(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 983 190**

(51) Int. Cl.:

H04N 19/33 (2014.01)
H04N 19/91 (2014.01)
H04N 19/93 (2014.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **01.08.2019 PCT/GB2019/052166**

(87) Fecha y número de publicación internacional: **06.02.2020 WO20025964**

(96) Fecha de presentación y número de la solicitud europea: **01.08.2019 E 19752544 (7)**

(97) Fecha y número de publicación de la concesión europea: **10.04.2024 EP 3831065**

(54) Título: **Codificación por entropía para codificación de mejora de señal**

(30) Prioridad:

03.08.2018 GB 201812708
03.08.2018 GB 201812709
03.08.2018 GB 201812710
20.03.2019 GB 201903844
23.03.2019 GB 201904014
29.03.2019 GB 201904492
15.04.2019 GB 201905325

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
22.10.2024

(73) Titular/es:

V-NOVA INTERNATIONAL LTD (100.0%)
1 Sheldon Square
Paddington London W2 6TT, GB

(72) Inventor/es:

MEARDI, GUIDO

(74) Agente/Representante:

UNGRÍA LÓPEZ, Javier

DESCRIPCIÓN

Codificación por entropía para codificación de mejora de señal

Antecedentes

Se ha propuesto previamente una tecnología de codificación híbrida compatible con versiones anteriores, por ejemplo, en los documentos WO 2014/170819 y WO 2018/046940.

En los mismos se propone un método que analiza un flujo de datos en primeras porciones de datos codificados y segundas porciones de datos codificados; implementa un primer decodificador para decodificar las primeras porciones de datos codificados en una primera representación de una señal; implementa un segundo decodificador para decodificar las segundas porciones de datos codificados en datos de reconstrucción, especificando los datos de reconstrucción cómo modificar la primera representación de la señal; y aplica los datos de reconstrucción a la primera representación de la señal para producir una segunda representación de la señal.

Se propone adicionalmente una adición en los mismos en la que un conjunto de elementos de restos es utilizable para reconstruir una representación de una primera muestra de tiempo de una señal. Se genera un conjunto de elementos de correlación espacio-temporal asociados con la primera muestra de tiempo. El conjunto de elementos de correlación espacio-temporal es indicativo de una extensión de correlación espacial entre una pluralidad de elementos de restos y una extensión de correlación temporal entre primeros datos de referencia basados en la representación y segundos datos de referencia basados en una representación de una segunda muestra de tiempo de la señal. El conjunto de elementos de correlación espacio-temporal se usa para generar datos de salida.

Las tecnologías de codificación de vídeo convencionales implican aplicar una operación de codificación por entropía a datos de salida. Existe la necesidad de un esquema de codificación por entropía de baja complejidad, simple y rápido para aplicar compresión de datos a los datos de reconstrucción o elementos de restos de las tecnologías propuestas anteriormente u otros datos de restos similares.

La publicación titulada "H.264 and MPEG-4 Video Compression, chapter 6, H.264/MPEG4 Part 10", de Iain E. Richardson, menciona que H.264 proporciona mecanismos para codificar vídeo que están optimizados para la eficiencia de compresión y tienen como objetivo satisfacer las necesidades de las aplicaciones prácticas de comunicación multimedia.

El documento US2009097548A1 describe técnicas de codificación de vídeo escalables. El documento WO2014170819A1 describe que el hardware de procesador informático analiza un flujo de datos en primeras porciones de datos codificados y segundas porciones de datos codificados; implementa un primer decodificador para decodificar las primeras porciones de datos codificados en una primera representación de una señal; implementa un segundo decodificador para decodificar las segundas porciones de datos codificados en datos de reconstrucción, especificando los datos de reconstrucción cómo modificar la primera representación de la señal; y aplica los datos de reconstrucción a la primera representación de la señal para producir una segunda representación de la señal.

Sumario de la invención

De acuerdo con la invención definida por las reivindicaciones adjuntas, se proporciona un método de decodificación de una señal de vídeo, un aparato de decodificación, un medio legible por ordenador y un flujo de bits codificado.

De acuerdo con un ejemplo que no forma parte de la invención reivindicada, se proporciona un método de codificación de una señal de vídeo.

De esta manera, el método de codificación proporciona una solución de baja complejidad, simple y rápida para la compresión de datos de los datos de restos. La solución aprovecha las características únicas de los datos de restos, tales como la aparición relativa de valores cero, la agrupación probable de los valores cero basándose en un orden de exploración de un proceso de transformada para generar los datos de restos (si se usa una transformada) y la variedad relativa de los valores de datos y su frecuencia/infrecuencia relativa potencial en los datos de restos.

El conjunto de símbolos puede ser secuencial en el flujo de bytes codificado. Los recuentos de valores cero consecutivos también pueden denominarse como una serie de ceros. Los datos de restos sobre los que se realiza la operación de codificación de longitud de serie pueden representar datos de restos cuantificados, preferiblemente un conjunto cuantificado de coeficientes de transformada. El conjunto cuantificado de coeficientes de transformada puede ordenarse por capa, es decir, por conjuntos de coeficientes del mismo tipo, por plano, por nivel de calidad o por superficie. Estos términos se describen y definen adicionalmente en el presente documento. Los símbolos pueden ser secuenciales en un orden de exploración correspondiente de la operación de transformada de tal manera que los datos de restos pueden coincidir fácilmente con un cuadro reconstruido de referencia.

Preferiblemente, la operación de codificación de longitud de serie comprende: codificar valores de datos distintos de

cero de los datos de restos en al menos un primer tipo de símbolo; y codificar recuentos de valores cero consecutivos en un segundo tipo de símbolo, de manera que los datos de restos se codifican como una secuencia de símbolos de diferentes tipos. Por lo tanto, los datos de restos pueden codificarse como una secuencia de símbolos que comprende un tipo de valor de datos y un tipo de serie de ceros. Los tipos facilitan la velocidad y la facilidad de decodificación en el decodificador, pero facilitan también una operación de codificación por entropía adicional como se elabora a continuación. Estructurar los datos en un conjunto minimizado de códigos de longitud fija de diferentes tipos es beneficioso para etapas posteriores.

En ciertos ejemplos, la operación de codificación de longitud de serie comprende: codificar valores de datos de los datos de restos en un primer tipo de símbolo y un tercer tipo de símbolo, comprendiendo cada uno del primer y tercer tipos de símbolo una parte de un valor de datos, de manera que las partes se pueden combinar en un decodificador para reconstruir el valor de datos. Cada tipo de símbolo puede tener un tamaño fijo y puede ser un byte. Los valores de datos mayores que un umbral o mayores que un tamaño disponible en un tipo de símbolo pueden transmitirse o almacenarse fácilmente usando el flujo de bytes. La estructuración de los datos en bytes o símbolos no solo facilita la decodificación sino que también facilita la codificación por entropía de símbolos de longitud fija como se expone a continuación. La introducción del tercer tipo de símbolo permite distinguir fácilmente tres tipos de símbolos entre sí en el lado del decodificador. Cuando nos referimos a tipos de símbolos, podrían usarse los términos bloques, bytes (donde el símbolo es un byte), contextos o tipos.

La operación de codificación de longitud de serie puede comprender: comparar un tamaño de cada valor de datos de los datos de restos a codificar con un umbral; y codificar cada valor de datos en el primer tipo de símbolo si el tamaño está por debajo del umbral y codificar una parte de cada valor de datos en el primer tipo de símbolo y una parte de cada valor de datos en el tercer tipo de símbolo si el tamaño está por encima del umbral.

Si el tamaño está por encima del umbral, el método puede comprender establecer un indicador en el símbolo del primer tipo de símbolo que indica que una parte del valor de datos representado se codifica en un símbolo adicional del tercer tipo de símbolo. El indicador puede ser un indicador de desbordamiento o un bit de desbordamiento y puede, en ciertos ejemplos, ser el bit menos significativo del símbolo o byte. Cuando el bit menos significativo del símbolo es un indicador, el valor de datos o parte del valor de datos puede estar comprendido en los bits restantes del byte. Establecer el bit de desbordamiento como el bit menos significativo facilita la facilidad de combinación con símbolos posteriores.

El método puede comprender adicionalmente insertar un indicador en cada símbolo que indica un tipo de símbolo codificado a continuación en el flujo de bytes codificado de longitud de serie. El indicador puede ser un bit de desbordamiento como se ha descrito anteriormente o puede ser además un indicador de serie o bit de serie que indica si el siguiente símbolo comprende o no una serie de ceros o un valor de datos o una parte de un valor de datos. Cuando el indicador representa si el siguiente símbolo comprende una serie (o un recuento de ceros consecutivos), el indicador puede ser un bit del símbolo, preferiblemente un bit más significativo del símbolo. El recuento de ceros consecutivos o el valor de datos puede estar comprendido en los bits restantes del símbolo. Por lo tanto, la 'serie de ceros' o segundo tipo de símbolo comprende 7 bits disponibles del byte para un recuento y los 'datos' o primer tipo de símbolo comprenden 6 bits disponibles para un valor de datos donde el bit de desbordamiento no está establecido y 7 bits disponibles para un valor de datos donde se establece el bit de desbordamiento.

En resumen, el indicador puede ser diferente para cada tipo de símbolo y puede indicar el tipo de símbolo siguiente en el flujo.

El método comprende adicionalmente aplicar una operación de codificación por entropía adicional al conjunto de símbolos producidos por la operación codificada por longitud de serie. Por lo tanto, los símbolos de longitud fija producidos deliberadamente por la estructura de la operación de codificación de longitud de serie pueden convertirse en códigos de longitud variable para reducir el tamaño global de los datos. La estructura de codificación de longitud de serie está diseñada para crear símbolos de longitud fija de alta frecuencia para facilitar una operación de codificación por entropía adicional mejorada y una reducción general en el tamaño de datos. La operación de codificación por entropía adicional aprovecha la probabilidad o frecuencia de que se produzca un símbolo en el flujo de bytes creado por la operación de codificación de longitud de serie.

La operación de codificación por entropía adicional es una operación de codificación Huffman o una operación de codificación aritmética. Preferiblemente, el método comprende adicionalmente aplicar una operación de codificación Huffman al conjunto de símbolos para generar datos codificados de Huffman que comprenden un conjunto de códigos que representan el flujo de bytes codificado de longitud de serie. La operación de codificación Huffman recibe como entradas símbolos del flujo de bytes codificado de longitud de serie y emite una pluralidad de códigos de longitud variable en un flujo de bits. La codificación Huffman es una técnica de codificación que puede reducir eficazmente códigos de longitud fija. La estructura de los bytes codificados de longitud de serie significa que es muy probable que los símbolos de los dos tipos se repliquen con frecuencia, lo que significa que solo se necesitarán unos pocos bits para cada símbolo recurrente. A través de un plano, es probable que se replique el mismo valor de datos (particularmente donde se cuantifican los restos) y, por tanto, el código de longitud variable puede ser pequeño (pero repetido).

La codificación Huffman está también bien optimizada para la implementación de software (como se anticipa en el presente documento) y es computacionalmente eficiente. La misma usa memoria mínima y tiene una decodificación más rápida en comparación con otras técnicas de codificación por entropía, ya que las etapas de procesamiento implican simplemente recorrer un árbol. Los beneficios computacionales surgen de la forma y profundidad del árbol creado por la operación de codificación de longitud de serie diseñada específicamente.

Las operaciones de codificación pueden realizarse en un grupo de coeficientes, es decir, una capa, en un plano de un cuadro, en un nivel de calidad, o en una superficie o cuadro completo. Es decir, las estadísticas o parámetros de cada operación pueden determinarse basándose en el grupo de valores de datos y valores de datos cero a codificar o los símbolos que representan esos valores.

La operación de codificación Huffman puede ser una operación de codificación Huffman canónica, de manera que los datos codificados de Huffman comprenden una longitud de código para cada símbolo único del conjunto de símbolos, representando la longitud de código una longitud de un código usado para codificar un símbolo correspondiente. La codificación Huffman canónica facilita una reducción en parámetros de codificación que necesitan señalizarse entre el codificador y el decodificador en metadatos asegurando que se unen longitudes de código idénticas a símbolos de una manera secuencial. El libro de códigos para el decodificador puede inferirse y solo es necesario enviar las longitudes de código. Por tanto, el codificador canónico de Huffman es particularmente eficiente para árboles poco profundos donde hay un gran número de valores de datos diferentes.

El método comprende comparar un tamaño de datos de al menos una porción del flujo de bytes codificado de longitud de serie con un tamaño de datos de al menos una porción de los datos codificados por Huffman; y, emitir el flujo de bytes codificado por longitud de serie o los datos codificados por Huffman en un flujo de bits de salida basándose en la comparación. Cada bloque o sección de los datos de entrada se envía de este modo selectivamente para reducir el tamaño de datos global asegurando que el decodificador puede diferenciar fácilmente entre los tipos de codificación. La diferencia puede estar dentro de un umbral o tolerancia tal que aunque un esquema puede dar como resultado datos más pequeños, las eficiencias computacionales pueden significar que se prefiere un esquema.

Para distinguir entre esquemas, el método puede comprender adicionalmente añadir un indicador a los metadatos de configuración que acompañan al flujo de bits de salida que indica si el flujo de bits representa el flujo de bytes codificado de longitud de serie o los datos codificados de Huffman. Esta es una forma rápida y eficiente de señalización. Como alternativa, el decodificador puede identificar el esquema usado a partir de la estructura de los datos, por ejemplo, los datos codificados de Huffman pueden tener una parte de encabezado y una parte de datos, mientras que los datos codificados de longitud de serie pueden comprender solo una parte de datos y, por tanto, el decodificador puede ser capaz de distinguir entre los dos esquemas.

La operación de codificación Huffman puede comprender generar una tabla de frecuencia separada para símbolos del primer tipo de símbolo y símbolos del segundo tipo de símbolo. Además, puede aplicarse una operación de codificación Huffman separada para cada tipo de símbolo. Estos conceptos facilitan la eficiencia particular del esquema de codificación de longitud variable. Por ejemplo, cuando el mismo valor de datos se replica a través de un plano, como es probable en la codificación de vídeo donde una escena puede tener colores o errores/mejoras similares, solo se pueden necesitar unos pocos códigos para estos valores (donde los códigos no están distorsionados por el tipo de símbolo de 'serie de ceros'). Por lo tanto, este enfoque es particularmente ventajoso para su uso en conjunto con la operación de codificación de longitud de serie anterior. Como se ha indicado anteriormente, cuando se cuantifican los valores, los símbolos pueden replicarse allí y pueden no ser demasiados valores diferentes dentro del mismo cuadro o grupo de coeficientes.

El uso de una tabla de frecuencia diferente para cada tipo de símbolo proporciona un beneficio particular en la operación de codificación Huffman después de la operación de codificación de longitud de serie específicamente diseñada.

El método puede comprender, por tanto, identificar un tipo de símbolo a codificar a continuación, seleccionar una tabla de frecuencia basándose en el tipo del siguiente símbolo, decodificar el siguiente símbolo usando la tabla de frecuencia seleccionada y emitir el símbolo codificado en secuencia. Una tabla de frecuencia puede corresponder a un libro de códigos único.

El método puede comprender generar un encabezado de flujo que puede comprender una indicación de la pluralidad de longitudes de código, de manera que un decodificador puede derivar longitudes de código y símbolos correspondientes respectivos para la operación de decodificación Huffman canónica. Las longitudes de código pueden ser, por ejemplo, la longitud de cada código usado para codificar un símbolo particular. Por lo tanto, una longitud de código tiene un código correspondiente y un símbolo correspondiente. El encabezado de flujo puede ser un primer tipo de encabezado de flujo y comprende adicionalmente una indicación del símbolo asociado con una respectiva longitud de código de la pluralidad de longitudes de código. Como alternativa, el encabezado de flujo puede ser un segundo tipo de encabezado de flujo y el método puede comprender además ordenar la pluralidad de longitudes de código en el encabezado de flujo basándose en un orden predeterminado de símbolos que corresponden a cada una

de las longitudes de código, de modo que las longitudes de código pueden asociarse con un símbolo correspondiente en el decodificador. Cada tipo proporcionará una señalización eficiente de los parámetros de codificación dependiendo de los símbolos y longitudes a decodificar. Las longitudes de código pueden señalizarse como una diferencia entre la longitud y otra longitud, preferiblemente una longitud de código mínima señalizada.

El segundo tipo de encabezado de flujo puede comprender adicionalmente un indicador que indica que un símbolo en el orden predeterminado del posible conjunto de símbolos no existe en el conjunto de símbolos en el flujo de bytes codificado de longitud de serie. De esta manera, solo las longitudes necesarias necesitan incluirse en el encabezado.

El método puede comprender adicionalmente comparar un número de códigos únicos en los datos codificados de Huffman con un umbral y generar el primer tipo de encabezado de flujo o el segundo tipo de encabezado de flujo basándose en la comparación.

El método puede comprender adicionalmente comparar un número de símbolos distintos de cero o símbolos de datos con un umbral y generar el primer tipo de encabezado de flujo o el segundo tipo de encabezado de flujo basándose en la comparación.

De acuerdo con un aspecto de la invención, se proporciona un método de decodificación de una señal de vídeo de acuerdo con la reivindicación 1 adjunta.

La operación de codificación de longitud de serie comprende identificar símbolos de un primer tipo de símbolo que representa valores de datos distintos de cero de los datos de restos; identificar símbolos de un segundo tipo de símbolo que representa recuentos de valores cero consecutivos, de modo que se decodifica una secuencia de símbolos de diferentes tipos para generar los datos de restos; y analizar el conjunto de símbolos de acuerdo con el tipo respectivo de cada símbolo.

La operación de codificación de longitud de serie comprende identificar símbolos de un primer tipo de símbolo y símbolos de un tercer tipo de símbolo, representando cada uno el primer y tercer tipos de símbolo una parte de un valor de datos; analizar un símbolo del primer tipo de símbolo y un símbolo del tercer tipo de símbolo para derivar partes de un valor de datos; y combinar las partes derivadas del valor de datos en un valor de datos.

El método puede comprender adicionalmente: recuperar un indicador de desbordamiento de un símbolo del primer tipo de símbolo que indica si una parte de un valor de datos del símbolo del primer tipo de símbolo está comprendida en un tercer tipo de símbolo posterior.

El método puede comprender adicionalmente recuperar un indicador de cada símbolo que indica un tipo posterior de símbolo que se espera en el conjunto de símbolos.

Se puede suponer que un símbolo inicial es el primer tipo de símbolo, de manera que el símbolo inicial se analiza para derivar al menos una parte de un valor de datos.

La operación de codificación Huffman puede ser una operación de codificación Huffman canónica.

El método puede comprender además recuperar iterativamente un indicador de un símbolo decodificado que indica un tipo posterior de símbolo que se espera derivar del flujo de bits y aplicar la operación de codificación Huffman al flujo de bits para derivar un símbolo posterior basándose en un conjunto de parámetros de codificación asociados con el tipo posterior de símbolo que se espera derivar del flujo de bits.

El método puede comprender adicionalmente: recuperar un encabezado de flujo que comprende una indicación de una pluralidad de longitudes de código a usar para la operación de codificación canónica de Huffman; asociar cada longitud de código con un respectivo símbolo de acuerdo con la operación de codificación canónica de Huffman; e identificar un código asociado con cada símbolo basándose en las longitudes de código de acuerdo con la operación de codificación canónica de Huffman. Por tanto, los códigos pueden asociarse con símbolos secuenciales donde las longitudes de código son idénticas.

El encabezado de flujo puede comprender una indicación de un símbolo asociado con una longitud de código respectiva de la pluralidad de longitudes de código y la etapa de asociar cada longitud de código con un símbolo comprende asociar cada longitud de código con el símbolo correspondiente en el encabezado de flujo.

La etapa de asociar cada longitud de código con un respectivo símbolo comprende asociar cada longitud de código con un símbolo de un conjunto de símbolos predeterminados de acuerdo con un orden en el que se recuperó cada longitud de código.

El método puede comprender adicionalmente no asociar un símbolo del conjunto de símbolos predeterminados con una respectiva longitud de código donde un indicador del encabezado de flujo indica que no existe longitud de código en el encabezado de flujo para ese símbolo.

De acuerdo con un ejemplo que no cae dentro del alcance de la invención reivindicada, se proporciona un método de codificación de una señal de vídeo.

El método comprende: recibir un cuadro de entrada; procesar el cuadro de entrada para generar datos de restos, permitiendo los datos de restos que un decodificador reconstruya el cuadro de entrada a partir de un cuadro reconstruido de referencia; y aplicar una operación de codificación Huffman a los datos de restos, en donde la operación de Huffman comprende generar un flujo de bits codificado que comprende un conjunto de códigos que codifican un conjunto de símbolos que representan los datos de restos.

De acuerdo con un aspecto adicional de la invención, se proporciona un aparato de decodificación de acuerdo con la reivindicación 2 adjunta.

Un aspecto adicional de la invención proporciona un medio de almacenamiento legible por ordenador de acuerdo con la reivindicación 3 adjunta.

De acuerdo con la invención, se proporciona un flujo de bits codificado de acuerdo con la reivindicación 4 adjunta.

Descripción detallada

Ejemplos de sistemas y métodos de acuerdo con la invención y con ejemplos útiles para comprender la invención se describirán ahora con referencia a los dibujos adjuntos, en los que:-

la Figura 1 muestra un esquema de alto nivel de un proceso de codificación;
la Figura 2 muestra un esquema de alto nivel de un proceso de decodificación;
la Figura 3 muestra un esquema de alto nivel de un proceso de creación de datos de restos;
la Figura 4 muestra un esquema de alto nivel de un proceso de creación de datos de restos adicionales a un nivel diferente de calidad;
la Figura 5 muestra un esquema de alto nivel de un proceso de reconstrucción de un cuadro a partir de datos de restos;
la Figura 6 muestra una estructura de datos jerárquica;
la Figura 7 muestra una estructura de datos jerárquica adicional;
la Figura 8 muestra un proceso de codificación de ejemplo;
la Figura 9 muestra un proceso de decodificación de ejemplo;
la Figura 10 muestra una estructura de un primer símbolo de datos;
la Figura 11 muestra una estructura de un segundo símbolo de datos;
la Figura 12 muestra una estructura de un símbolo de serie;
la Figura 13 muestra una estructura de un primer encabezado de flujo;
la Figura 14 muestra una estructura de un segundo encabezado de flujo;
la Figura 15 muestra una estructura de un tercer encabezado de flujo;
la Figura 16 muestra una estructura de un cuarto encabezado de flujo;
las Figuras 17A-17E muestran árboles de Huffman; y,
la Figura 18 muestra una máquina de estado de codificación de longitud de serie.

La presente invención se refiere a métodos. En particular, la presente invención se refiere a métodos para decodificar señales. El procesamiento de datos puede incluir, pero sin limitación, obtener, derivar, emitir, recibir y reconstruir datos.

La tecnología de codificación analizada en el presente documento es un formato de codificación flexible, adaptable, altamente eficiente y computacionalmente económico que combina un formato de codificación de vídeo, un códec base (por ejemplo, AVC, HEVC o cualquier otro códec presente o futuro) con un nivel de mejora de datos codificados, que se han codificado usando una técnica diferente.

La tecnología usa una señal de origen submuestreada codificada usando un códec base para formar un flujo base. Se forma un flujo de mejora usando un conjunto codificado de restos que corrigen o mejoran el flujo base, por ejemplo, aumentando la resolución o aumentando la tasa de cuadros. Puede haber múltiples niveles de datos de mejora en una estructura jerárquica. Vale la pena señalar que normalmente se espera que el flujo base sea decodificable por un decodificador de hardware mientras que se espera que el flujo de mejora sea adecuado para la implementación de procesamiento de software con un consumo de potencia adecuado.

Se necesitan métodos y sistemas para transmitir y almacenar de manera eficiente la información codificada de mejora.

Es importante que cualquier operación de codificación por entropía usada en la nueva tecnología de codificación se adapte a los requisitos o restricciones específicos del flujo de mejora y sea de baja complejidad. Tales requisitos o restricciones incluyen: la reducción potencial en la capacidad computacional resultante de la necesidad de decodificación de software del flujo de mejora; la necesidad de combinación de un conjunto decodificado de restos con un cuadro decodificado; la estructura probable de los datos de restos, es decir, la proporción relativamente alta

de valores cero con valores de datos altamente variables en un intervalo grande; los matices del bloque cuantificado de entrada de coeficientes; y siendo la estructura del flujo de mejora un conjunto de cuadros restos discretos separados en planos, capas, etc. La codificación por entropía debe también ser adecuada para múltiples niveles de mejora dentro del flujo de mejora.

La investigación realizada por los inventores ha establecido que es poco probable que los esquemas de codificación por entropía contemporáneos usados en vídeo, tales como codificación aritmética binaria adaptativa basada en contexto (CABAC) o codificación de longitud variable adaptativa al contexto (CAVLC), sean apropiados. Por ejemplo, los mecanismos predictivos pueden ser innecesarios o pueden no producir suficientes beneficios para compensar sus cargas computacionales dada la estructura de los datos de entrada. Es más, la codificación aritmética es generalmente costosa desde el punto de vista computacional y la implementación en software a menudo no es deseable.

Obsérvese que las restricciones colocadas en el flujo de mejora significan que una operación de codificación por entropía simple y rápida es esencial para posibilitar que el flujo de mejora corrija o mejore eficazmente cuadros individuales del vídeo decodificado de base. Obsérvese que en algunos escenarios el flujo base se está decodificando también sustancialmente simultáneamente antes de la combinación, ejerciendo presión sobre los recursos.

Este presente documento cumple preferiblemente los requisitos de los siguientes documentos de ISO/IEC: "Call for Proposals for Low Complexity Video Coding Enhancements" ISO/IEC JTC1/SC29/WG11 N17944, Macao, China, Oct. 2018 y "Requirements for Low Complexity Video Coding Enhancements" ISO/IEC JTC1/SC29/WG11 N18098, Macao, China, octubre de 2018. Además, los enfoques descritos en el presente documento pueden incorporarse en los productos PERSEUS® suministrados por V-Nova International Ltd.

La estructura general del esquema de codificación propuesto en el que se pueden aplicar las técnicas descritas actualmente, usa una señal de origen submuestreada codificada con un códec base, agrega un primer nivel de datos de corrección a la salida decodificada del códec base para generar una señal corregida instantánea y, a continuación, añade un nivel adicional de datos de mejora a una versión muestreada de forma ascendente de la instantánea corregida.

Por lo tanto, los flujos se consideran que son un flujo base y un flujo de mejora. Vale la pena señalar que normalmente se espera que el flujo base sea decodificable por un decodificador de hardware mientras que se espera que el flujo de mejora sea adecuado para la implementación de procesamiento de software con un consumo de potencia adecuado.

Esta estructura crea una pluralidad de grados de libertad que permiten una gran flexibilidad y adaptabilidad a muchas situaciones, haciendo así que el formato de codificación sea adecuado para muchos casos de uso que incluyen transmisión Over-The-Top (OTT), transmisión en directo, ultra alta definición (UHD) en directo difusión, y así sucesivamente.

Aunque la salida decodificada del códec base no está destinada a visualización, es un vídeo completamente decodificado a una resolución más baja, haciendo que la salida sea compatible con decodificadores existentes y, cuando se considere adecuado, también utilizable como una salida de resolución más baja.

En general, los restos se refieren a una diferencia entre un valor de una matriz de referencia o cuadro de referencia y una matriz o cuadro real de datos. Debería observarse que este ejemplo generalizado es agnóstico en cuanto a las operaciones de codificación realizadas y la naturaleza de la señal de entrada. La referencia a "datos de restos" como se usa en el presente documento se refiere a datos derivados de un conjunto de restos, por ejemplo un conjunto de restos en sí mismos o una salida de un conjunto de operaciones de procesamiento de datos que se realizan en el conjunto de restos.

En ciertos ejemplos descritos en el presente documento, puede generarse un número de flujos codificados y enviarse independientemente a un decodificador. Es decir, en métodos de ejemplo de codificación de una señal, la señal puede codificarse usando al menos dos niveles de codificación. Un primer nivel de codificación puede realizarse usando un primer algoritmo de codificación y un segundo nivel puede codificarse usando un segundo algoritmo de codificación. El método puede comprender: obtener una primera porción de un flujo de bits codificando el primer nivel de la señal; obtener una segunda porción de un flujo de bits codificando el segundo nivel de la señal; y enviar la primera porción del flujo de bits y la segunda porción del flujo de bytes como dos flujos de bits independientes.

Cabe señalar que la técnica de codificación por entropía propuesta en el presente documento no se limita a los múltiples LoQ propuestos en las Figuras, sino que proporciona utilidad en cualquier dato de resto utilizado para proporcionar mejora o corrección a un cuadro de un vídeo reconstruido a partir de un flujo codificado usando un legado tecnología de codificación de vídeo tal como HEVC. Por ejemplo, la utilidad de la técnica de codificación en la codificación de datos de restos de diferentes LoQ es particularmente beneficiosa.

Obsérvese que proporcionamos ejemplos de codificación y decodificación en las Figuras 1 a 5 de esquemas en los que las técnicas de codificación por entropía proporcionadas en el presente documento pueden proporcionar utilidad, pero se entenderá que la técnica de codificación puede utilizarse generalmente para codificar datos de restos.

Ciertos ejemplos descritos en el presente documento se refieren a un proceso de codificación y decodificación generalizado que proporciona una tecnología de codificación flexible escalable, jerárquica. La primera porción de un flujo de bits o primer flujo de bits independiente puede decodificarse usando un primer algoritmo de decodificación, y la segunda porción del flujo de bits o flujo de bits segundo o independiente puede decodificarse usando un segundo algoritmo de decodificación. El primer algoritmo de decodificación es capaz de decodificarse por un decodificador heredado usando hardware heredado.

Volviendo al proceso inicial descrito anteriormente que proporciona un flujo base y dos niveles de mejora dentro de un flujo de mejora, se representa un ejemplo de un proceso de codificación generalizado en el diagrama de bloques de la Figura 1. Un vídeo de entrada de resolución completa 100 se procesa para generar diversos flujos codificados 101, 102, 103. Un primer flujo codificado (flujo base codificado) se produce alimentando un códec base (por ejemplo, AVC, HEVC o cualquier otro códec) con una versión submuestreada del vídeo de entrada. El flujo base codificado puede denominarse como la capa base o nivel base. Se produce un segundo flujo codificado (flujo de nivel 1 codificado) procesando los restos obtenidos tomando la diferencia entre el vídeo de códec base reconstruido y la versión submuestreada del vídeo de entrada. Se produce un tercer flujo codificado (flujo de nivel 0 codificado) procesando los restos obtenidos tomando la diferencia entre una versión muestreada de manera ascendente de una versión corregida del vídeo codificado en base reconstruido y el vídeo de entrada.

Puede aplicarse una operación de submuestreo al vídeo de entrada para producir un vídeo submuestreado a codificar por un códec base. El submuestreo se puede realizar tanto en dirección vertical como horizontal o, como alternativa, solo en la dirección horizontal.

Cada proceso de codificación de flujo de mejora puede no incluir necesariamente una etapa de sobremuestreo. En la Figura 1, por ejemplo, el primer flujo de mejora es conceptualmente un flujo de corrección mientras que el segundo flujo de mejora se muestrea de manera ascendente para proporcionar un nivel de mejora.

Al observar el proceso de generación de los flujos de mejora con más detalle, para generar el flujo de nivel 1 codificado, el flujo base codificado se decodifica 114 (es decir, se aplica una operación de decodificación al flujo base codificado para generar un flujo base decodificado). La diferencia entre el flujo base decodificado y el vídeo de entrada submuestreado se crea a continuación 110 (es decir, se aplica una operación de resta al vídeo de entrada submuestreado y al flujo base decodificado para generar un primer conjunto de restos).

En este caso, el término restos se usa de la misma manera que la conocida en la técnica, es decir, el error entre un cuadro de referencia y un cuadro deseado. En este caso, el cuadro de referencia es el flujo base decodificado y el cuadro deseado es el vídeo de entrada submuestreado. Por lo tanto, los restos usados en el primer nivel de mejora pueden considerarse como un vídeo corregido ya que 'corrigen' el flujo base decodificado al vídeo de entrada submuestreado que se usó en la operación de codificación base.

De nuevo, la operación de codificación por entropía descrita más adelante es adecuada para cualquier dato de resto, por ejemplo cualquier dato asociado con un conjunto de restos.

A continuación, la diferencia se codifica 115 para generar el flujo de nivel 1 codificado 102 (es decir, se aplica una operación de codificación al primer conjunto de restos para generar un primer flujo de mejora).

Como se ha indicado anteriormente, el flujo de mejora puede comprender un primer nivel de mejora 102 y un segundo nivel de mejora 103. El primer nivel de mejora 102 puede considerarse que es un flujo corregido. El segundo nivel de mejora 103 puede considerarse que es un nivel adicional de mejora que convierte el flujo corregido al vídeo de entrada original.

El nivel adicional de mejora 103 se crea codificando un conjunto adicional de restos que son la diferencia 119 entre una versión 117 muestreada ascendentemente de un flujo de nivel 1 118 decodificado y el vídeo 100 de entrada.

Como se ha indicado, un flujo muestreado ascendentemente se compara con el vídeo de entrada que crea un conjunto adicional de restos (es decir, se aplica una operación de diferencia al flujo recreado muestreado ascendentemente para generar un conjunto adicional de restos). El conjunto adicional de restos se codifica a continuación 121 como el flujo de mejora de nivel 0 codificado (es decir, a continuación se aplica una operación de codificación al conjunto adicional de restos para generar un flujo de mejora adicional codificado).

Por lo tanto, como se ilustra en la Figura 1 y se ha descrito anteriormente, la salida del proceso de codificación es un flujo base 101 y uno o más flujos de mejora 102, 103 que comprenden preferiblemente un primer nivel de mejora y un nivel adicional de mejora.

Un correspondiente proceso de decodificación generalizada se representa en el diagrama de bloques de la Figura 2. El decodificador recibe los tres flujos 101, 102, 103 generados por el codificador junto con encabezamientos que contienen información de decodificación adicional. El flujo base codificado se decodifica por un decodificador base

correspondiente al códec base usado en el codificador, y su salida se combina con los restos decodificados obtenidos del flujo de nivel 1 codificado. El vídeo combinado se muestrea de manera ascendente y se combina adicionalmente con los restos decodificados obtenidos a partir del flujo de nivel 0 codificado.

En el proceso de decodificación, el decodificador puede analizar los encabezados (configuración global, configuración de imagen, bloque de datos) y configurar el decodificador basándose en esos encabezados. Para recrear el vídeo de entrada, el decodificador puede decodificar cada uno del flujo base, el primer flujo de mejora y el flujo de mejora adicional. Los cuadros del flujo pueden sincronizarse y luego combinarse para derivar el vídeo decodificado.

En cada una de las Figuras 1 y 2, las operaciones de codificación de nivel 0 y nivel 1 pueden incluir las etapas de transformación, cuantificación y codificación por entropía. De manera similar, en la etapa de decodificación, los restos pueden pasarse a través de un decodificador por entropía, un descuantificador y un módulo de transformada inversa. Puede usarse cualquier codificación adecuada y operación de decodificación correspondiente. Sin embargo, preferiblemente, las etapas de codificación de nivel 0 y nivel 1 pueden realizarse en software.

Resumiendo, los métodos y aparatos en el presente documento se basan en un algoritmo general que se construye sobre un algoritmo de codificación y/o decodificación existente (tal como normas MPEG tales como AVC/H.264, HEVC/H.265, etc., así como algoritmo no estándar tal como VP9, AV1 y otros) que funciona como una línea de base para una capa de mejora que funciona de acuerdo con un algoritmo de codificación y/o decodificación diferente. La idea detrás del algoritmo global es codificar/decodificar jerárquicamente el cuadro de vídeo en oposición al uso de enfoques basados en bloques como se usa en la familia de algoritmos de MPEG. Codificar jerárquicamente un cuadro incluye generar restos para todo el cuadro, y a continuación un cuadro diezmado y así sucesivamente.

Los datos de restos de compresión de vídeo para el cuadro de vídeo de tamaño completo pueden denominarse LoQ-0 (por ejemplo, 1920 x 1080 para un cuadro de vídeo HD), mientras que los del cuadro diezmado pueden denominarse LoQ-x, donde x indica el número de decimaciones jerárquicas. En los ejemplos descritos de las Figuras 1 y 2, la variable x tiene un valor máximo de 1 y, por lo tanto, hay 2 niveles jerárquicos para los que se generarán restos de compresión.

La Figura 3 ilustra un ejemplo de cómo podría generarse LoQ-1 en un dispositivo de codificación. En la presente Figura, el algoritmo y los métodos generales se describen usando un algoritmo de codificación/decodificación AVC/H.264 como algoritmo de línea de base, pero se entiende que pueden usarse otros algoritmos de codificación/decodificación como algoritmos de línea de base sin ningún impacto en la forma en que el funciona el algoritmo.

Por supuesto, se entenderá que los bloques de la Figura 3 son meramente ejemplos de cómo podrían implementarse los conceptos generales.

El diagrama de la Figura 3 muestra el proceso de generación de datos de restos codificados por entropía para el nivel de jerarquía LoQ-1. En el ejemplo, la primera etapa es diezmar el vídeo sin comprimir entrante por un factor de 2. Esta cuadro diezmado se pasa a continuación a través de un algoritmo de codificación de base (en este caso, un algoritmo de codificación de AVC/H.264) donde a continuación se genera y almacena un cuadro de referencia codificado por entropía. A continuación, se genera una versión decodificada del cuadro de referencia codificado, y la diferencia entre el cuadro de referencia decodificado y el cuadro diezmado (restos de LoQ-1) formará la entrada al bloque de transformada.

La transformada (por ejemplo, una transformada basada en Hadamard en este ejemplo ilustrado) convierte esta diferencia en 4 componentes (o planos), en concreto A (promedio), H (horizontal), V (vertical) y D (diagonal). Estos componentes se cuantifican a continuación mediante el uso de variables denominadas 'anchuras escalonadas' (por ejemplo, un valor resto puede dividirse por la anchura de etapa y el valor entero más cercano seleccionado). Un proceso de codificación por entropía adecuado es el objeto de esta divulgación y se describe en detalle a continuación. Estos restos cuantificados se codifican a continuación por entropía para eliminar cualquier información redundante. Los coeficientes o componentes codificados cuantificados (Ae, He, Ve y De) se colocan a continuación dentro de un flujo en serie con paquetes de definición insertados al inicio del flujo, esta etapa final se logra usando una rutina de serialización de archivos. Los datos de paquetes pueden incluir tal información tal como la especificación del codificador, el tipo de sobremuestreo a emplear, si se descartan o no los planos A y D, y otra información para posibilitar que un decodificador decodifique los flujos.

Tanto los datos de referencia (cuadro codificado por entropía de línea de base de tamaño medio) como los datos de restos de LoQ-1 codificados por entropía pueden almacenarse en memoria intermedia, transmitirse o almacenarse para su uso por el decodificador durante el proceso de reconstrucción.

Con el fin de producir datos de restos de LoQ-0, la salida cuantificada se ramifica y se realizan procesos de cuantificación y transformación inversa en la misma para reconstruir los restos de LoQ-1, que a continuación se añaden a los datos de referencia decodificados (codificados y decodificados) para recuperar un cuadro de vídeo que se asemeje mucho al cuadro de entrada diezmado originalmente.

Este proceso idealmente imita el proceso de decodificación y, por lo tanto, no se usa el cuadro diezmado originalmente.

La Figura 4 ilustra un ejemplo de cómo podría generarse LoQ-0 en un dispositivo de codificación. Para derivar los restos de LoQ-0, el cuadro dimensionado de LoQ-1 reconstruido se deriva como se describe en la sección anterior.

La siguiente etapa es realizar un muestreo ascendente del cuadro reconstruido a tamaño completo (en 2). En este punto, pueden usarse diversos algoritmos para mejorar el proceso de muestreo ascendente, tales como algoritmos más cercanos, bilineales, precisos o cúbicos. Este cuadro de tamaño completo reconstruido denominado cuadro 'previsto' se resta a continuación de la entrada de vídeo sin comprimir original que crea algunos restos (restos de LoQ-1).

Similar al proceso de LoQ-1, la diferencia se transforma a continuación, se cuantifica, se codifica por entropía y se serializa el archivo que a continuación forma los datos terceros y finales. Esto puede almacenarse en memoria intermedia, transmitirse o almacenarse para su uso posterior por el decodificador. Como puede observarse, puede derivarse un componente denominado "promedio previsto" (descrito a continuación) usando el proceso de sobremuestreo, y usarse en lugar del componente A (promedio) para mejorar adicionalmente la eficiencia del algoritmo de codificación.

La Figura 5 muestra esquemáticamente cómo podría realizarse el proceso de decodificación en un ejemplo particular. Los datos codificados por entropía, los datos de restos codificados por entropía de LOQ-1 y los datos de restos codificados por entropía de LOQ-0 (por ejemplo, como datos codificados serializados en archivos). Los datos codificados por entropía incluyen la base codificada de tamaño reducido (por ejemplo, tamaño medio, es decir con dimensiones W/2 y H/2 con respecto al cuadro completo que tiene dimensiones W y H).

Los datos codificados por entropía se decodifican a continuación usando el algoritmo de decodificación correspondiente al algoritmo que se ha usado para codificar esos datos (en el ejemplo, un algoritmo de decodificación AVC/H.264). Al final de esta etapa, se produce un cuadro de vídeo decodificado, que tiene un tamaño reducido (por ejemplo, medio tamaño) (indicado en el presente ejemplo como un vídeo AVC/H.264).

En paralelo, se decodifican los datos de restos codificados por entropía de LoQ-1. Como se ha analizado anteriormente, los restos de LoQ-1 se codifican usando cuatro coeficientes o componentes (A, V, H y D) que, como se muestra en esta Figura, tienen una dimensión de un cuarto de la dimensión de cuadro completo, en concreto, W/4 y H/4. Esto se debe a que, como se analiza en las solicitudes de patente anteriores US 13/893.669 y PCT/EP2013/059847, los cuatro componentes contienen toda la información relativa a los restos y se generan aplicando un núcleo de transformada 2x2 a los restos cuya dimensión, para LoQ-1, sería W/2 y H/2, es decir, la misma dimensión de los datos codificados por entropía de tamaño reducido. Los cuatro componentes se decodifican por entropía, a continuación se descuantifican y finalmente se transforman de nuevo en los restos de LoQ-1 originales usando una transformada inversa (en este caso, una transformada inversa de Hadamard 2x2).

Los restos de LoQ-1 decodificados se añaden a continuación al cuadro de vídeo decodificado para producir un cuadro de vídeo reconstruido a tamaño reducido (en este caso, tamaño medio), identificado como reconstrucción de tamaño medio 2D.

Este cuadro de vídeo reconstruido se muestrea a continuación para llevarlo a la resolución completa (por lo tanto, en este ejemplo, desde la mitad de la anchura (W/2) y la mitad de la altura (H/2) hasta la anchura completa (W) y la altura completa (H)) usando un filtro de muestreo ascendente tal como bilineal, bicúbico, preciso, etc. El cuadro de vídeo reconstruido muestreado ascendentemente será un cuadro previsto (tamaño completo, W x H) al que se añaden a continuación los restos decodificados de LoQ-0.

En particular, se decodifican los datos de restos codificados de LoQ-0. Como se ha analizado anteriormente, los restos de LoQ-0 se codifican usando cuatro coeficientes o componentes (A, V, H y D) que, como se muestra en esta Figura, tienen una dimensión de la mitad de la dimensión de cuadro completo, en concreto, W/2 y H/2. Esto se debe a que, como se analiza en las solicitudes de patente anteriores US 13/893.669 y PCT/EP2013/059847, los cuatro componentes contienen toda la información relativa a los restos y se generan aplicando un núcleo de transformada 2x2 a los restos cuya dimensión, para LoQ-0, sería W y H, es decir, la misma dimensión del cuadro completo. Los cuatro componentes se decodifican por entropía (véase el proceso a continuación), a continuación se descuantifican y finalmente se transforman de nuevo en los restos de LoQ-0 originales usando una transformada inversa (en este caso, una transformada inversa de Hadamard 2x2).

Los restos de LoQ-0 decodificados se añaden a continuación al cuadro previsto para producir un cuadro de vídeo completo reconstruido - el cuadro de salida.

La estructura de datos se representa de una manera ilustrativa en la Figura 6. Como se ha analizado, la descripción anterior se ha hecho con referencia a tamaños específicos y algoritmos de línea de base, pero los métodos anteriores se aplican a otros tamaños y/o algoritmos de línea de base, y la descripción anterior se proporciona únicamente a

modo de ejemplo de los conceptos más generales descritos.

En el algoritmo de codificación/decodificación descrito anteriormente, normalmente hay 3 planos (por ejemplo, YUV o RGB), con dos niveles de calidades LoQ que se describen como LoQ-0 (o nivel superior, resolución completa) y LoQ-1 (o menor nivel, resolución de tamaño reducido tal como media resolución) en cada plano. Cada plano puede representar un componente de color diferente para una señal de vídeo.

Cada LoQ contiene cuatro componentes o capas, en concreto, A, H, V y D. Esto da un total de 3x2x4 = 24 superficies de las cuales 12 son de tamaño completo (por ejemplo, WxH) y 12 son de tamaño reducido (por ejemplo, W/2xH/2). Cada capa puede comprender valores de coeficiente para uno particular de los componentes, por ejemplo la capa A puede comprender un valor de coeficiente A superior izquierdo para cada bloque de 2x2 de la imagen o cuadro de entrada.

La Figura 7 ilustra una vista alternativa de una estructura de datos jerárquica propuesta.

Los datos codificados se separan en fragmentos. Cada carga útil se ordena jerárquicamente en fragmentos. Es decir, cada carga útil se agrupa en planos, a continuación dentro de cada plano cada nivel se agrupa en capas y cada capa comprende un conjunto de fragmentos para esa capa. El nivel representa cada nivel de mejora (primero o adicional) y capa representa un conjunto de coeficientes de transformada.

El método puede comprender recuperar fragmentos para dos niveles de mejora para cada plano. El método puede comprender recuperar 16 capas para cada nivel (por ejemplo, si se usa una transformada 4x4). Por tanto, cada carga útil se ordena en un conjunto de fragmentos para todas las capas en cada nivel y a continuación el conjunto de fragmentos para todas las capas en el siguiente nivel del plano. A continuación, la carga útil comprende el conjunto de fragmentos para las capas del primer nivel del siguiente plano y así sucesivamente.

Por tanto, el método puede decodificar encabezados y emitir coeficientes codificados por entropía agrupados por plano, niveles y capas que pertenecen a la mejora de imagen que se está decodificando. Por tanto, la salida de acuerdo con la invención es (nPlanos)x(nNivel)x(nCapa) superficies de matriz con elementos superficies[nPlanos][nNivel][nCapa].

Obsérvese que las técnicas de codificación por entropía proporcionadas en el presente documento pueden realizarse en cada grupo, es decir, pueden realizarse por superficie, por plano, por nivel (LoQ) o por capa. Obsérvese que la codificación por entropía puede realizarse en cualquier dato de resto y no necesariamente en coeficientes cuantificados y/o transformados.

Como se muestra en las Figuras 4 y 5, las operaciones de codificación y decodificación propuestas incluyen una etapa u operación de codificación por entropía. Se propone a continuación que la operación de codificación por entropía comprenda o bien un componente de codificación de longitud de serie o un componente de codificación de longitud de serie (RLE) y un componente de codificación Huffman. Estos dos componentes pueden interrelacionarse para proporcionar beneficios adicionales.

Como se observa y como se ilustra en la Figura 8, la entrada del codificador por entropía es una superficie (por ejemplo, datos de restos derivados de un conjunto cuantificado de restos como se ilustra en este ejemplo) y la salida del proceso es una versión codificada por entropía de los restos (Ae, He, Ve, De). Sin embargo, como anteriormente, debería observarse que la codificación por entropía puede realizarse en cualquier dato de resto y no necesariamente en coeficientes cuantificados y/o transformados. La Figura 9 ilustra un decodificador de alto nivel correspondiente con entradas y salidas inversas. Es decir, el decodificador por entropía toma como entradas restos codificados por entropía (Ae, He, Ve, De) y emite datos de restos (por ejemplo, restos cuantificados en este ejemplo ilustrado).

Obsérvese que la Figura 8 introduce en general la codificación RLE antes de la codificación Huffman. En aspectos que no caen dentro del alcance de la invención reivindicada, se observa que la codificación de longitud de serie puede proporcionarse sin codificación Huffman y, en casos comparativos, la codificación de longitud de serie y la codificación Huffman pueden no realizarse (ya sea reemplazada por codificación de entropía sin pérdidas alternativa o no codificación por entropía realizada en absoluto). Por ejemplo, en un ejemplo de producción con un énfasis reducido en la compresión de datos, una tubería de codificación puede no comprender codificación por entropía, pero los beneficios de los restos pueden residir en el almacenamiento en capas para la distribución.

Un codificador de longitud de serie comparativa (RLE) puede comprimir datos codificando secuencias del mismo valor de datos como un único valor de datos y un recuento de ese valor de datos. Por ejemplo, la secuencia 555500022 puede codificarse como (4,5)(3,0)(2,2). Es decir, hay una serie de cuatro 5s, una serie de tres 0s seguida de una serie de dos 2s.

Para codificar los datos de restos se propone una operación de codificación RLE modificada. Para aprovechar la estructura de los datos de restos, se propone codificar únicamente series de ceros. Es decir, cada valor se envía como un valor con cada cero se envía como una serie de ceros. Las operaciones de codificación RLE modificadas descritas

en el presente documento pueden usarse, por ejemplo, para proporcionar codificación y/o decodificación por entropía en una o más capas de LoQ, por ejemplo como se muestra en las Figuras 1 a 5.

Por lo tanto, la operación de codificación por entropía comprende analizar los datos de restos y codificar valores cero como una serie de ceros consecutivos.

Como se ha indicado anteriormente, para proporcionar un nivel adicional de compresión de datos (que puede ser sin pérdidas), la operación de codificación por entropía aplica adicionalmente una operación de codificación Huffman a los datos codificados de longitud de serie.

La codificación Huffman y la codificación de longitud de serie se han emparejado previamente juntas, por ejemplo, en la codificación de facsímil (por ejemplo, las recomendaciones de ITU T.4 y T.45) y el formato de intercambio de archivo JPEG, pero se han reemplazado durante los últimos 30 años. La presente descripción propone técnicas y métodos para implementar codificación Huffman en combinación con codificación RLE, técnicas y métodos para intercambiar datos y metadatos de manera eficiente entre un codificador y decodificador, y técnicas y métodos para reducir el tamaño de datos general de tal combinación cuando se usa para codificar datos de restos.

Un código de Huffman es un código de prefijo óptimo que se usa para compresión de datos (por ejemplo, compresión sin pérdidas). Un código de prefijo es un sistema de código para el que no hay trabajo de código en el sistema que es el prefijo de cualquier otro código. Es decir, una operación de codificación Huffman toma un conjunto de símbolos de entrada y convierte cada símbolo de entrada en un código correspondiente. El código elegido se basa en la frecuencia con la que aparece cada símbolo en el conjunto de datos original. De esta manera, pueden asociarse códigos más pequeños con símbolos que se producen con más frecuencia para reducir el tamaño global de los datos.

Para facilitar la codificación Huffman, los datos de entrada se estructuran preferiblemente como símbolos. La salida de la operación de codificación de longitud de serie se estructura preferiblemente para ser un flujo de bytes de datos codificados.

En un ejemplo, la operación de codificación de longitud de serie emite dos tipos de bytes o símbolos. Un primer tipo de símbolo es un valor de un píxel distinto de cero y un segundo tipo de símbolo es una serie de valores cero consecutivos. Es decir, una cantidad de valores cero que se producen consecutivamente en el conjunto de datos original.

En un ejemplo adicional, para codificar ciertos valores de datos, pueden combinarse dos bytes o símbolos. Es decir, por ejemplo, cuando un valor de píxel es mayor que un valor umbral, pueden usarse dos bytes o símbolos para codificar el valor de datos en un símbolo. Estos dos símbolos se pueden combinar en el decodificador para recrear el valor de datos original del píxel.

Cada símbolo o byte puede incluir 6 o 7 bits de datos y uno o más indicadores o bits que indican el tipo de símbolo.

En un ejemplo de implementación, los datos codificados de longitud de serie pueden codificarse eficazmente insertando, en cada símbolo, una o más indicadores que indican el siguiente símbolo que se espera en el flujo de bytes.

Para facilitar la sincronización entre el codificador y el decodificador, el primer byte del flujo de datos codificados de longitud de serie puede ser un tipo de símbolo de valor de datos. Este byte puede indicar entonces el siguiente tipo de byte que se ha codificado.

En ciertas realizaciones, el indicador que indica el siguiente byte puede ser diferente dependiendo del tipo de símbolo. Por ejemplo, el indicador o bit puede ubicarse al principio o al final del byte o ambos. Por ejemplo, al comienzo del byte, el bit puede indicar que el siguiente símbolo puede ser una serie de ceros. Al final del byte, el bit puede ser un bit de desbordamiento, que indica que el siguiente símbolo contiene una parte del valor de datos a combinar con el valor de datos en el símbolo actual.

Lo siguiente describe un ejemplo de implementación específico de una operación de codificación de longitud de serie. El RLE tiene tres contextos: RLC_RESTO_LSB, RLC_RESTO_MSB y RLC_SERIE_CERO. La estructura de estos bytes se ilustra en las Figuras 10, 11 y 12.

En un primer tipo de símbolo, RLC_RESTO_LSB, ilustrado en la Figura 10, se codifican los 6 bits menos significativos de un píxel distinto de cero. Puede proporcionarse un bit de serie que indica que el siguiente byte está codificando el recuento de una serie de ceros. Se puede establecer un bit de desbordamiento si el valor de píxel no se ajusta dentro de 6 bits de datos. Cuando se establece el bit de desbordamiento, el contexto del siguiente byte será un tipo RLC_RESTO_MSB. Es decir, el siguiente símbolo comprenderá un byte de datos que puede combinarse con los bits del símbolo actual para codificar un valor de datos. Cuando se establece el bit de desbordamiento, el siguiente contexto no puede ser una serie de ceros y, por lo tanto, el símbolo puede usarse para codificar datos.

La Figura 10 indica un ejemplo de este tipo de símbolo. Si el valor de datos a codificar es mayor que un valor umbral de 64, o el valor de píxel no se ajusta dentro de 6 bits de datos, el bit de desbordamiento puede establecerse por el proceso de codificación. Si se establece un bit de desbordamiento como el bit menos significativo del byte, entonces los bits restantes del byte pueden codificar datos. Si el valor de píxel se ajusta dentro de 6 bits, el bit de desbordamiento puede no establecerse y puede incluirse un bit de serie que indica si el siguiente símbolo es o no un valor de datos o una serie de ceros.

Un segundo tipo de símbolo, RLC_RESTO_MSB, ilustrado en la Figura 11, codifica los bits 7 a 13 de valores de píxel que no encajan dentro de 6 bits de datos. El bit 7 de este tipo de símbolo codifica si el siguiente byte es o no una serie de ceros.

Un tercer tipo de símbolo, RLC_SERIE_CERO, ilustrado en la Figura 12, codifica 7 bits de un recuento de serie cero. Es decir, el símbolo comprende el número de ceros consecutivos en los datos de restos. El bit de serie del símbolo es alto si se necesitan más bits para codificar el recuento. El bit de serie puede ser el bit más significativo del byte. Es decir, cuando hay un número de ceros consecutivos que requiere más de los 7 bits disponibles, digamos 178 o 255, el bit de serie indica que el siguiente bit indicará una serie adicional de ceros.

Opcionalmente, el símbolo adicional puede comprender una segunda serie de ceros que se puede combinar con la primera serie o puede comprender un conjunto de bits de un valor que se puede combinar con los bits del primer símbolo en el decodificador para indicar el recuento.

Como se ha indicado anteriormente, para que el decodificador comience en un contexto conocido, el primer símbolo en el flujo codificado puede ser un tipo de símbolo de valor de datos resto, es decir, puede ser un tipo RLC_RESTO_LSB.

En un ejemplo específico, los datos RLE pueden organizarse en bloques. Cada bloque puede tener una capacidad de salida de 4096 bytes. RLE puede conmutar a un nuevo bloque en los siguientes casos:

- el bloque actual está lleno;
- los datos RLE actuales son una serie y quedan menos de 5 bytes en el bloque actual; y/o
- los datos RLE actuales se conducen a un par de LSB/MSB y quedan menos de 2 bytes en el bloque actual.

Resumiendo, puede realizarse una operación de codificación de longitud de serie en los datos de restos de la nueva tecnología de codificación que comprende codificar en un flujo un conjunto de valores de datos y un recuento de valores cero consecutivos. En una implementación específica, la salida de la operación de codificación de longitud de serie puede ser un flujo de bytes o símbolos, donde cada byte es uno de tres tipos o contextos. El byte indica el siguiente tipo de byte que se espera en el flujo de bytes.

Debería observarse que estas estructuras se proporcionan como un ejemplo, y que pueden aplicarse diferentes esquemas de codificación de bits mientras se siguen las enseñanzas funcionales descritas en el presente documento. Por ejemplo, puede intercambiarse la codificación de los bits menos y más significativos y/o pueden promulgarse diferentes longitudes de bits. También, los bits de indicador pueden ubicarse en diferentes posiciones predefinidas dentro de un byte.

Como se ha indicado anteriormente, puede aplicarse una operación de codificación Huffman al flujo de bytes para reducir adicionalmente el tamaño de los datos. El proceso puede crear una tabla de frecuencia de la aparición relativa de cada símbolo en el flujo. A partir de la tabla de frecuencia, el proceso puede generar un árbol de Huffman. Puede generarse un código de Huffman para cada símbolo atravesando el árbol desde la raíz hasta que se alcanza el símbolo, asignando un bit del código para cada rama tomada.

En una implementación preferida para su uso con las especificaciones de los símbolos codificados de longitud de serie de los restos cuantificados (es decir, los datos de restos), puede aplicarse una operación de codificación Huffman canónica. Los códigos canónicos de Huffman pueden reducir los requisitos de almacenamiento de un conjunto de códigos dependiendo de la estructura de los datos de entrada. Un procedimiento de Huffman canónico proporciona una manera de generar un código que contiene implícitamente la información de qué palabra de código se aplica a qué símbolo. Los códigos de Huffman pueden establecerse con un conjunto de condiciones, por ejemplo, todos los códigos de una longitud dada pueden tener valores lexicográficamente consecutivos en el mismo orden que los símbolos que representan y los códigos más cortos preceden lexicográficamente a los códigos más largos. En una implementación de codificación Huffman canónica, únicamente necesitan transmitirse las palabras de código y las longitudes de cada código para que el decodificador pueda replicar cada símbolo.

Después de aplicar la operación de codificación Huffman canónica, el tamaño de datos de los datos de salida puede compararse con el tamaño de los datos después de la operación de codificación de longitud de serie. Si el tamaño de datos es menor, entonces los datos más pequeños pueden transmitirse o almacenarse. La comparación de tamaño de datos puede realizarse basándose en el tamaño de un bloque de datos, un tamaño de una capa, plano o superficie o el tamaño total del cuadro o vídeo. Para señalizar al decodificador cómo se codificaron los datos, de acuerdo con la

invención se transmite un indicador en un encabezado de los datos de que los datos se codifican usando codificación Huffman, codificación RLE o ambas. En una implementación alternativa que no cae dentro del alcance de la invención reivindicada, el decodificador puede ser capaz de identificar a partir de características de los datos que los datos se han codificado usando una operación de codificación particular. Por ejemplo, como se indica a continuación, los datos pueden dividirse en un encabezado y parte de datos donde se usa codificación Huffman canónica para señalizar las longitudes de código de símbolos codificados.

Los datos codificados de Huffman canónicos comprenden una parte que señaliza al decodificador las longitudes del código usado para cada símbolo y una parte que comprende un flujo de bits que representan los datos codificados. En una implementación específica propuesta en el presente documento, las longitudes de código pueden señalizarse en una parte de encabezado y los datos señalizarse en una parte de datos. Preferiblemente, se señalizará una parte de encabezado para cada superficie, pero se puede señalizar para cada bloque u otra subdivisión dependiendo de la configuración.

En un ejemplo propuesto, el encabezado puede ser diferente dependiendo de la cantidad de códigos distintos de cero a codificar. La cantidad de códigos distintos de cero a codificar (para cada superficie o bloque, por ejemplo) puede compararse con un valor umbral y un encabezado usado basándose en la comparación. Por ejemplo, cuando hay más de 31 códigos distintos de cero, el encabezado puede indicar secuencialmente todos los símbolos, comenzando por una señal predeterminada tal como 0. En orden, puede señalizarse la longitud de cada símbolo. Cuando hay menos de 31 códigos distintos de cero, cada valor de símbolo puede señalizarse en el encabezado con una longitud de código correspondiente para ese símbolo.

Las Figuras 13 a 16 ilustran una implementación específica de los formatos de encabezado y cómo las longitudes de código pueden escribirse en un encabezado de flujo dependiendo de la cantidad de códigos distintos de cero.

La Figura 13 ilustra una situación en la que puede haber más de 31 valores distintos de cero en los datos. El encabezado incluye una longitud de código mínima y una longitud de código máxima. La longitud de código para cada símbolo se envía secuencialmente. Un indicador indica que la longitud del símbolo es distinta de cero. Los bits de la longitud de código se envían a continuación como una diferencia entre la longitud de código y la longitud señalizada mínima. Esto reduce el tamaño total del encabezado.

La Figura 14 ilustra un encabezado similar a la Figura 13 pero usado donde hay menos de 31 códigos distintos de cero. El encabezado incluye adicionalmente el número de símbolos en los datos, seguido por un valor de símbolo y la longitud de la palabra de código para ese símbolo, enviado de nuevo como una diferencia.

Las Figuras 15 y 16 ilustran encabezados adicionales a enviarse en casos periféricos. Por ejemplo, cuando las frecuencias son todas cero, el encabezado de flujo puede ser como se ilustra en la Figura 14 que indica dos valores de 31 en los campos de longitud mínima y máxima para indicar una situación especial. Cuando solo hay un código en el árbol de Huffman, el encabezado de flujo puede ser como se indica en la Figura 16 usando un valor 0 en los campos de longitud mínima y máxima para indicar una situación especial y luego seguido por el valor de símbolo a usar.

En este último ejemplo, donde solo hay un valor de símbolo, esto puede indicar que solo hay un valor de datos en los datos de restos.

Por lo tanto, el proceso de codificación se puede resumir de la siguiente manera -

Analizar datos de restos para identificar valores de datos y recuentos consecutivos de valores cero.
Generar un conjunto de símbolos donde cada símbolo es un byte y cada byte comprende un valor de datos o una serie de ceros así como una indicación del siguiente símbolo en el conjunto de símbolos. Los símbolos pueden incluir un bit de desbordamiento que indica que el siguiente símbolo incluye una parte del valor de datos incluido en el símbolo actual o un bit de serie que indica que el siguiente símbolo es una serie de ceros.
Convertir cada símbolo de longitud fija en un código variable usando codificación canónica de Huffman. La codificación canónica de Huffman analiza los símbolos para identificar la frecuencia con la que se produce cada símbolo en el conjunto de símbolos y asigna un código al símbolo basándose en la frecuencia y el valor del símbolo (para longitudes de código idénticas).
Generar y emitir un conjunto de longitudes de código, cada una asociada con un símbolo. Estando las longitudes de código que son la longitud de cada código asignadas a cada símbolo.
Combinar los códigos de longitud variable en un flujo de bits.
Emitir el flujo de bits codificado.

En la implementación específica descrita, se observa que la longitud de código máxima depende del número de símbolos codificados y del número de muestras usadas para derivar la tabla de frecuencia de Huffman. Para N símbolos, la longitud de código máxima es N-1. Sin embargo, para que un símbolo tenga un código Huffman de k bits, el número de muestras usadas para derivar la tabla de frecuencia necesita ser al menos:

$$S_k = \sum_{i=0}^{k+1} F_i$$

donde F_i es el $i^{ésimo}$ número de Fibonacci.

Por ejemplo, símbolos de 8 bits, la longitud de código máxima teórica es 255. Sin embargo, para un vídeo de HD 1080p, el número máximo de muestras usadas para derivar la tabla de frecuencia para un contexto de RLE de un resto es 1920*1080/4 = 518 400, lo que está entre S26 y S27. Por lo tanto, ningún símbolo puede tener un código de Huffman mayor que 26 bits para este ejemplo. Para 4K este número aumenta a 29 bits.

Para completar, usando la Figura 17, proporcionamos un ejemplo de codificación Huffman de un conjunto de símbolos, en este ejemplo los símbolos son alfanuméricos. Como se ha indicado anteriormente, un código de Huffman es un código de prefijo óptimo que puede usarse para compresión de datos sin pérdidas. Un código de prefijo es un sistema de código para el que no hay palabra de código en el sistema que sea el prefijo de cualquier otro código.

Para encontrar un código de Huffman para un conjunto dado de símbolos, es necesario crear un árbol de Huffman. En primer lugar, los símbolos se ordenan por frecuencia, por ejemplo:

Símbolo	Frecuencia
A	3
B	8
C	10
D	15
E	20
F	43

Los dos elementos más bajos se eliminan de la lista y se convierten en hojas, con un nodo padre que tiene una frecuencia la suma de las frecuencias de los dos elementos más bajos. El árbol parcial se ilustra en la Figura 17a.

La nueva lista de frecuencias ordenada es:

Símbolo	Frecuencia
C	10
*	11
D	15
E	20
F	43

A continuación, se repite el bucle, combinando los dos elementos más bajos como se ilustra en la Figura 17b.

La nueva lista es:

Símbolo	Frecuencia
D	15
E	20
*	21
F	43

Esto se repite hasta que solo queda un elemento en la lista como se ilustra en las Figuras 17c, 17d, 17e y las siguientes tablas correspondientes.

Símbolo	Frecuencia
*	21
*	35
F	43

Símbolo	Frecuencia
F	43
*	56

Símbolo	Frecuencia
*	99

Una vez que se construye el árbol, para generar el código de Huffman para un símbolo, el árbol se atraviesa desde la raíz hasta este símbolo, emitiendo un 0 cada vez que se toma una rama izquierda y un 1 cada vez que se toma una rama derecha. En el ejemplo anterior, esto da el siguiente código:

Símbolo	Código	Longitud de código
A	1010	3
B	1011	3
C	100	2
D	110	2
E	111	2
F	0	0

La longitud de código de un símbolo es la longitud de su correspondiente código.

Para decodificar un código de Huffman, el árbol se atraviesa comenzando en la raíz, tomando una trayectoria izquierda si se lee un 0 y una trayectoria derecha si se lee un 1. El símbolo se encuentra al golpear una hoja.

Como se ha descrito anteriormente, los símbolos de RLE pueden codificarse usando codificación Huffman. En un ejemplo preferido, se usa codificación canónica de Huffman. En una implementación de codificación Huffman canónica, puede usarse un procedimiento de codificación Huffman y el código producido transformarse en códigos de Huffman canónicos. Es decir, la tabla se reorganiza de modo que longitudes idénticas son secuenciales en el mismo orden que los símbolos que representan y que un código posterior siempre tendrá un valor mayor que uno anterior. Cuando los símbolos son alfabéticos, pueden reorganizarse en orden alfabético. Cuando los símbolos son valores, pueden estar en orden de valor secuencial y las palabras de código correspondientes cambiarse para cumplir con las restricciones anteriores. La codificación canónica de Huffman y/u otros enfoques de codificación Huffman descritos en el presente documento pueden diferir del ejemplo básico de las Figuras 17a a 17e.

En los ejemplos descritos anteriormente, puede aplicarse una operación de codificación Huffman a símbolos de datos codificados usando una operación de codificación de longitud de serie modificada. Se propone adicionalmente crear una tabla de frecuencia para cada contexto o estado de RLE. Para cada tipo de símbolo codificado por la operación de codificación RLE, puede haber un conjunto diferente de códigos de Huffman. En una implementación, puede haber un codificador de Huffman diferente para cada contexto o estado de RLE.

Se contempla que puede enviarse un encabezado de flujo para cada contexto de RLE o tipo de símbolo, es decir, una pluralidad de encabezados de flujo puede indicar las longitudes de código para cada conjunto de códigos, es decir, para cada codificador de Huffman o cada tabla de frecuencia.

Lo siguiente describe un ejemplo de etapas de implementación específicas del codificador. En una implementación, la codificación de los datos emitidos por la RLE se realiza en tres etapas principales:

1. Inicializar los codificadores (por ejemplo, mediante una función *InicializarCodificación*):

    a. Inicializar cada codificador (uno por estado de RLE) con la tabla de frecuencia correspondiente generada por

el RLE;
b. Crear el árbol de Huffman;
c. Calcular la longitud de código de cada símbolo; y
d. Determinar las longitudes de código mínima y máxima.

2. Escribir la longitud de código y tabla de códigos en el encabezado de flujo (por ejemplo, mediante una función *EscribirLongitudesCódigo* función) para cada codificador:

a. Asignar códigos a los símbolos (por ejemplo, usando una función *AsignarCódigos*). Obsérvese que la función usada para asignar códigos a símbolo es ligeramente diferente del ejemplo de las Figuras 19a a 19e anteriores, ya que asegura que todos los códigos de una longitud dada son secuenciales, es decir, usa codificación Huffman canónica;
b. Escribir las longitudes de código mínima y máxima en el encabezado de flujo; y
c. Escribir las longitudes de código de cada símbolo en el flujo.

3. Codificar los datos RLE:

a. Establecer contexto actual de RLE a RLC_RESTO_LSB;
b. Codificar el símbolo actual en el flujo de entrada con el codificador correspondiente al contexto actual;
c. Usar la máquina de estado de RLE para obtener el siguiente contexto (véase Figura 20). Si este no es el último símbolo del flujo, ir a b; y
d. Si los datos RLE son solo un bloque y el flujo codificado es más grande que RLE, almacenar el flujo codificado por RLE en lugar del flujo codificado por Huffman.

Para cada bloque de RLE, los codificadores de Huffman crean una parte codificada de Huffman correspondiente.

La salida del proceso de codificación descrito en el presente documento es, por lo tanto, un flujo de bits codificado por Huffman o un flujo de bytes codificado por longitud de serie. Un decodificador correspondiente emplea por tanto una operación de decodificación Huffman correspondiente de una manera inversa a la operación de codificación. Sin embargo, hay matices a la operación de decodificación descrita a continuación para mejorar la eficiencia y para adaptarse a la especificidad de la operación de codificación descrita anteriormente.

Cuando el codificador señaliza en un encabezado u otros metadatos de configuración que se ha usado una operación de codificación RLE o Huffman, el decodificador de acuerdo con la invención reivindicada identifica en primer lugar que debería aplicarse una operación de decodificación Huffman al flujo. De manera similar, en la forma descrita anteriormente, el decodificador en una disposición que no cae bajo el alcance de la invención reivindicada puede identificar a partir del flujo de datos si se ha usado o no codificación Huffman identificando el número de partes del flujo. Si hay una parte de encabezado y una parte de datos, entonces debería aplicarse una operación de decodificación Huffman. Si solo hay una parte de datos en el flujo, entonces solo se puede aplicar una parte de decodificación RLE.

El decodificador por entropía realiza las transformaciones inversas del decodificador. En el presente ejemplo, decodificación Huffman seguida de decodificación de longitud de serie.

El siguiente ejemplo describe un proceso de decodificación en el que el flujo de bits se ha codificado usando una operación de codificación Huffman canónica y una operación de codificación de longitud de serie como se ha descrito anteriormente en el ejemplo del flujo de codificación. Sin embargo, se entenderá que las operaciones de codificación Huffman y las operaciones de codificación de longitud de serie pueden aplicarse por separado para decodificar flujos codificados de una manera diferente.

La entrada al proceso es un flujo de bits codificado. El flujo de bits puede recuperarse de un archivo almacenado o transmitirse, ya sea localmente o remotamente. El flujo de bits codificado es una serie de bits secuenciales sin una estructura inmediatamente discernible. Solo una vez que se aplica la operación de codificación Huffman al flujo de bits puede deducirse una serie de símbolos.

Por lo tanto, el proceso aplica en primer lugar una operación de codificación Huffman al flujo de bits.

Se supone que el primer símbolo codificado es un tipo de símbolo de valor de datos. De esta manera, cuando la operación de codificación Huffman usa diferentes tablas o parámetros de frecuencia, la operación de codificación Huffman puede usar un libro de códigos correcto para un tipo de símbolo.

La operación de codificación Huffman canónica debe recuperar en primer lugar parámetros de codificación o metadatos adecuados para facilitar la decodificación del flujo de bits. En el presente ejemplo, los parámetros son un conjunto de longitudes de código recuperadas de un encabezado de flujo. Por simplicidad, describiremos un conjunto de longitudes de código que se intercambian entre el codificador y el decodificador, pero tenga en cuenta que, como anteriormente, se pueden señalizar múltiples conjuntos de longitudes de código para que la operación de codificación

pueda usar diferentes parámetros para cada tipo de símbolo que espera estar decodificando.

La operación de codificación asignará las longitudes de código recibidas en el encabezado de flujo a un símbolo correspondiente. Como se ha indicado anteriormente, el encabezado de flujo puede enviarse en diferentes tipos. Cuando el encabezado de flujo incluye un conjunto de longitudes de código con símbolos correspondientes, cada longitud de código puede asociarse con un valor de símbolo correspondiente. Cuando las longitudes de código se envían secuencialmente, el decodificador puede asociar cada longitud de código recibida con un símbolo en un orden predeterminado. Por ejemplo, las longitudes de código pueden recuperarse en un orden 3, 4, 6, 2, etc. El proceso de codificación puede a continuación asociar cada longitud con un símbolo correspondiente. Aquí el orden es secuencial - (símbolo, longitud) - (0,3)(1,4)(3,6)(4,2).

Como se ilustra en la Figura 15, algunos símbolos pueden no enviarse, pero un indicador puede indicar que el símbolo tiene una longitud de código cero correspondiente, es decir, que el símbolo no existe en el flujo de bits a decodificar (o no se usa).

La operación de codificación Huffman canónica es entonces capaz de asignar un código a cada símbolo basándose en la longitud de código. Por ejemplo, donde la longitud de código es 2, el código para ese símbolo será '1x'. Donde x indica que el valor no es importante. En implementaciones específicas, el código será '10'.

Cuando las longitudes son idénticas, la operación de codificación canónica de Huffman asigna códigos basándose en un orden secuencial de los símbolos. Cada código se asigna de tal manera que a medida que se analiza el flujo de bits, el decodificador puede seguir comprobando el siguiente bit hasta que solo existe un código posible. Por ejemplo, a un primer símbolo secuencial de longitud 4 se le puede asignar un código 1110 y a un segundo símbolo secuencial de longitud 4 se le puede asignar un código 1111. Por tanto, analizando el flujo de bits, si los primeros bits del flujo de bits son 1110x, solo es posible que el flujo de bits haya codificado el primer símbolo secuencial de longitud 4.

Por consiguiente, una vez que la operación de codificación ha establecido un conjunto de códigos asociados con cada símbolo, la operación de codificación puede pasar a analizar el flujo de bits. En ciertas realizaciones, la operación de codificación construirá un árbol para establecer el símbolo asociado con el código del flujo de bits. Es decir, la operación de codificación tomará cada bit del flujo de bits y atravesará un árbol de acuerdo con el valor del bit en el flujo de bits hasta que se encuentre una hoja. Una vez que se encuentra una hoja, se emite el símbolo asociado con la hoja. El proceso continúa en la raíz del árbol con el siguiente bit del flujo de bits. De esta manera, la operación de codificación puede emitir un conjunto de símbolos derivados de un conjunto de códigos almacenados en el flujo de bits codificado. En realizaciones preferidas, los símbolos son cada uno un byte, como se describe en otra parte en el presente documento.

El conjunto de símbolos puede pasarse a continuación a una operación de codificación de longitud de serie. La operación de codificación de longitud de serie identifica el tipo de símbolo y analiza el símbolo para extraer un valor de datos o una serie de ceros. A partir de los valores de datos y series de ceros, la operación puede recrear los datos de restos originales que se codificaron.

La salida de la codificación Huffman es, como se ha indicado anteriormente, probablemente un conjunto de símbolos. El desafío para la siguiente etapa de decodificación es extraer de esos símbolos la información relevante, observando que cada símbolo comprende datos en un formato diferente y cada símbolo representará información diferente a pesar de que la información no sea inmediatamente discernible del símbolo o byte.

En ejemplos preferidos, la operación de codificación consulta una máquina de estado como se ilustra en la Figura 18. En resumen, la operación de codificación supone en primer lugar que el primer símbolo es un tipo de valor de datos (el valor de datos puede ser, por supuesto, 0). A partir de esto, la operación de codificación puede identificar si el byte incluye un indicador de desbordamiento o un indicador de serie. El indicador de desbordamiento y serie se describe anteriormente y se ilustra en la Figura 10.

La operación de codificación puede comprobar en primer lugar el bit menos significativo del símbolo (o byte). Aquí, este es el indicador (o bit) de desbordamiento. Si se establece el bit menos significativo, la operación de codificación identifica que el siguiente símbolo será también un valor de datos y por del tipo RLC_RESTO_MSB. Los bits restantes del símbolo serán bits menos significativos de un valor de datos.

El proceso de codificación continúa para analizar el siguiente símbolo y extraer los bits menos significativos como la parte restante del valor de datos. Los bits del primer símbolo se pueden combinar con los bits del segundo símbolo para reconstruir el valor de datos.

En este símbolo actual, ilustrado en la Figura 11, habrá también un indicador de serie, en este caso el bit más significativo. Si se establece esto, el siguiente símbolo será un símbolo de serie y no un símbolo de datos.

En el símbolo de serie, ilustrado en la Figura 12, la operación de codificación comprueba el bit más significativo para indicar si el siguiente símbolo es un símbolo de serie o un símbolo de datos y extrae los bits restantes como la serie

de ceros. Es decir, una serie máxima de 127 ceros.

Si el símbolo de datos indica que no hay desbordamiento (en el indicador de desbordamiento, aquí el bit menos significativo), entonces el símbolo también contendrá en el bit más significativo un bit de serie y la operación de codificación podrá identificar a partir de este bit que el siguiente el símbolo en el conjunto es un símbolo de serie o un símbolo de datos. La operación de codificación puede extraer, por lo tanto, el valor de datos de los bits 6-1. Aquí hay 31 valores de datos disponibles sin desbordamiento.

Como se ha indicado, la máquina de estado de la Figura 18 ilustra el proceso de manera sencilla.

Comenzando en un símbolo RLC_RESTO_LSB, si el bit de desbordamiento = 0 y el bit de serie = 0, entonces el siguiente símbolo será un símbolo RLC_RESTO_LSB. Si el bit de desbordamiento = 1, entonces el siguiente símbolo será un RLC_RESTO_MSB. Si el bit de desbordamiento = 1, no habrá un bit de serie. Si el bit de serie = 1 y el bit de desbordamiento = 0, entonces el siguiente bit será un símbolo RLC_SERIE_CERO.

En el RLC RLC_RESTO_MSB, si el bit de serie = 0, el siguiente símbolo será un símbolo RLC_RESTO_LSB. Si el bit de serie = 1, el siguiente símbolo será un símbolo RLC_SERIE_CERO.

En el símbolo RLC_SERIE_CERO, si el bit de serie = 0, el siguiente bit será un símbolo RLC_RESTO_LSB. Si el bit de serie = 1, el siguiente bit será un símbolo RLC_SERIE_CERO.

Por supuesto, los bits pueden invertirse (0/1, 1/0, etc.) sin pérdida de funcionalidad. De manera similar, las ubicaciones dentro de los símbolos o bytes de las indicadores son meramente ilustrativas.

La operación de codificación de longitud de serie puede identificar el siguiente símbolo en el conjunto de símbolos y extraer un valor de datos o una serie de ceros. La operación de codificación puede combinar a continuación estos valores y ceros para recrear los datos de restos. El orden puede estar en el orden extraído o, como alternativa, en algún orden predeterminado.

Por lo tanto, la operación de codificación puede emitir datos de restos que se han codificado en el flujo de bytes.

Se ha descrito anteriormente cómo pueden usarse múltiples metadatos o parámetros de codificación en el proceso de codificación. En el lado de decodificación, la operación de codificación puede incluir una retroalimentación entre las operaciones. Es decir, el siguiente símbolo esperado puede extraerse del símbolo actual (por ejemplo, usando los bits de desbordamiento y serie) y el siguiente símbolo esperado dado a la operación de codificación Huffman.

La operación de codificación Huffman puede generar opcionalmente un libro de códigos separado para cada tipo de símbolo (y recuperar múltiples encabezados de flujo o construir una tabla de múltiples códigos y símbolos).

Suponiendo que el primer símbolo será un valor de datos, la operación de codificación Huffman decodificará el primer código para que coincida con un símbolo de ese tipo. A partir del símbolo coincidente, la operación de codificación puede identificar un bit de desbordamiento y/o un bit de serie e identificar el siguiente tipo de símbolo. La operación de codificación Huffman puede usar a continuación un libro de códigos correspondiente para ese tipo de símbolo para extraer el siguiente código en el flujo de bits. El proceso lleva de esta manera iterativa usando el símbolo actualmente decodificado para extraer una indicación del siguiente símbolo y cambiar los libros de códigos en consecuencia.

Se ha descrito anteriormente cómo una operación de decodificación puede combinar una operación de codificación Huffman, preferiblemente una operación de codificación Huffman canónica, y una operación de codificación de longitud de serie para recrear datos de restos a partir de un flujo de bits codificado. Se entenderá que las técnicas de la operación de codificación de longitud de serie pueden aplicarse a un flujo de bytes que no se ha codificado usando una operación de codificación Huffman. De manera similar, la operación de codificación Huffman puede aplicarse sin la etapa posterior de la operación de codificación de longitud de serie.

Lo siguiente describe un ejemplo de etapas de implementación específicas del decodificador. En la implementación de ejemplo, si el flujo tiene más de una parte, se realizan las siguientes etapas:

1. Leer las longitudes de código del encabezado de flujo (por ejemplo, usando una función *LeerLongitudesCódigo*):

    a. Establecer las longitudes de código para cada símbolo;
    b. Asignar códigos a símbolos a partir de las longitudes de código (por ejemplo, usando una función *AsignarCódigos*); y
    c. Generar una tabla para buscar los subconjuntos de códigos con longitudes idénticas. Cada elemento de la tabla registra el primer índice de una longitud dada y el código correspondiente (*primerIdx, primerCódigo*).

2. Decodificar los datos RLE:

a. Establecer el contexto de RLE en RLC_RESTO_LSB.
b. Decodificar el código actual buscando la longitud de código correcta en la tabla generada e indexando en matriz de código con: *primerIdx* - (*código_actual* - *primerCódigo*). Esto funciona porque todos los códigos para una longitud dada son secuenciales por construcción del árbol de Huffman.
c. Usar la máquina de estado de RLE para obtener el siguiente contexto (véase Figura 18). Si el flujo no está vacío, ir a b.

El decodificador de longitud de serie lee los datos codificados de longitud de serie byte a byte. Por construcción, se garantiza que el contexto del primer byte de datos es RLC_RESTO_LSB. El decodificador usa la máquina de estado mostrada en la Figura 18 para determinar el contexto del siguiente byte de datos. El contexto le dice al decodificador cómo interpretar el byte actual de datos como se ha descrito anteriormente.

Obsérvese que en este ejemplo de implementación específico la máquina de estado de longitud de serie también se usa por el proceso de codificación y decodificación Huffman paren concreto qué código de Huffman usar para el símbolo o palabra de código actual.

Tanto en el codificador como en el decodificador, por ejemplo implementado en un servidor de transmisión en continuo o dispositivo cliente o decodificación de dispositivo cliente a partir de un almacén de datos, los métodos y procesos descritos en el presente documento pueden incorporarse como código (por ejemplo, código de software) y/o datos. El codificador y el decodificador pueden implementarse en hardware o software como es bien conocido en la técnica de la compresión de datos. Por ejemplo, la aceleración de hardware usando una Unidad de Procesamiento Gráfico (GPU) específicamente programada o una Matriz de Puertas Programables en Campo (FPGA) específicamente diseñada puede proporcionar ciertas eficiencias. Para completar, tal código y datos pueden almacenarse en uno o más medios legibles por ordenador, que pueden incluir cualquier dispositivo o medio que pueda almacenar código y/o datos para su uso por un sistema informático. Cuando un sistema informático lee y ejecuta el código y/o datos almacenados en un medio legible por ordenador, el sistema informático realiza los métodos y procesos incorporados como estructuras de datos y código almacenados dentro del medio de almacenamiento legible por ordenador. En ciertas realizaciones, una o más de las etapas de los métodos y procesos descritos en el presente documento pueden realizarse por un procesador (por ejemplo, un procesador de un sistema informático o sistema de almacenamiento de datos).

En general, cualquiera de las funcionalidades descritas en este texto o ilustradas en las figuras puede implementarse usando software, firmware (por ejemplo, circuitería de lógica fija), hardware programable o no programable, o una combinación de estas implementaciones. Los términos "componente" o "función" como se usan en el presente documento representan por lo general software, firmware, hardware o una combinación de estos. Por ejemplo, en el caso de una implementación de software, los términos "componente" o "función" pueden referirse a código de programa que realiza tareas especificadas cuando se ejecuta en un dispositivo o dispositivos de procesamiento. La separación ilustrada de componentes y funciones en unidades distintas puede reflejar cualquier agrupación física real o conceptual y asignación de tal software y/o hardware y tareas.

En la presente solicitud, hemos descrito un método para codificar y decodificar una señal, en particular una señal de vídeo y/o una señal de imagen.

En particular, se describe un método de codificación de una señal, comprendiendo el método recibir un cuadro de entrada y procesar el cuadro de entrada para generar al menos un primer conjunto de datos de restos, permitiendo dichos datos de restos que un decodificador reconstruya el cuadro original a partir de una referencia marco reconstruido.

En un ejemplo, el método comprende obtener el cuadro reconstruido a partir de un cuadro decodificado obtenido desde un módulo de decodificación, en donde el módulo de decodificación está configurado para generar dicho cuadro decodificado decodificando un primer cuadro codificado que se ha codificado de acuerdo con un primer método de codificación. El método comprende además submuestrear el cuadro de entrada para obtener un cuadro submuestreado, y pasar dicho cuadro submuestreado a un módulo de codificación configurado para codificar dicho cuadro submuestreado de acuerdo con el primer método de codificación para generar el primer cuadro codificado. Obtener el cuadro reconstruido puede comprender adicionalmente sobremuestrear el cuadro decodificado para generar el cuadro reconstruido.

En otro ejemplo, el método comprende obtener el cuadro reconstruido a partir de una combinación de un segundo conjunto de datos de restos y un cuadro decodificado obtenido desde un módulo de decodificación, en donde el módulo de decodificación está configurado para generar dicho cuadro decodificado decodificando un primer cuadro codificado que se ha codificado de acuerdo con un primer método de codificación. El método comprende además submuestrear el cuadro de entrada para obtener un cuadro submuestreado y pasar dicho cuadro submuestreado a un módulo de codificación configurado para codificar dicho cuadro submuestreado de acuerdo con el primer método de codificación para generar el primer cuadro codificado. El método comprende adicionalmente generar dicho segundo conjunto de datos de restos tomando una diferencia entre el cuadro decodificado y el cuadro submuestreado. El método comprende adicionalmente codificar dicho segundo conjunto de datos de restos para generar un primer conjunto de datos de restos codificados. La codificación de dicho segundo conjunto de datos de restos puede realizarse de acuerdo

con un segundo método de codificación. El segundo método de codificación comprende transformar el segundo conjunto de datos de restos en un segundo conjunto transformado de datos de restos. Transformar el segundo conjunto de datos de restos comprende seleccionar un subconjunto del segundo conjunto de datos de restos, y aplicar una transformación en dicho subconjunto para generar un subconjunto correspondiente del segundo conjunto transformado de datos de restos.

Uno del subconjunto del segundo conjunto transformado de datos de restos puede obtenerse promediando el subconjunto del segundo conjunto de datos de restos. Obtener el cuadro reconstruido puede comprender adicionalmente sobremuestrear la combinación del segundo conjunto de datos de restos y un cuadro decodificado para generar el cuadro reconstruido.

En un ejemplo, generar el al menos un conjunto de datos de restos comprende tomar una diferencia entre el cuadro reconstruido de referencia y el cuadro de entrada. El método comprende adicionalmente codificar dicho primer conjunto de datos de restos para generar un primer conjunto de datos de restos codificados. La codificación de dicho primer conjunto de datos de restos puede realizarse de acuerdo con un tercer método de codificación. El tercer método de codificación comprende transformar el primer conjunto de datos de restos en un primer conjunto transformado de datos de restos. Transformar el primer conjunto de datos de restos comprende seleccionar un subconjunto del primer conjunto de datos de restos, y aplicar una transformación en dicho subconjunto para generar un subconjunto correspondiente del primer conjunto transformado de datos de restos. Uno de los subconjuntos del primer conjunto transformado de datos de restos puede obtenerse por la diferencia entre un promedio de un subconjunto del cuadro de entrada y un elemento correspondiente de la combinación del segundo conjunto de datos de restos y el cuadro decodificado.

En particular, se describe un método de decodificación de una señal, comprendiendo el método recibir un cuadro codificado y al menos un conjunto de datos de restos codificados. El primer cuadro codificado puede codificarse usando un primer método de codificación.

El al menos un conjunto de datos de restos puede codificarse usando un segundo y/o un tercer método de codificación.

El método comprende adicionalmente pasar el primer cuadro codificado a un módulo de decodificación, en donde el módulo de decodificación está configurado para generar un cuadro decodificado decodificando el cuadro codificado que se ha codificado de acuerdo con un primer método de codificación.

El método puede comprender adicionalmente decodificar el al menos un conjunto de datos de restos codificados de acuerdo con el respectivo método de codificación usado para codificarlos.

En un ejemplo, se decodifica un primer conjunto de datos de restos codificados aplicando un segundo método de decodificación que corresponde a dicho segundo método de codificación para obtener un primer conjunto de datos de restos decodificados. El método comprende además combinar el primer conjunto de datos de restos con el cuadro decodificado para obtener un cuadro combinado. El método comprende adicionalmente sobremuestrear el cuadro combinado para obtener un cuadro decodificado de referencia.

El método comprende adicionalmente decodificar un segundo conjunto de datos de restos codificados aplicando un tercer método de decodificación correspondiente a dicho tercer método de codificación para obtener un segundo conjunto de datos de restos decodificados. El método comprende además combinar el segundo conjunto de datos de restos decodificados con el cuadro decodificado de referencia para obtener un cuadro reconstruido.

En otro ejemplo, el método comprende sobremuestrear el cuadro decodificado para obtener un cuadro decodificado de referencia.

El método comprende adicionalmente decodificar un conjunto de datos de restos codificados aplicando un segundo o tercer método de decodificación correspondiente a dicho segundo o tercer método de codificación para obtener un conjunto de datos de restos decodificados. El método comprende además combinar el conjunto de datos de restos decodificados con el cuadro decodificado de referencia para obtener un cuadro reconstruido.

REIVINDICACIONES

1. Un método de decodificar una señal de vídeo, comprendiendo el método:

recuperar un flujo de bits codificado (102, 103) ordenado jerárquicamente en fragmentos, en donde el flujo de bits se agrupa en planos y los planos se dividen en niveles, y dentro de cada plano, cada nivel se agrupa en capas y cada capa comprende un conjunto de fragmentos para esa capa, en donde cada nivel corresponde a un nivel de mejora espacial y cada capa comprende conjuntos de coeficientes del mismo tipo;
decodificar el flujo de bits codificado (102, 103) para generar datos de restos para un flujo de mejora, y,
reconstruir un cuadro original de la señal de vídeo a partir de los datos de restos y un cuadro reconstruido de referencia,
en donde la etapa de decodificar el flujo de bits codificado (102, 103) comprende, para cada fragmento:

recuperar un indicador de los metadatos de configuración que acompañan al flujo de bits codificado (102, 103) indicando si se va a omitir una operación de codificación Huffman al decodificar el fragmento;
aplicar selectivamente la operación de codificación Huffman basándose en el indicador para generar un conjunto de bytes decodificados de Huffman, en donde cada byte corresponde a un recuento de valores cero consecutivos de los datos de restos, los bits menos significativos de un valor de datos distinto de cero de los datos de restos o los bits más significativos de un valor de datos distinto de cero de los datos de restos;
aplicar una operación de codificación de longitud de serie en el conjunto de bytes decodificados de Huffman o un flujo de bytes codificado de longitud de serie del flujo de bits codificado, basándose en el indicador, para generar los datos de restos para el fragmento, en donde la operación de codificación de longitud de serie comprende:

identificar un conjunto de bytes que representa los bits menos significativos de valores de datos distintos de cero de los datos de restos, un conjunto de bytes que representa los bits más significativos de valores de datos distintos de cero de los datos de restos, y un conjunto de bytes que representa recuentos de valores cero consecutivos de los datos de restos;
analizar los bytes para derivar los valores de datos distintos de cero de los datos de restos y los recuentos de valores cero consecutivos; y,

generar los datos de restos a partir de los valores de datos distintos de cero y los recuentos de valores cero consecutivos.

2. Un aparato de decodificación configurado para realizar el método de la reivindicación 1.

3. Un medio legible por ordenador que comprende instrucciones que cuando se ejecutan por un procesador hacen que el procesador lleve a cabo el método de la reivindicación 1.

4. Un flujo de bits codificado (102, 103) que comprende una versión codificada de datos de restos para un flujo de mejora,

en donde el flujo de bits codificado se ordena jerárquicamente en fragmentos, en donde el flujo de bits se agrupa en planos y los planos se dividen en niveles, y dentro de cada plano, cada nivel se agrupa en capas y cada capa comprende un conjunto de fragmentos para esa capa, en donde cada nivel corresponde a un nivel de mejora espacial y cada capa comprende conjuntos de coeficientes del mismo tipo;
pudiendo usarse los datos de restos con un cuadro reconstruido de referencia para reconstruir un cuadro original de la señal de vídeo,
en donde el flujo de bits codificado comprende, para cada fragmento:

metadatos de configuración que comprenden un indicador que indica si una operación de codificación Huffman se va a omitir en la decodificación del fragmento; y,
basándose en el indicador, un conjunto de bytes codificados por Huffman, en donde cada byte corresponde a un resultado de aplicar una operación de codificación de longitud de serie a los datos de restos;
o, basándose en el indicador, el resultado de aplicar la operación de codificación de longitud de serie a los datos de restos,

en donde la operación de codificación de longitud de serie da como resultado un conjunto de bytes que representan los bits menos significativos de valores de datos distintos de cero de los datos de restos, un conjunto de bytes que representan los bits más significativos de valores de datos distintos de cero de los datos de restos, y un conjunto de bytes que representan recuentos de valores cero consecutivos de los datos de restos, pudiendo los conjuntos de bytes analizarse por un decodificador para derivar valores de datos distintos de cero de los datos de restos y recuentos de valores cero consecutivos, y para generar los datos de restos a partir de los datos distintos de cero valores de datos cero y los recuentos de valores cero consecutivos.

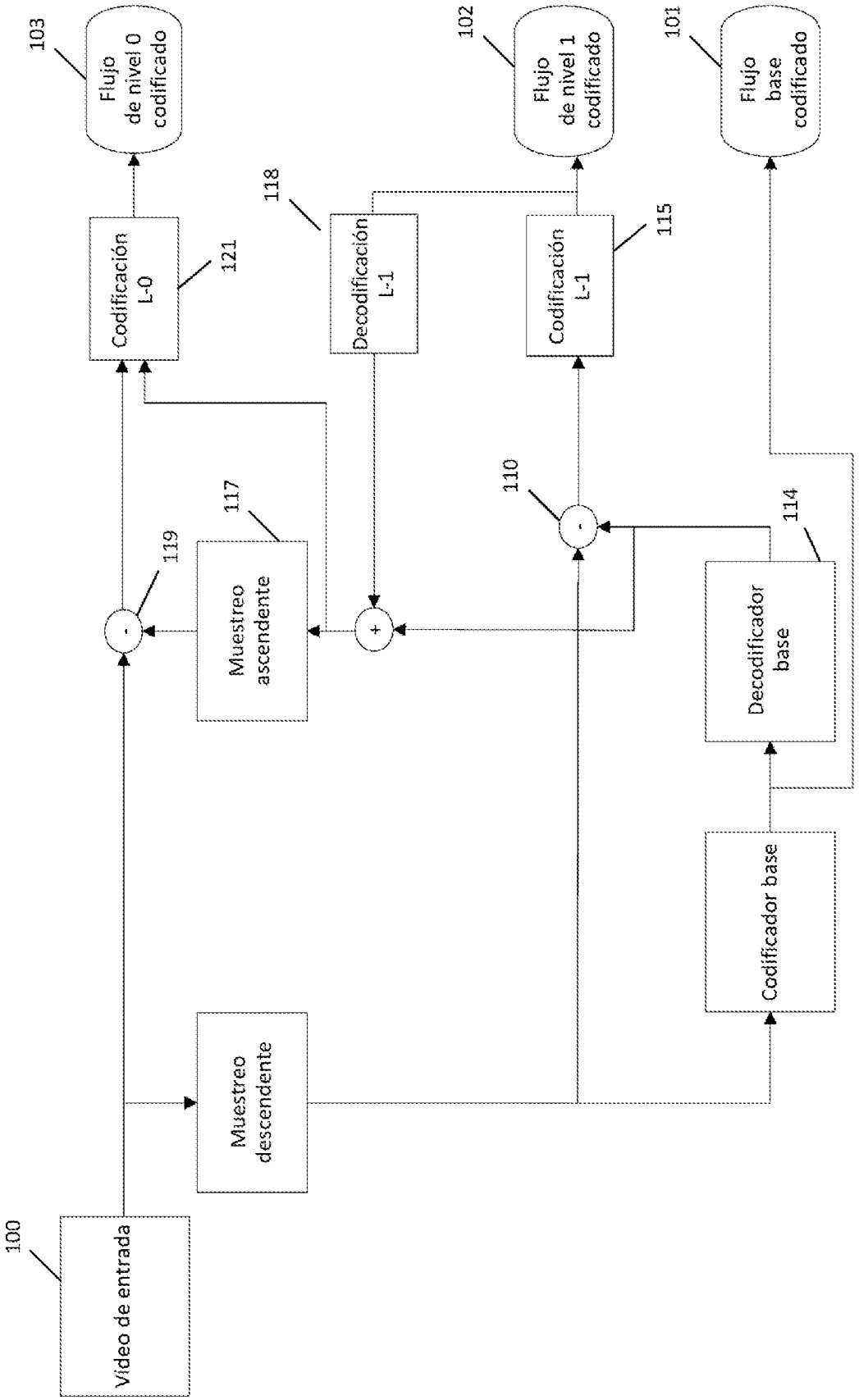
Vídeo de entrada
100
Muestreo descendente
Codificador base
Decodificador base
114
Flujo base codificado
101
Flujo de nivel 1 codificado
102
Flujo de nivel 0 codificado
103
Codificación L-1
115
Decodificación L-1
118
Codificación L-0
121
Muestreo ascendente
117
110
119
+
-
-

Figura 1

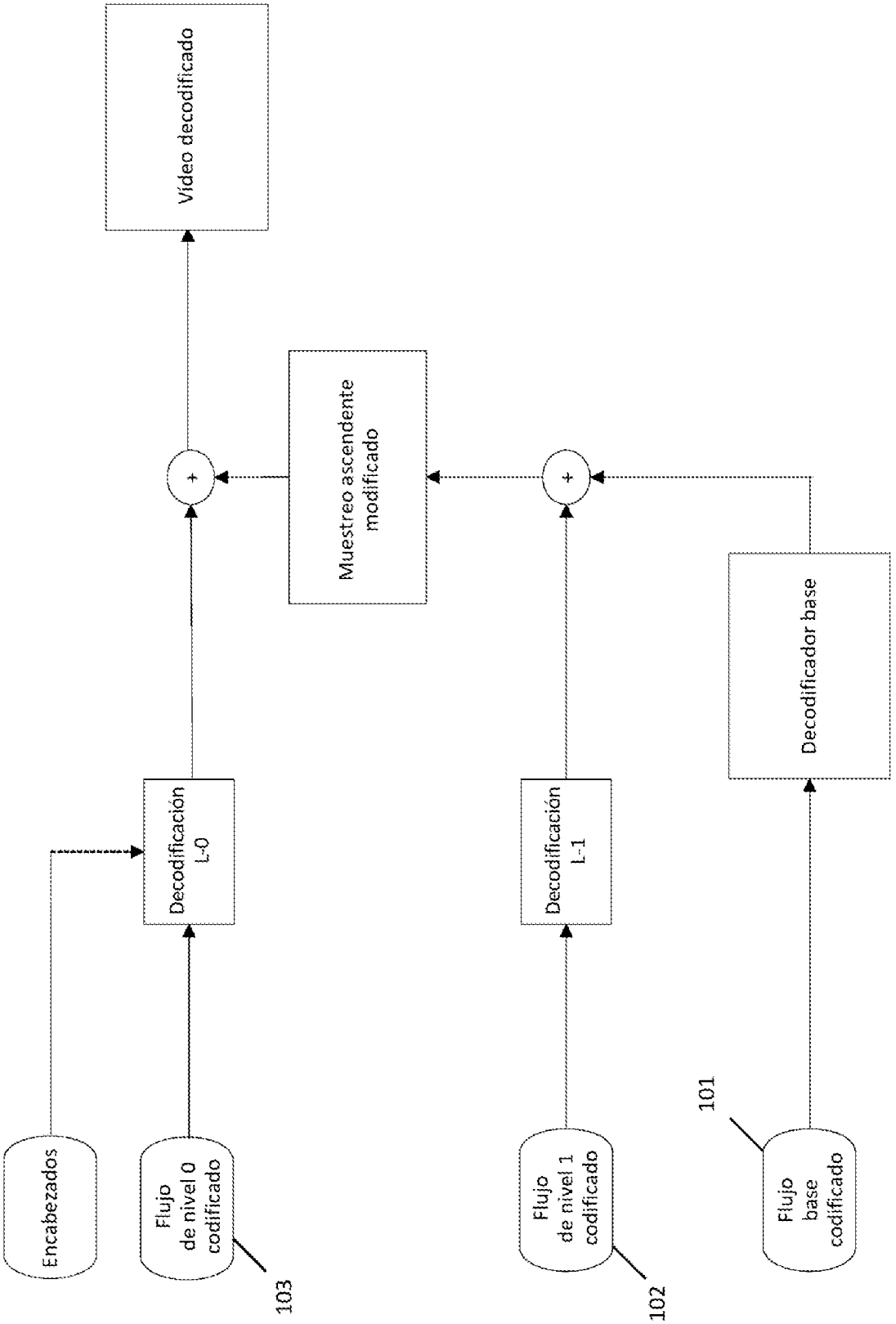
Vídeo decodificado
+
Muestreo ascendente modificado
+
Encabezados
Decodificación L-0
Decodificación L-1
Decodificador base
Flujo de nivel 0 codificado
Flujo de nivel 1 codificado
Flujo base codificado
103
102
101

Figura 2

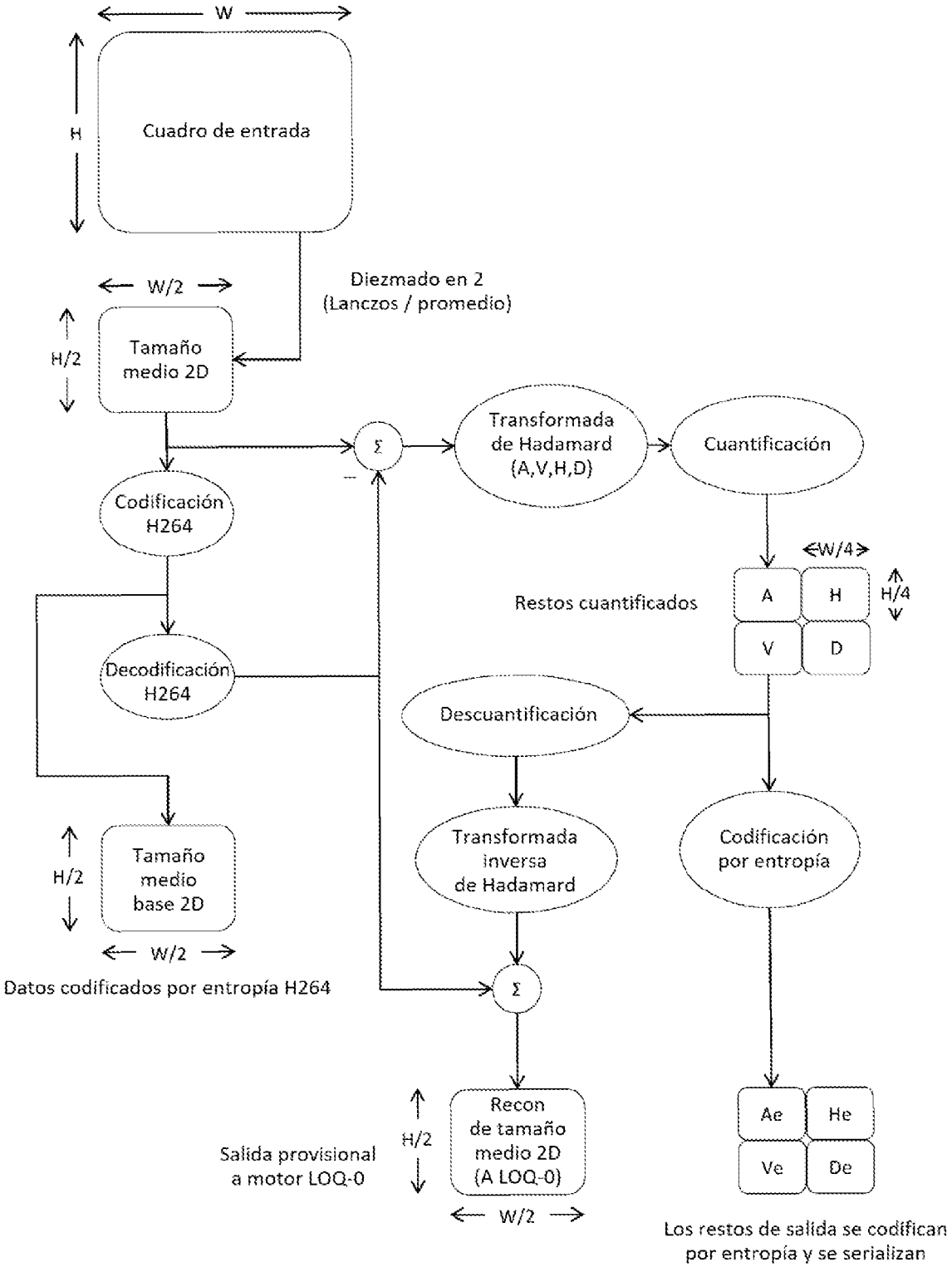
W
H
Cuadro de entrada
W/2
H/2
Tamaño medio 2D
Diezmado en 2 (Lanczos / promedio)
Σ
Transformada de Hadamard (A,V,H,D)
Cuantificación
Codificación H264
W/4
H/4
A
H
V
D
Restos cuantificados
Decodificación H264
Descuantificación
Transformada inversa de Hadamard
Codificación por entropía
Tamaño medio base 2D
Datos codificados por entropía H264
Recon de tamaño medio 2D (A LOQ-0)
Salida provisional a motor LOQ-0
Ae
He
Ve
De
Los restos de salida se codifican por entropía y se serializan

Figura 3

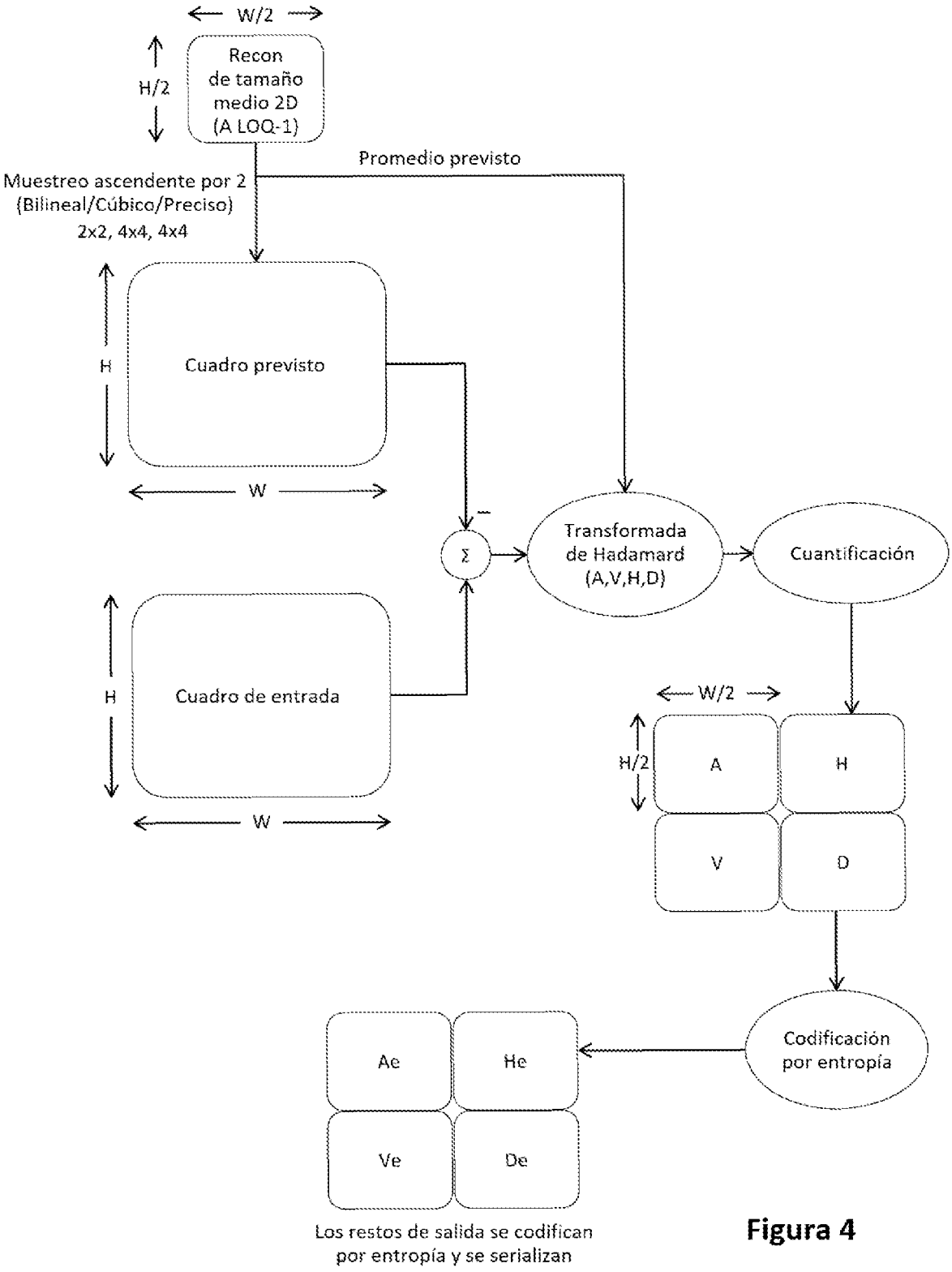
W/2
H/2
Recon
de tamaño
medio 2D
(A LOQ-1)
Promedio previsto
Muestreo ascendente por 2
(Bilineal/Cúbico/Preciso)
2x2, 4x4, 4x4
H
Cuadro previsto
W
Σ
Transformada
de Hadamard
(A,V,H,D)
Cuantificación
H
Cuadro de entrada
W
W/2
H/2
A
H
V
D
Codificación
por entropía
Ae
He
Ve
De
Los restos de salida se codifican
por entropía y se serializan

Figura 4

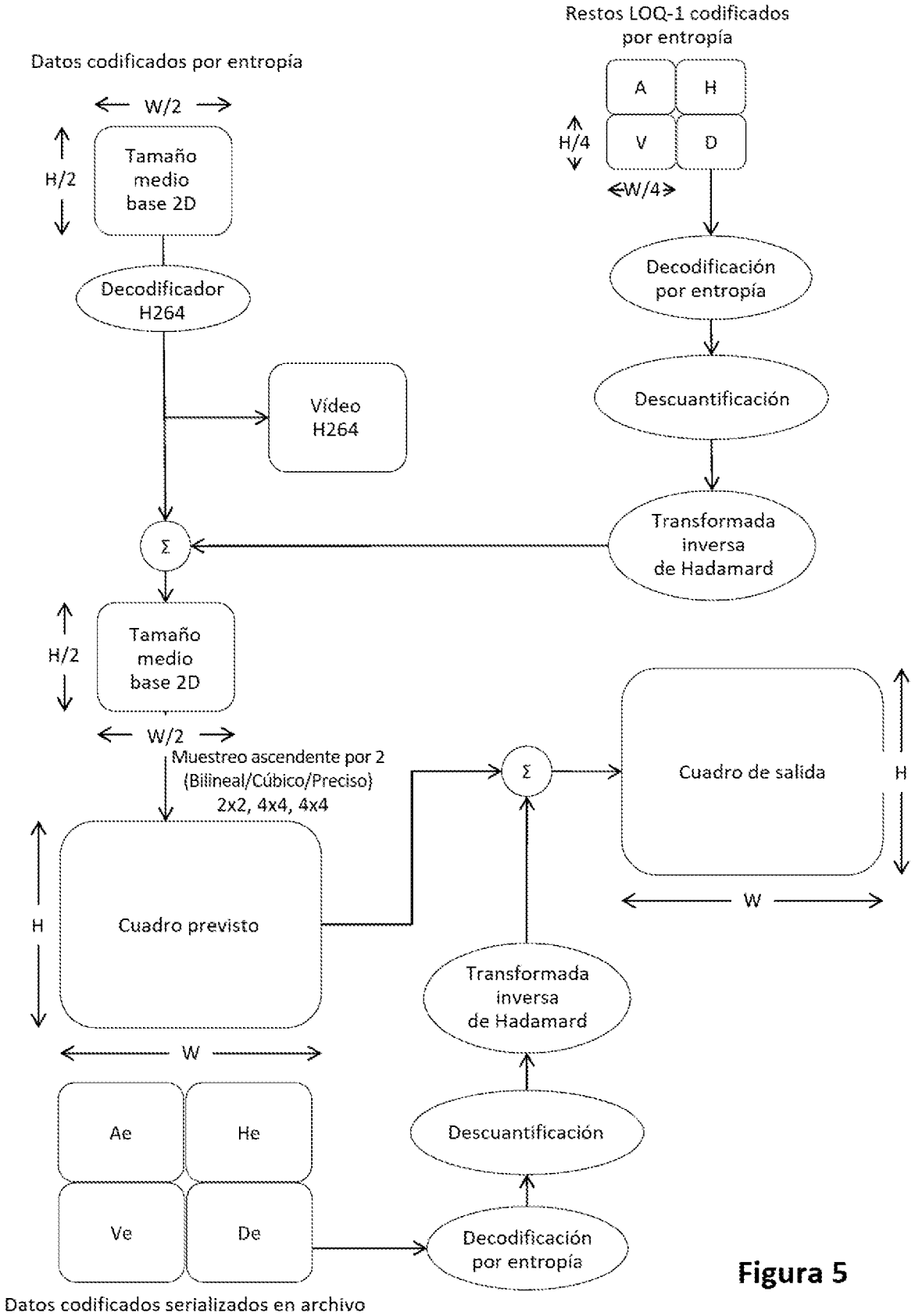
Restos LOQ-1 codificados por entropía
Datos codificados por entropía
A
H
V
D
H/4
W/4
W/2
H/2
Tamaño medio base 2D
Decodificador H264
Decodificación por entropía
Descuantificación
Vídeo H264
Transformada inversa de Hadamard
Σ
Tamaño medio base 2D
H/2
W/2
Muestreo ascendente por 2
(Bilineal/Cúbico/Preciso)
2x2, 4x4, 4x4
Σ
Cuadro de salida
H
W
H
Cuadro previsto
W
Transformada inversa de Hadamard
Ae
He
Descuantificación
Ve
De
Decodificación por entropía
Datos codificados serializados en archivo

Figura 5

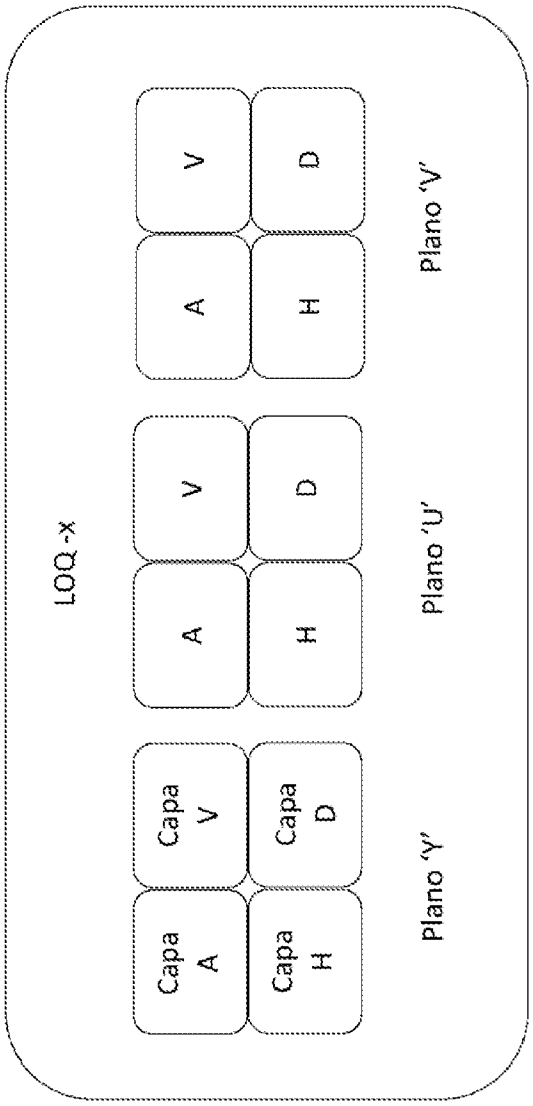
LOQ -x
Capa A
Capa H
Capa V
Capa D
Plano 'Y'
A
H
V
D
Plano 'U'
A
H
V
D
Plano 'V'

Figura 6

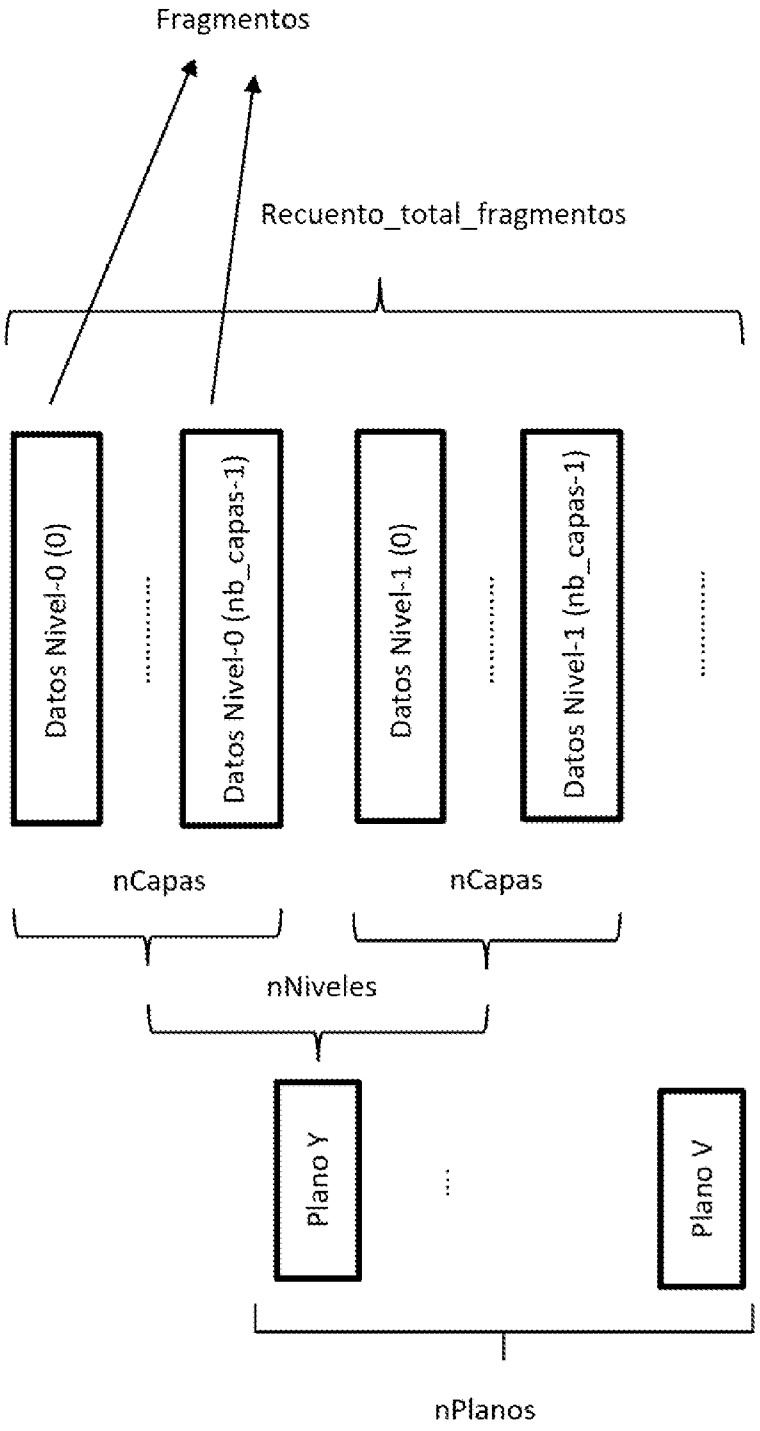
Fragmentos
Recuento_total_fragmentos
Datos Nivel-0 (0)
Datos Nivel-0 (nb_capas-1)
Datos Nivel-1 (0)
Datos Nivel-1 (nb_capas-1)
nCapas
nCapas
nNiveles
Plano Y
Plano V
nPlanos

Figura 7

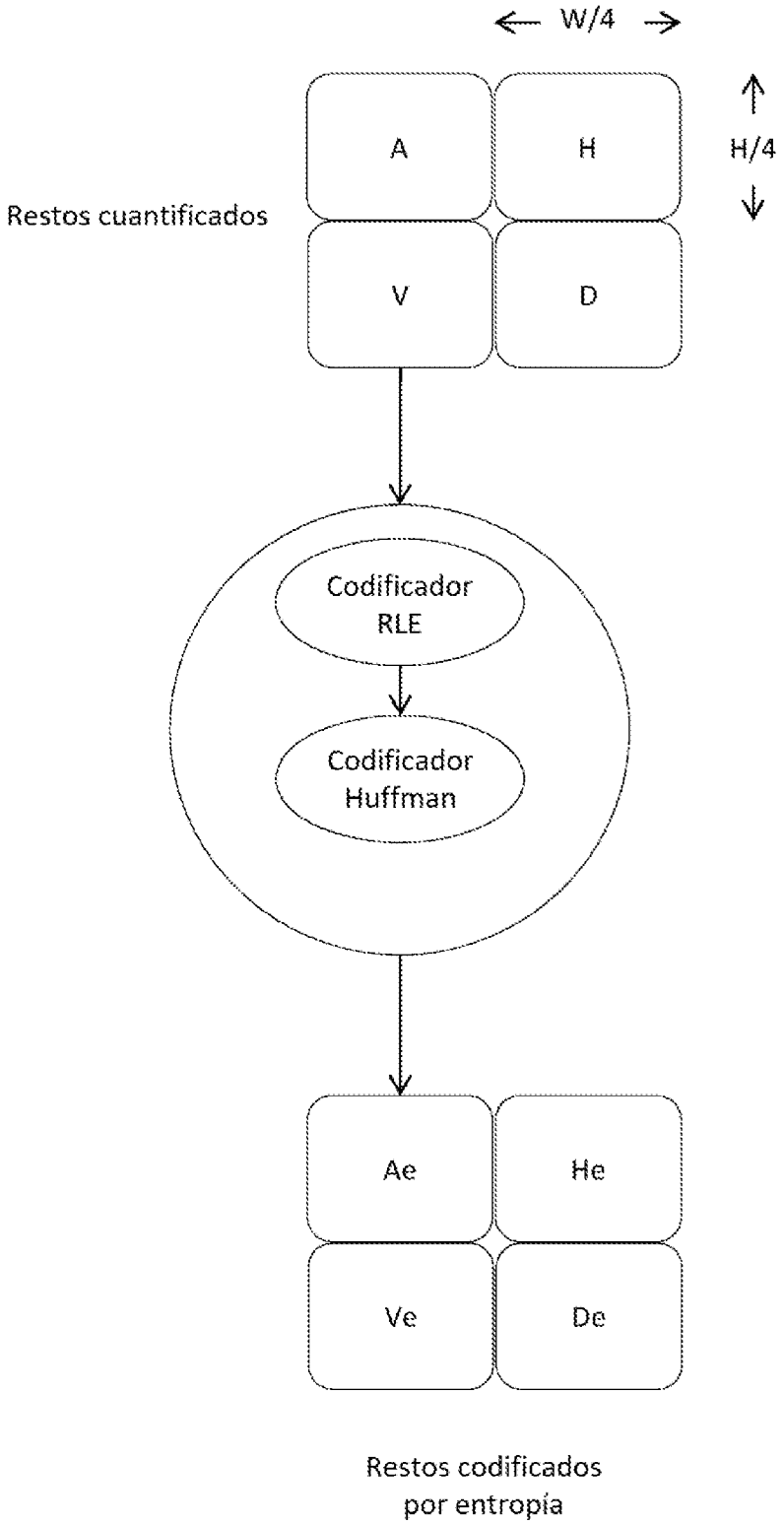
W/4
H/4
A
H
V
D
Restos cuantificados
Codificador RLE
Codificador Huffman
Ae
He
Ve
De
Restos codificados por entropía

Figura 8

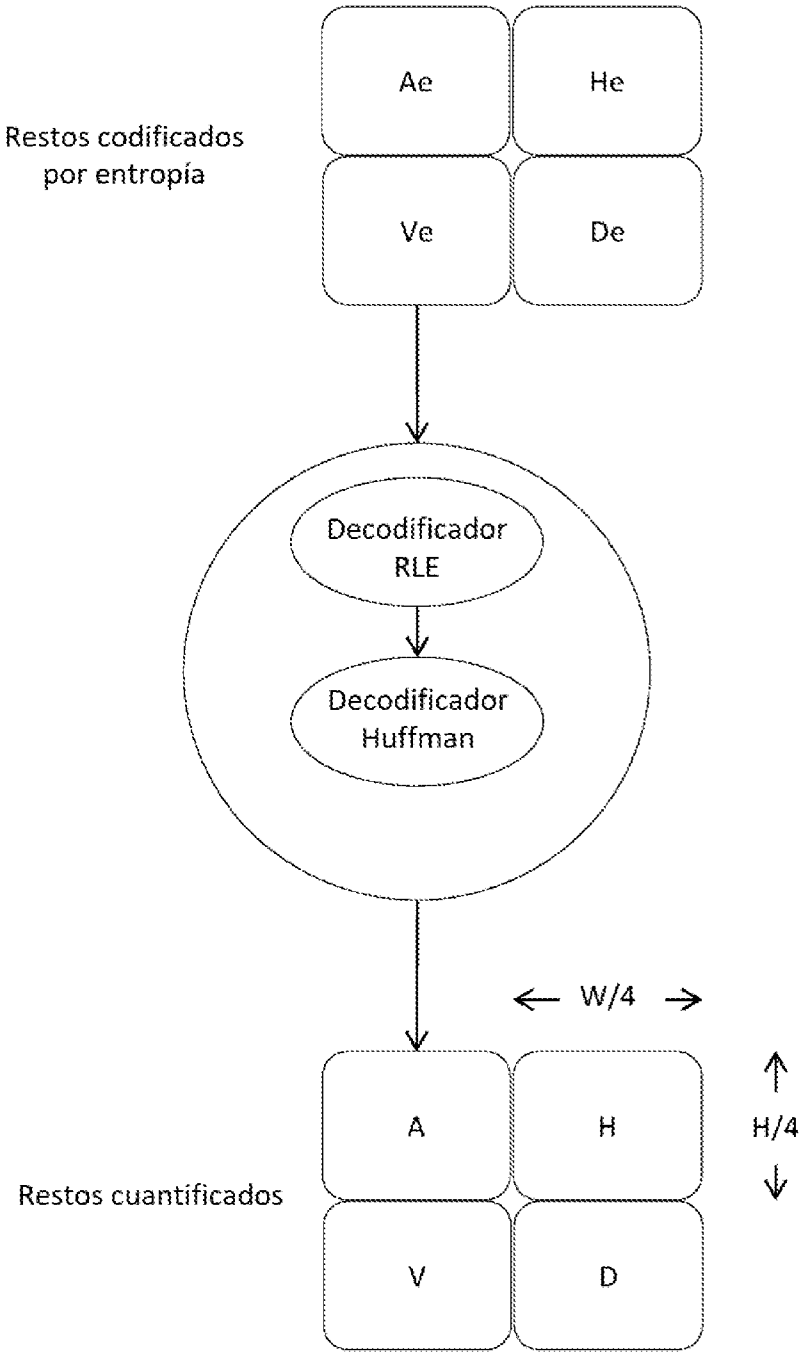
Restos codificados por entropía
Ae
He
Ve
De
Decodificador RLE
Decodificador Huffman
W/4
H/4
A
H
V
D
Restos cuantificados

Figura 9

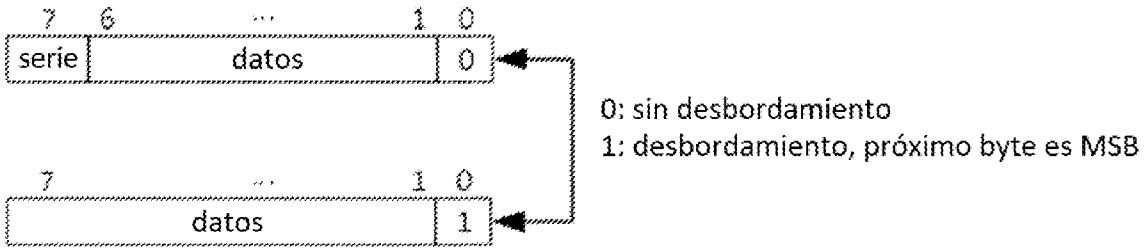

Figura 10

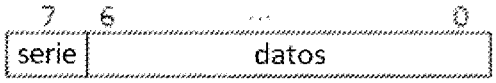

Figura 11

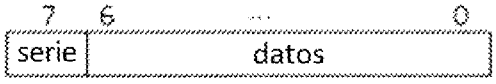

Figura 12

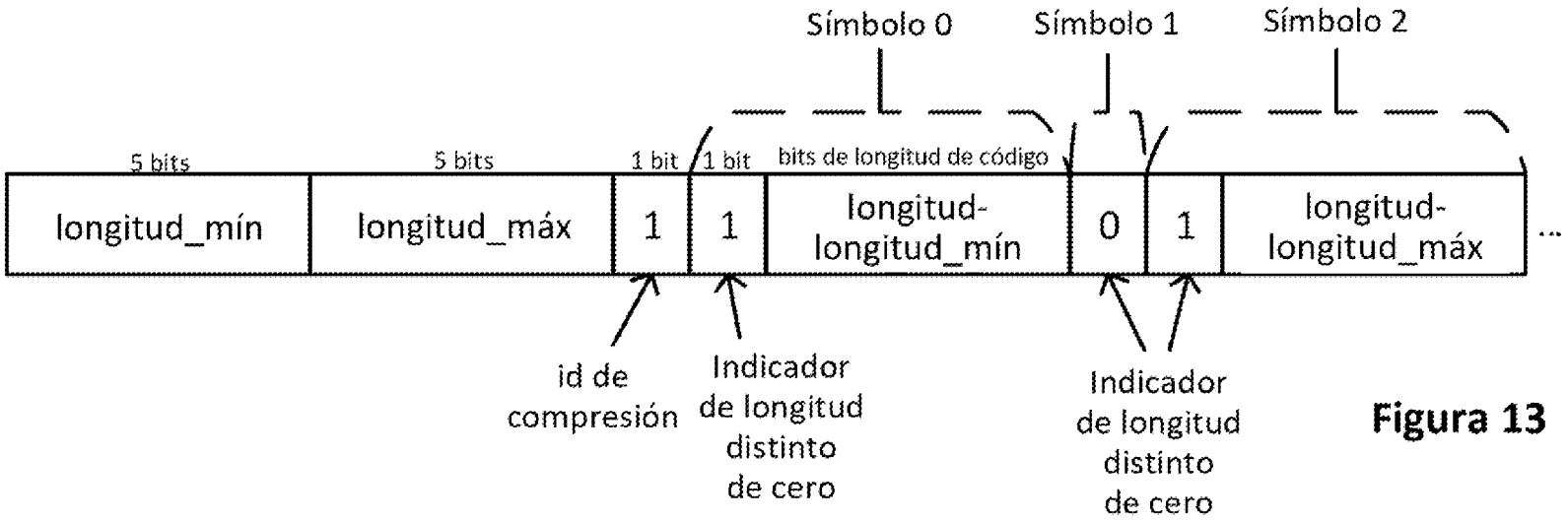

Figura 13

bits de longitud de código = log2(longitud_máx- longitud_mín+1)

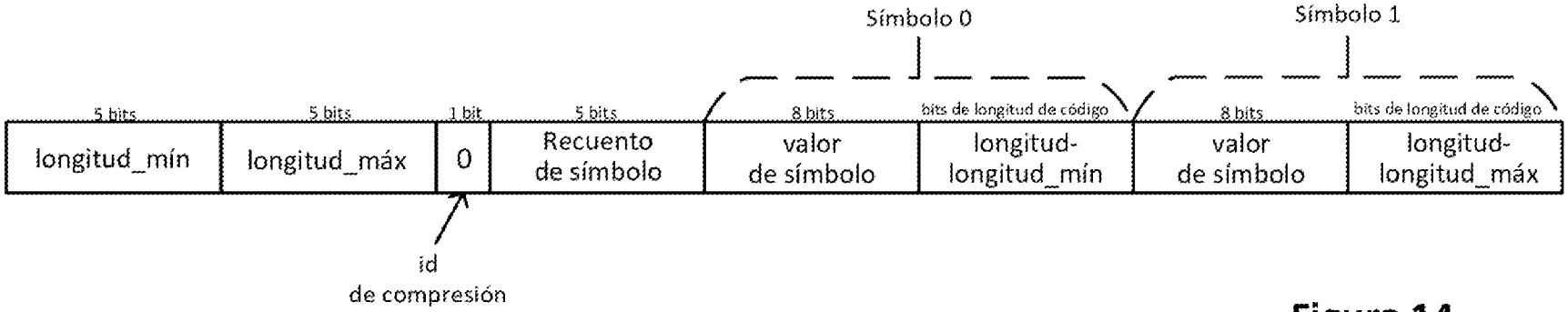

Figura 14

bits de longitud de código = log2(longitud_máx- longitud_mín+1)

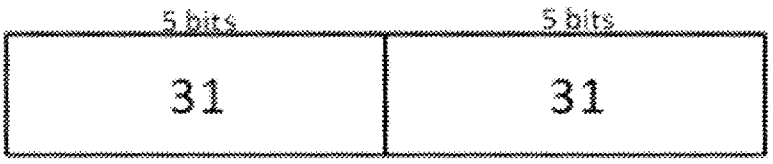

Figura 15

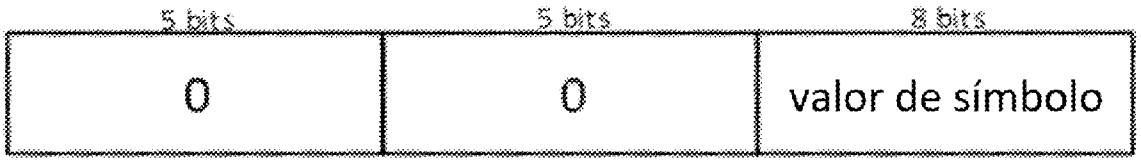

Figura 16

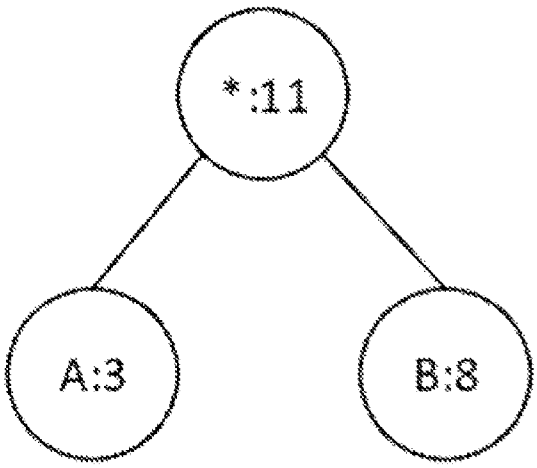

Figura 17a

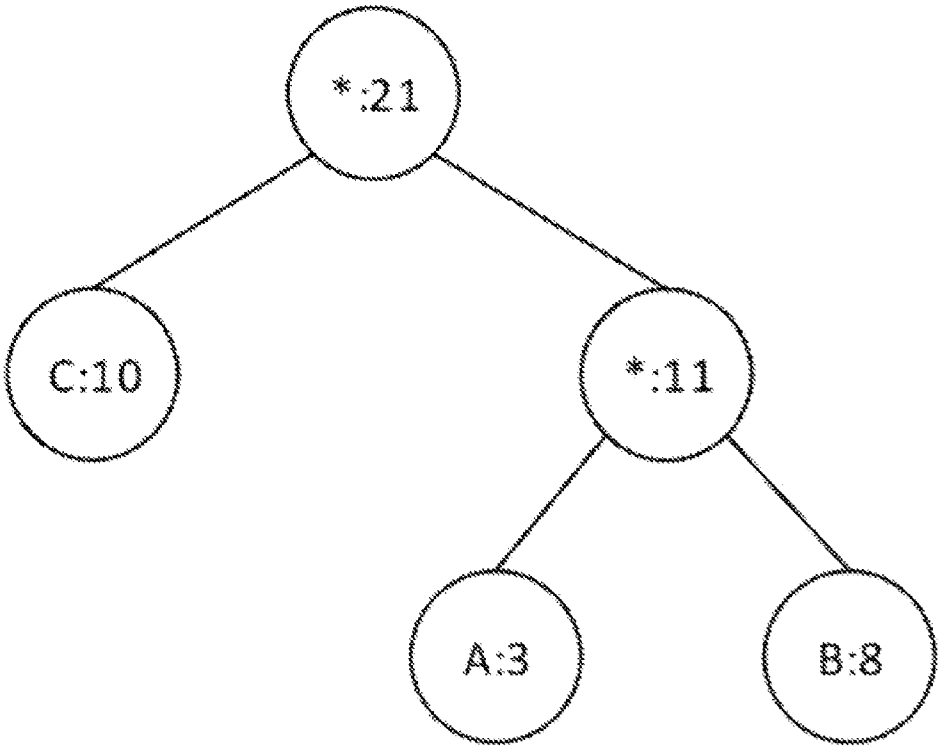

Figura 17b

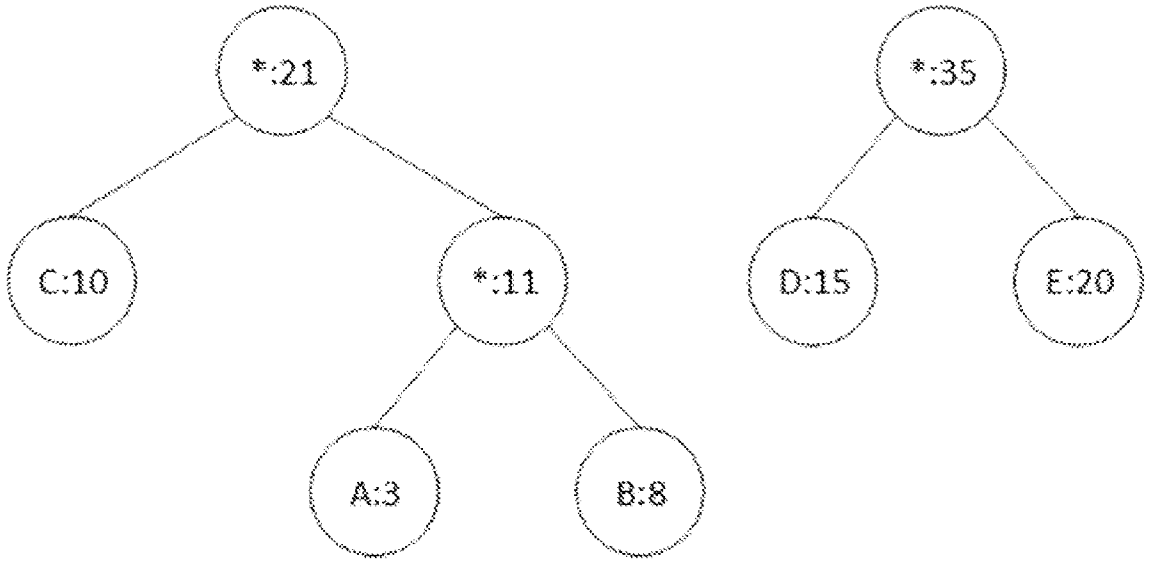

Figura 17c

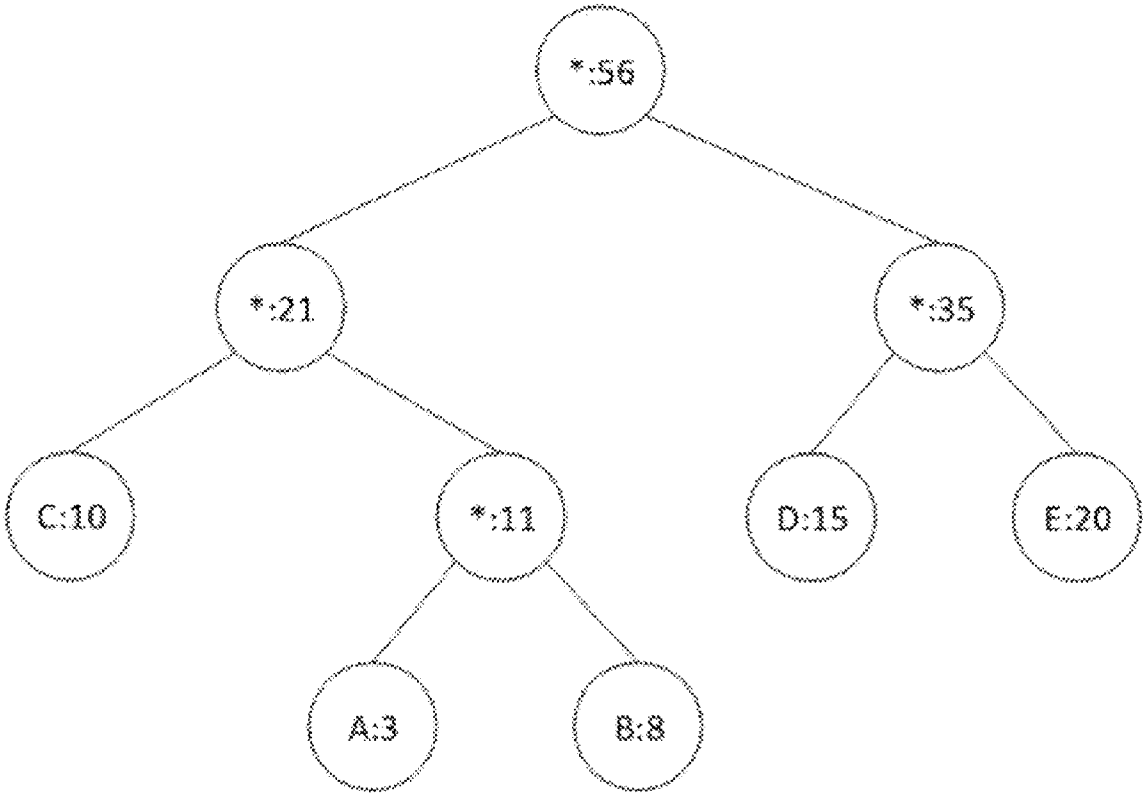

Figura 17d

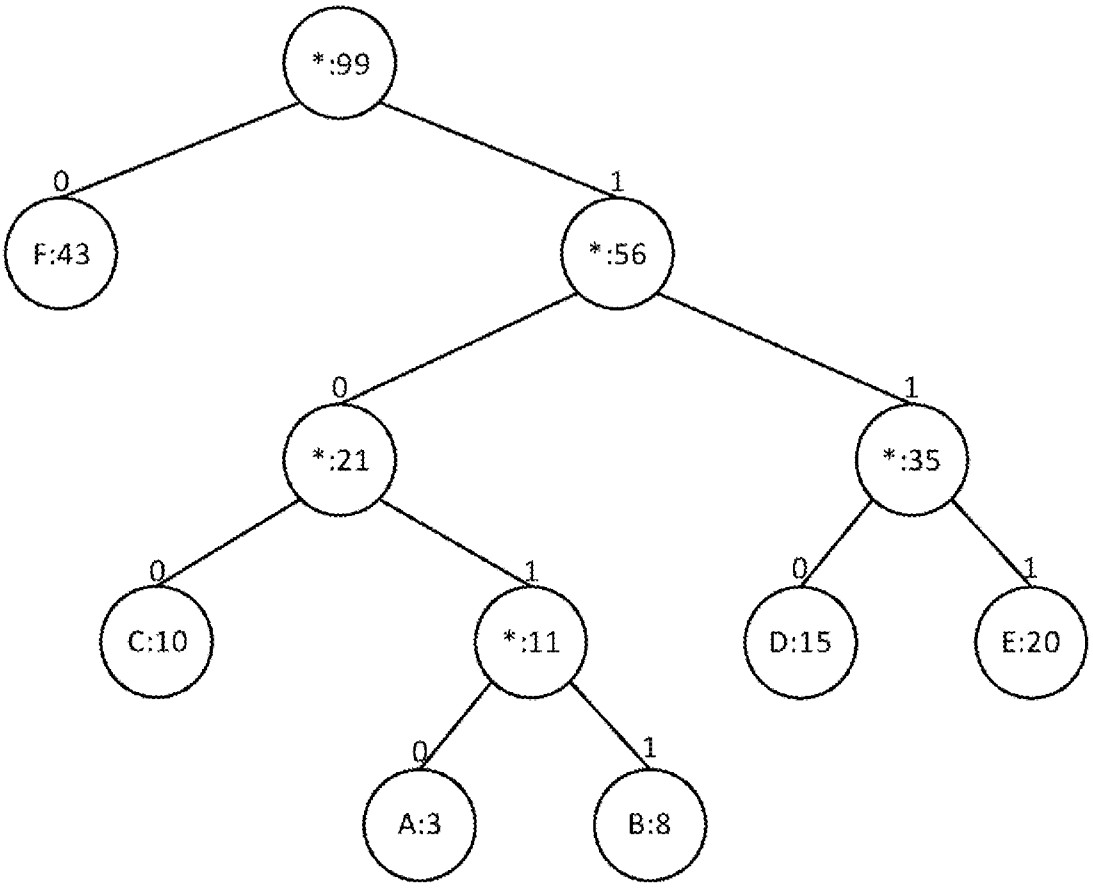

Figura 17e

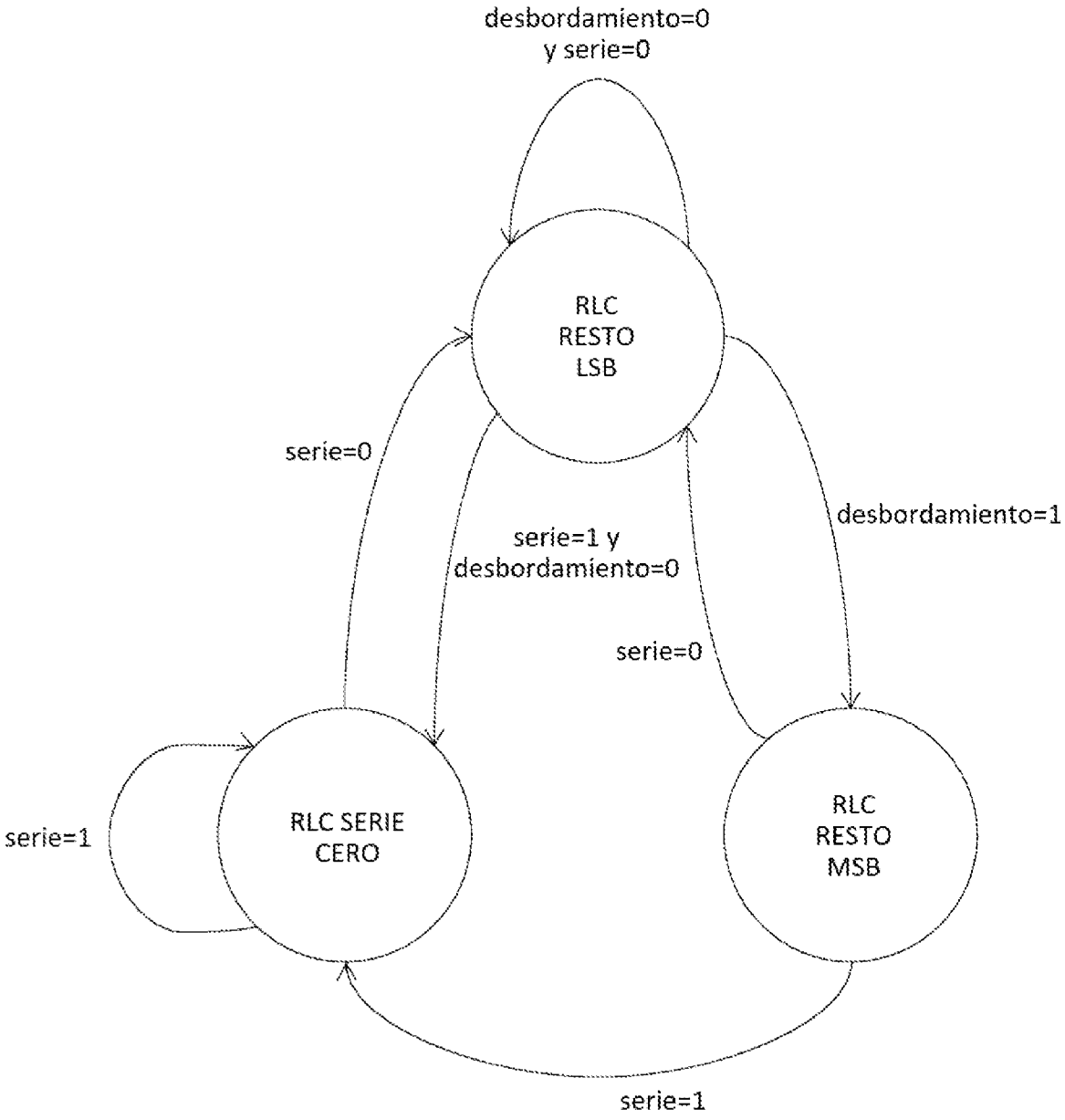
desbordamiento=0
y serie=0
RLC
RESTO
LSB
serie=0
serie=1 y
desbordamiento=0
desbordamiento=1
serie=0
serie=1
RLC SERIE
CERO
RLC
RESTO
MSB
serie=1

Figura 18