_flag &&	
(slice_adaptive_loop_filter_flag slice_sample_adaptive_offset_flag	
disable_deblocking_filter_flag)) {	
slice_loop_filter_across_slices_enabled_flag	u(1)
if (tiles_or_entropy_coding_sync_idc >= 0) {	
num_entry_point_offsets	ue(v)
if (num_entry_point_offsets > 0) {	
offset_len_minus1	ue(v)
for (i = 0; i < num_entry_point_offsets; i++) {	
entry_point_offset[i]	u(v)
}	
}	
}	
}	

Фиг. 16

зависимости
кодирования



Фиг. 17

pic_parameter_set_rbsp() {	Descriptor
pic_parameter_set_id	ue(v)
seq_parameter_set_id	ue(v)
sign_data_hiding_flag	u(1)
if(sign_data_hiding_flag)	
sign_hiding_threshold	u(4)
cabac_init_present_flag	u(1)
num_ref_idx_l0_default_active_minus1	ue(v)
num_ref_idx_l1_default_active_minus1	ue(v)
{Ed. (B8): not present in HM software }	
pic_init_qp_minus26	se(v)
constrained_intra_pred_flag	u(1)
slice_granularity	u(2)
diff_cu_qp_delta_depth	ue(v)
cb_qp_offset	se(v)
cr_qp_offset	se(v)
weighted_pred_flag	u(1)
weighted_bipred_idc	u(2)
output_flag_present_flag	u(1)
transquant_bypass_enable_flag	u(1)
dependent_slice_enabled_flag	u(1)
tiles_or_entropy_coding_sync_idc	u(2)
if(tiles_or_entropy_coding_sync_idc $\neq$ 1) {	
// num tile columns minus1	ue(v)
// num tile rows minus1	ue(v)
// uniform_spacing_flag	u(1)
if(uniform_spacing_flag) {	
for(i = 0; i < num tile columns minus1; i++)	
// column_width[i]	ue(v)
for(i = 0; i < num tile rows minus1; i++)	
// row_height[i]	ue(v)
} // }	
// loop filter across tiles enabled_flag	u(1)
} else if(tiles_or_entropy_coding_sync_idc $\neq$ 3)	
// cabac_independent_flag	u(1)
. . .	
}	

Fig. 22

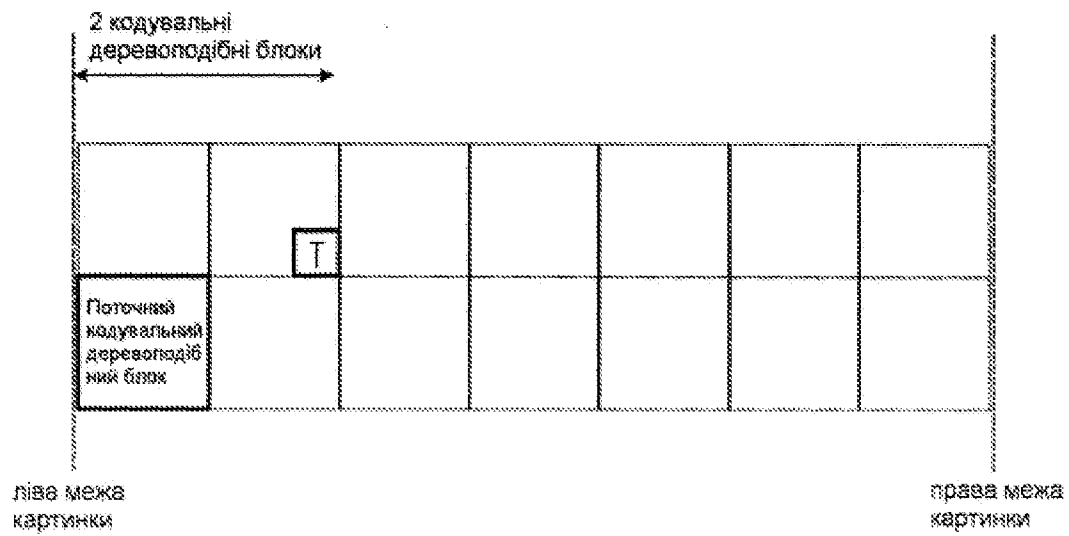


Fig. 23

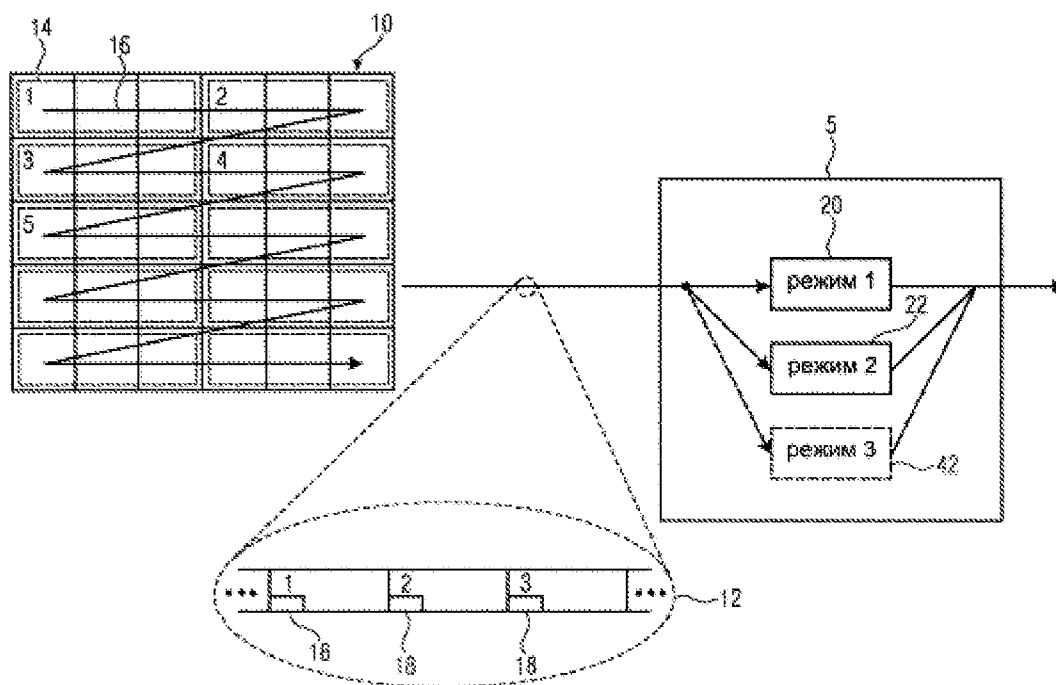


Fig. 24

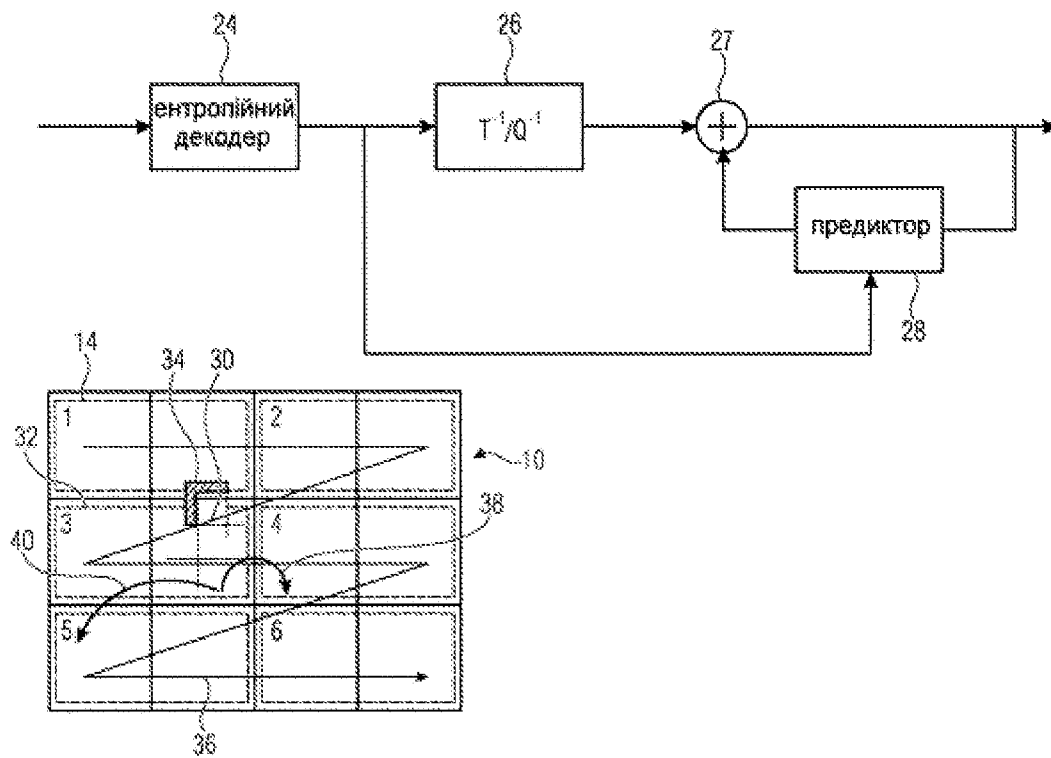


Fig. 25

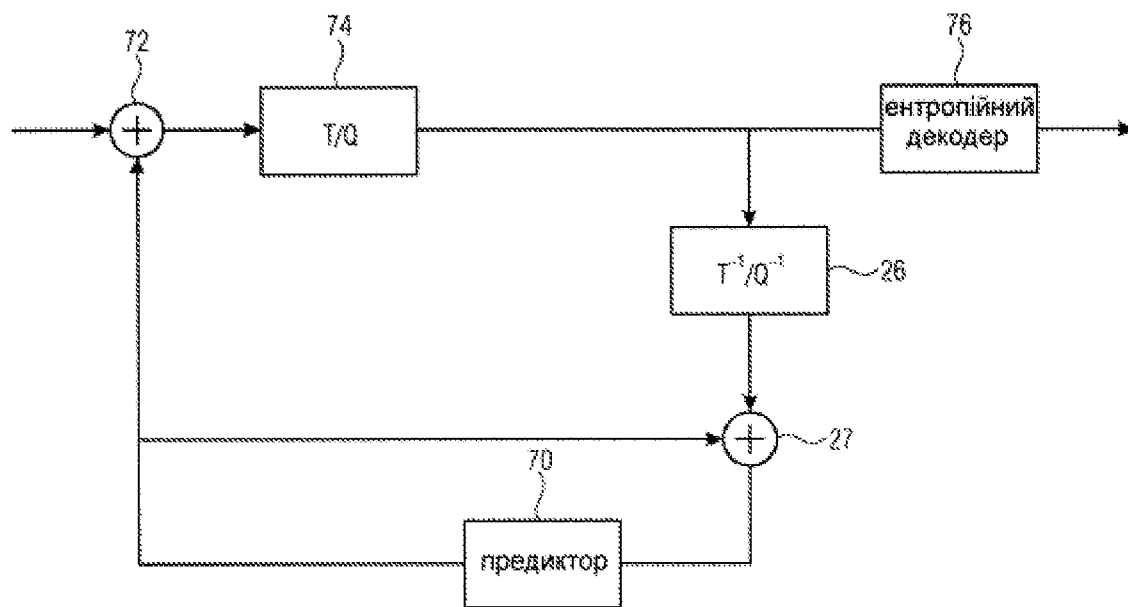
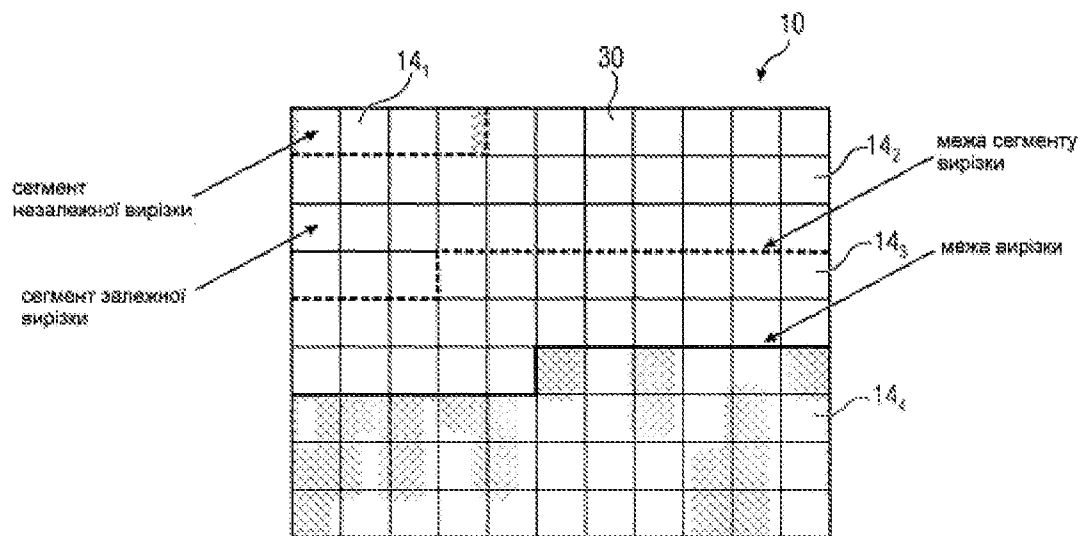
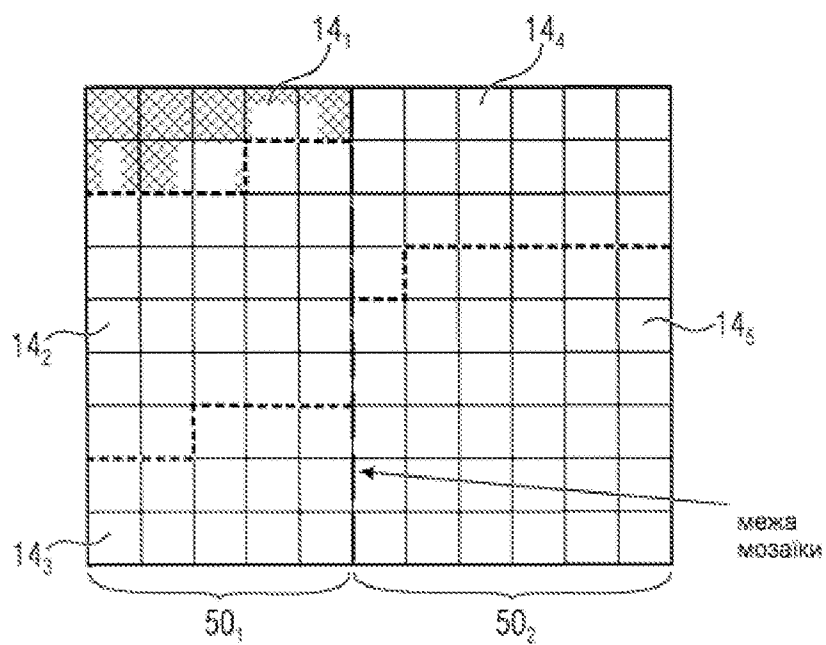


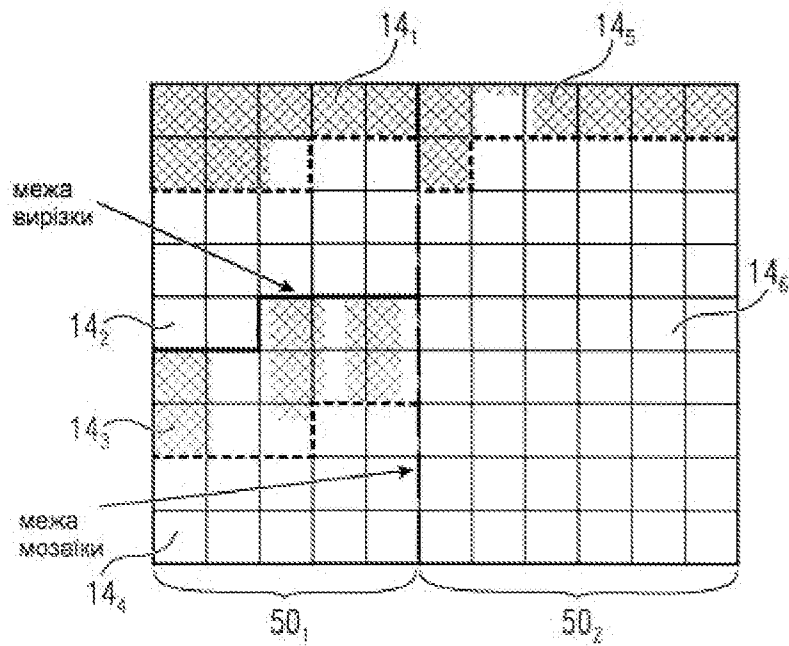
Fig. 26



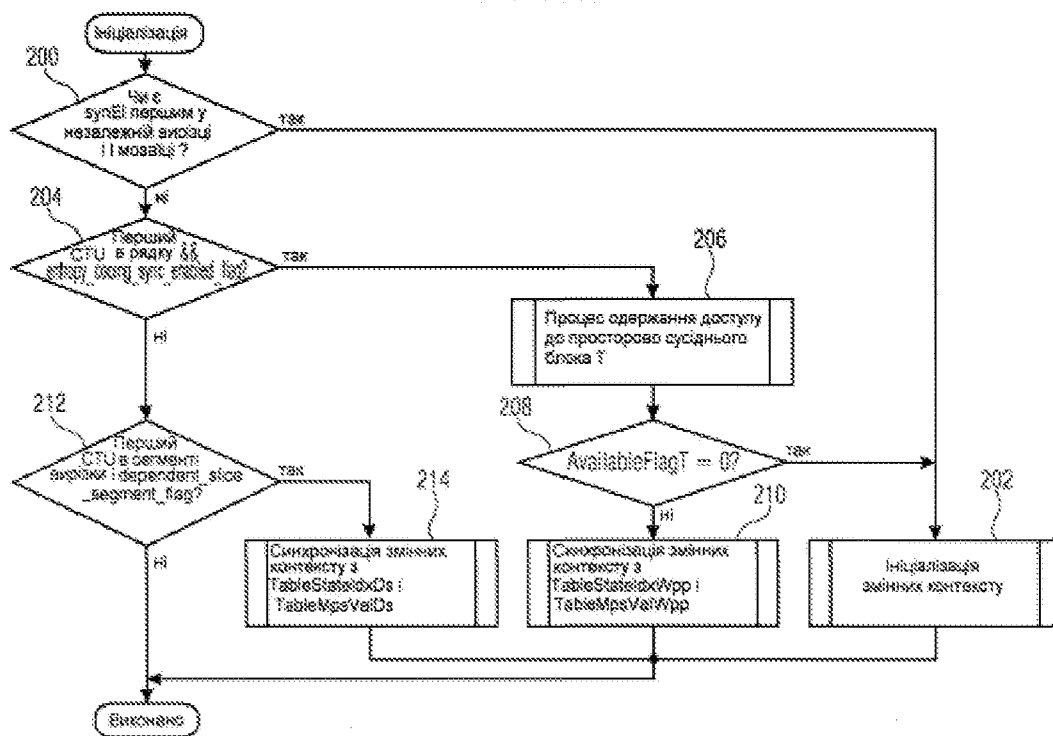
Фиг. 27



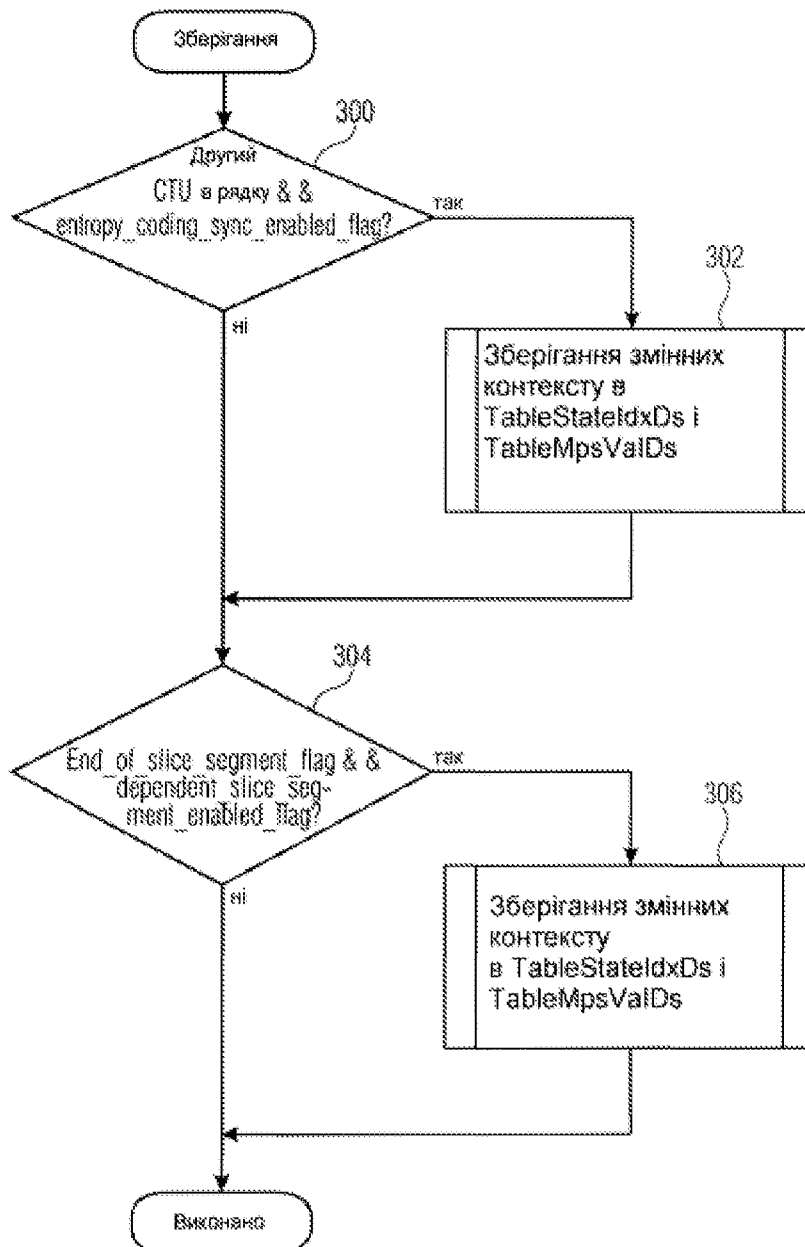
Фиг. 28А



Фіг. 28В



Фіг. 29



Фіг. 30

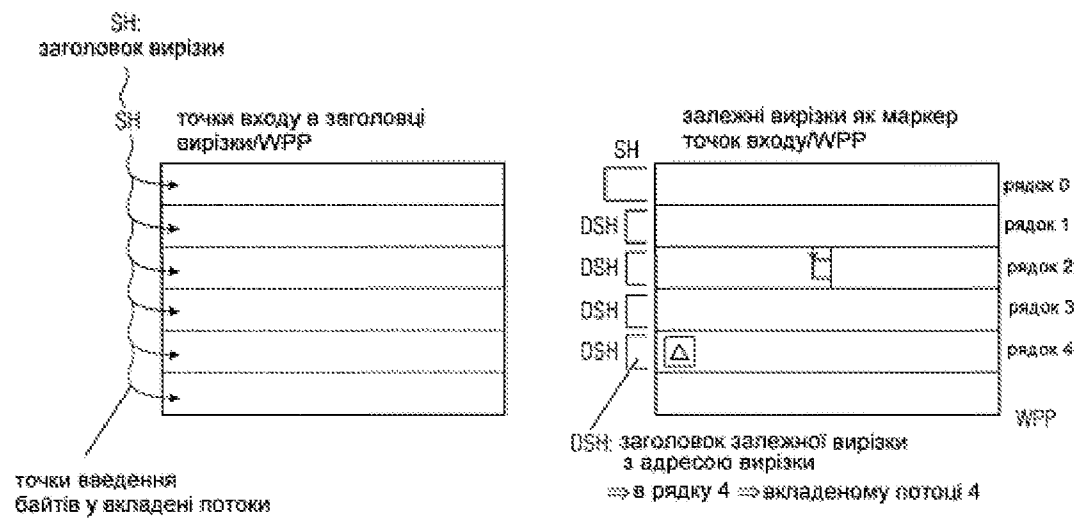


Fig. 31