(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2021년04월08일
(11) 등록번호 10-2238058
(24) 등록일자 2021년04월02일

(51) 국제특허분류(Int. Cl.)
H04N 19/109 (2014.01) *H04N 19/11* (2014.01)
H04N 19/124 (2014.01) *H04N 19/13* (2014.01)
H04N 19/186 (2014.01)
(52) CPC특허분류
H04N 19/109 (2015.01)
H04N 19/11 (2015.01)
(21) 출원번호 10-2017-7001810
(22) 출원일자(국제) 2015년07월21일
심사청구일자 2020년07월17일
(85) 번역문제출일자 2017년01월20일
(65) 공개번호 10-2017-0036683
(43) 공개일자 2017년04월03일
(86) 국제출원번호 PCT/EP2015/025053
(87) 국제공개번호 WO 2016/012105
국제공개일자 2016년01월28일
(30) 우선권주장
1412937.3 2014년07월21일 영국(GB)
(56) 선행기술조사문헌
EP2723071 A1
XIAOLIN WU et al: "Context-Based, Adpative, Lossless Image Coding", IEEE Trans. on Communications, vol. 45, no. 4, 1 April 1997.

(73) 특허권자
구루로직 마이크로시스템스 오이
핀란드 투르쿠 20100 린난카투 34
(72) 발명자
칼레보 오씨
핀란드 아카아 핀-37800 케툰헨테 1
(74) 대리인
김태홍, 김진희

전체 청구항 수 : 총 30 항 심사관 : 김영태

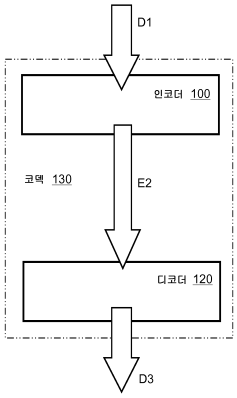
(54) 발명의 명칭 **인코더, 디코더 및 방법**

(57) 요 약

대응하는 인코딩된 데이터(E2)를 발생시키기 위해 입력 데이터(D1)를 인코딩하는 인코더(100)가 제공된다. 인코더(100)는 입력(D1)을 처리하고 적어도 그 일부분을 적어도 하나의 Delta 인코딩 알고리즘을 사용하여 인코딩하기 위해 그리고 입력 데이터(D1)의 하나 이상의 후속 부분들을 인코딩하는 데 사용하기 위한 하나 이상의 예측자

(뒷면에 계속)

대 표 도 - 도5



들을 발생시키기 위해 동작가능하며, 여기서 인코더(100)는 또한 인코딩된 데이터(E2)를 발생시키기 위해 적어도 하나의 Delta 인코딩 알고리즘에 의해 발생된 데이터 및 하나 이상의 예측자들을 적어도 하나의 엔트로피 인코딩 알고리즘을 이용하는 것에 의해 인코딩하기 위해 동작가능하다. 대응하는 디코딩된 데이터(D3)를 발생시키기 위해 인코딩된 데이터(E2)를 디코딩하는 디코더(120)가 제공되고; 임의로, 디코딩된 데이터(D3) 및 입력 데이터(D1)는 서로 유사하다.

(52) CPC특허분류
H04N 19/124 (2015.01)
H04N 19/13 (2015.01)
H04N 19/186 (2015.01)

명 세 서

청구범위

청구항 1

대응하는 인코딩된 데이터를 발생시키기 위해 입력 데이터를 인코딩하는 인코더(100)로서,

상기 인코더(100)는, 상기 입력 데이터를 처리하고 적어도 일부분의 원래 값들을, 적어도 하나의 Delta 인코딩 알고리즘을 사용하여, 상기 원래 값들의 값 범위에 비해 증가되지 않은 값 범위를 사용하여 표현되는 델타 값들로 인코딩하기 위해 그리고 상기 입력 데이터의 하나 이상의 후속 부분을 인코딩하는 데 사용하기 위한 하나 이상의 예측자를 발생시키도록 동작가능하며,

상기 인코더(100)는 또한, 적어도 하나의 엔트로피 인코딩 알고리즘을 이용하는 것에 의해, 상기 적어도 하나의 Delta 인코딩 알고리즘 및 상기 하나 이상의 예측자에 의해 발생된 데이터를 인코딩하여 상기 인코딩된 데이터를 발생시키도록 동작가능하고,

상기 적어도 하나의 Delta 인코딩 알고리즘은 ODelta, DDelta, IDelta, 및 PDelta의 유형들 중 하나이고,

상기 적어도 하나의 Delta 인코딩 알고리즘의 선택을 나타내는 정보는 상기 인코딩된 데이터에 포함되고,

상기 하나 이상의 예측자는,

(i) 하나 이상의 시간 예측자(temporal predictor);

(ii) 양자화가 적용된 하나 이상의 로컬 공간 예측자; 및

(iii) 미리 계산된 값들을 이용하는 하나 이상의 로컬 공간 예측자

중 적어도 하나를 포함하는 것인, 인코더(100).

청구항 2

제1항에 있어서, 상기 ODelta 유형의 Delta 인코딩 알고리즘은,

(a) 데이터 처리 장치(data processing arrangement)를 사용해서, 차분 인코딩 및 합 인코딩 중 적어도 하나의 형태를 상기 입력 데이터에 적용하여, 하나 이상의 대응하는 인코딩된 시퀀스를 발생시키는 것; 및

(b) 상기 데이터 처리 장치를 사용해서, 상기 하나 이상의 대응하는 인코딩된 시퀀스에, 최대 값에서의 랩어라운드(wrap around a maximum value) 및 최소 값에서의 랩어라운드(wrap around a minimum value) 중 적어도 하나를 적용하는 것

을 포함하는 것인, 인코더(100).

청구항 3

제1항 또는 제2항에 있어서,

상기 DDelta 유형의 Delta 인코딩 알고리즘은, 상기 델타 값들이 마이너스 값들만 포함할 때, 상기 델타 값들을 부호 없이 표현하는 것을 포함하고,

상기 IDelta 유형의 Delta 인코딩 알고리즘은, 상기 델타 값들이 플러스 값들만 포함할 때, 상기 델타 값들을 부호 없이 표현하는 것을 포함하고,

상기 PDelta 유형의 Delta 인코딩 알고리즘은, 상기 원래 값들의 상기 값 범위 내의 상기 델타 값들을 표현하기 위해 오프셋 값을 사용하는 것을 포함하는 것인, 인코더(100).

청구항 4

제1항 또는 제2항에 있어서, 상기 인코더(100)는 상기 입력 데이터를 인코딩하여 상기 인코딩된 데이터를 발생시킬 때, 적어도 하나의 양자화 알고리즘을 이용하도록 동작가능하고, 상기 적어도 하나의 양자화 알고리즘으로

인해 상기 인코더(100)가 상기 입력 데이터의 손실 인코딩을 제공하게 되는 것인, 인코더(100).

청구항 5

제1항 또는 제2항에 있어서, 상기 인코더(100)는 상기 입력 데이터에 존재하는 상이한 데이터 구조들의 데이터를 인코딩하는 데 상이한 알고리즘들을 이용하도록 동작가능한 것인, 인코더(100).

청구항 6

제1항 또는 제2항에 있어서, 상기 인코더(100)는 상기 입력 데이터를 블록 단위로 인코딩할 때 RD(rate-distortion) 최적화를 이용하도록 동작가능한 것인, 인코더(100).

청구항 7

제6항에 있어서, 상기 RD 최적화는 하기 식의 값(V)을 최소화하기 위해 상기 인코더(100) 내에서 계산되고,

$$V = D + \lambda * R$$

왜곡(D)은 상기 입력 데이터와, 상기 인코딩된 데이터로 인코딩되고 디코딩된 데이터로 디코딩된 상기 입력 데이터의 표현과의 사이의 제곱 오차(squares error)의 합이며, 레이트(rate)(R)는 비트로서 측정되는 인코딩된 데이터의 양을 나타내는 것인, 인코더(100).

청구항 8

제1항 또는 제2항에 있어서, 상기 적어도 하나의 Delta 인코딩 알고리즘 및 상기 적어도 하나의 엔트로피 인코딩 알고리즘 중의 적어도 하나는 DC 방법, 슬라이드 방법, 멀티레벨 방법, DCT 방법, 라인 방법, 스케일 방법, 데이터베이스 방법, 범위 코딩, Huffman 코딩, RLE 코딩, SRLE 코딩 중 적어도 하나를 이용하도록 동작가능한 것인, 인코더(100).

청구항 9

제1항 또는 제2항에 있어서, 상기 인코더(100)는 YUV 채널들, BGR 채널들 중 적어도 하나에 대응하는 데이터 구조들을 포함하는 상기 입력 데이터를 인코딩하도록 동작가능한 것인, 인코더(100).

청구항 10

제9항에 있어서, 상기 인코더(100)는 상기 채널들의 데이터를 Y, U, V 순서로 또는 G, B, R 순서로 인코딩하도록 동작가능한 것인, 인코더(100).

청구항 11

대응하는 인코딩된 데이터를 발생시키기 위해 입력 데이터를 인코딩하는 데 인코더(100)를 사용하는 방법으로서, 상기 방법은,

(i) 상기 인코더(100)를 사용하여, 상기 입력 데이터를 처리하고, 적어도 일부분의 원래 값들을, 적어도 하나의 Delta 인코딩 알고리즘을 사용하여, 상기 원래 값들의 값 범위에 비해 증가되지 않은 값 범위를 사용하여 표현되는 델타 값들로 인코딩하는 단계;

(ii) 상기 인코더(100)를 사용하여, 상기 입력 데이터의 하나 이상의 후속 부분을 인코딩하는 데 사용하기 위한 하나 이상의 예측자를 발생시키는 단계; 및

(iii) 상기 인코더(100)를 사용하여, 적어도 하나의 엔트로피 인코딩 알고리즘을 이용하는 것에 의해 상기 적어도 하나의 Delta 인코딩 알고리즘 및 상기 하나 이상의 예측자에 의해 발생된 데이터를 인코딩하여, 상기 인코딩된 데이터를 발생시키는 단계

를 포함하고,

상기 적어도 하나의 Delta 인코딩 알고리즘은 ODelta, DDelta, IDelta, 및 PDelta의 유형들 중 하나이고,

상기 적어도 하나의 Delta 인코딩 알고리즘의 선택을 나타내는 정보는 상기 인코딩된 데이터에 포함되고,

상기 하나 이상의 예측자는,

(i) 하나 이상의 시간 예측자;

(ii) 양자화가 적용된 하나 이상의 로컬 공간 예측자; 및

(iii) 미리 계산된 값들을 이용하는 하나 이상의 로컬 공간 예측자

중 적어도 하나를 포함하는 것인, 인코더(100)를 사용하는 방법.

청구항 12

제11항에 있어서, 상기 ODelta 유형의 Delta 인코딩 알고리즘은,

(a) 데이터 처리 장치를 사용해서, 차분 인코딩 및 합 인코딩 중 적어도 하나의 형태를 상기 입력 데이터에 적용하여, 하나 이상의 대응하는 인코딩된 시퀀스를 발생시키는 것; 및

(b) 상기 데이터 처리 장치를 사용해서, 상기 하나 이상의 대응하는 인코딩된 시퀀스에, 최대 값에서의 랩어라운드 및 최소 값에서의 랩어라운드 중 적어도 하나를 적용하는 것

을 포함하는 것인, 인코더(100)를 사용하는 방법.

청구항 13

제11항에 있어서,

상기 DDelta 유형의 Delta 인코딩 알고리즘은, 상기 델타 값들이 마이너스 값들만 포함할 때, 상기 델타 값들을 부호 없이 표현하는 것을 포함하고,

상기 IDelta 유형의 Delta 인코딩 알고리즘은, 상기 델타 값들이 플러스 값들만 포함할 때, 상기 델타 값들을 부호 없이 표현하는 것을 포함하고,

상기 PDelta 유형의 Delta 인코딩 알고리즘은, 상기 원래 값들의 상기 값 범위 내의 상기 델타 값들을 표현하기 위해 오프셋 값을 사용하는 것을 포함하는 것인, 인코더(100)를 사용하는 방법.

청구항 14

제11항 내지 제13항 중 어느 한 항에 있어서, 상기 방법은, 상기 입력 데이터를 인코딩하여 상기 인코딩된 데이터를 발생시킬 때, 적어도 하나의 양자화 알고리즘을 이용하도록 상기 인코더(100)를 구성하는 단계를 포함하고, 상기 적어도 하나의 양자화 알고리즘으로 인해 상기 인코더(100)가 상기 입력 데이터의 손실 인코딩을 제공하게 되는 것인, 인코더(100)를 사용하는 방법.

청구항 15

제11항 내지 제13항 중 어느 한 항에 있어서, 상기 방법은, 상기 입력 데이터에 존재하는 상이한 데이터 구조들의 데이터를 인코딩하는 데 상이한 알고리즘들을 이용하도록 상기 인코더(100)를 구성하는 단계를 포함하는 것인, 인코더(100)를 사용하는 방법.

청구항 16

제11항 내지 제13항 중 어느 한 항에 있어서, 상기 방법은, 상기 입력 데이터를 블록 단위로 인코딩할 때 RD 최적화를 이용하도록 상기 인코더(100)를 구성하는 단계를 포함하는 것인, 인코더(100)를 사용하는 방법.

청구항 17

제16항에 있어서, 상기 RD 최적화는 하기 식의 값(V)을 최소화하기 위해 상기 인코더(100) 내에서 계산되고,

$$V = D + \lambda * R$$

왜곡(D)은 상기 입력 데이터와, 상기 인코딩된 데이터로 인코딩되고 디코딩된 데이터로 디코딩된 상기 입력 데이터의 표현과의 사이의 제곱 오차의 합이며, 레이트(R)는 비트로서 측정되는 인코딩된 데이터의 양을 나타내는 것인, 인코더(100)를 사용하는 방법.

청구항 18

제11항 내지 제13항 중 어느 한 항에 있어서, 상기 적어도 하나의 Delta 인코딩 알고리즘 및 상기 적어도 하나의 엔트로피 인코딩 알고리즘 중의 적어도 하나는 DC 방법, 슬라이드 방법, 멀티레벨 방법, DCT 방법, 라인 방법, 스케일 방법, 데이터베이스 방법, 범위 코딩, Huffman 코딩, RLE 코딩, SRLE 코딩 중 적어도 하나를 이용하도록 동작가능한 것인, 인코더(100)를 사용하는 방법.

청구항 19

제11항 내지 제13항 중 어느 한 항에 있어서, 상기 방법은, YUV 채널들, BGR 채널들 중 적어도 하나에 대응하는 데이터 구조들을 포함하는 상기 입력 데이터를 인코딩하도록 상기 인코더(100)를 구성하는 단계를 포함하는 것인, 인코더(100)를 사용하는 방법.

청구항 20

제19항에 있어서, 상기 방법은, 상기 채널들의 데이터를 Y, U, V 순서로 또는 G, B, R 순서로 인코딩하도록 상기 인코더(100)를 구성하는 단계를 포함하는 것인, 인코더(100)를 사용하는 방법.

청구항 21

대응하는 디코딩된 데이터를 발생시키기 위해 인코딩된 데이터를 디코딩하는 디코더(120)로서,

상기 디코더(120)는, 상기 인코딩된 데이터에 적어도 하나의 엔트로피 디코딩 알고리즘을 적용하는 것에 의해 상기 인코딩된 데이터를 처리해서 처리된 데이터를 발생시키고, 적어도 하나의 Delta 디코딩 알고리즘과 결합하여 하나 이상의 예측자를 사용해서 상기 처리된 데이터를 디코딩하여, 디코딩된 데이터를 발생시키도록 동작가능하고,

상기 처리된 데이터는 상기 인코딩된 데이터가 발생된 원래 데이터의 값 범위에 비해 증가되지 않은 값 범위를 사용하여 표현되는 델타 값들을 포함하고,

상기 적어도 하나의 Delta 디코딩 알고리즘은 역 ODelta, 역 DDelta, 역 IDelta, 및 역 PDelta의 유형들 중 하나이고,

상기 적어도 하나의 Delta 디코딩 알고리즘의 선택을 나타내는 정보는 상기 인코딩된 데이터에 포함되고,

상기 하나 이상의 예측자는,

(i) 하나 이상의 시간 예측자;

(ii) 양자화가 적용된 하나 이상의 로컬 공간 예측자; 및

(iii) 미리 계산된 값들을 이용하는 하나 이상의 로컬 공간 예측자

중 적어도 하나를 포함하는 것인, 디코더(120).

청구항 22

제21항에 있어서, 상기 역 ODelta 유형의 Delta 디코딩 알고리즘은,

(a) 데이터 처리 장치를 사용해서, 차분 디코딩 및 합 디코딩 중 적어도 하나의 형태를 상기 인코딩된 데이터에 적용하여, 하나 이상의 대응하는 디코딩된 시퀀스를 발생시키는 것; 및

(b) 상기 데이터 처리 장치를 사용해서, 상기 하나 이상의 대응하는 디코딩된 시퀀스에, 최대 값에서의 랩어라운드 및 최소 값에서의 랩어라운드 중 적어도 하나를 적용하는 것

을 포함하는 것인, 디코더(120).

청구항 23

제21항 또는 제22항에 있어서,

상기 역 DDelta 유형의 Delta 디코딩 알고리즘은, 상기 델타 값들이 마이너스 값들만 포함할 때, 상기 델타 값들을 디코딩하는 것을 포함하고,

상기 역 IDelta 유형의 Delta 디코딩 알고리즘은, 상기 델타 값들이 플러스 값들만 포함할 때, 상기 델타 값들을 디코딩하는 것을 포함하고,

상기 역 PDelta 유형의 Delta 디코딩 알고리즘은, 상기 원래 데이터의 상기 값 범위 내에서 표현되는 상기 델타 값들을 디코딩하기 위해 오프셋 값을 사용하는 것을 포함하는 것인, 디코더(120).

청구항 24

제21항 또는 제22항에 있어서, 상기 디코더(120)는 상기 인코딩된 데이터를 디코딩하여 상기 디코딩된 데이터를 발생시킬 때, 적어도 하나의 양자화 알고리즘을 이용하도록 동작가능하고, 상기 적어도 하나의 양자화 알고리즘으로 인해 상기 디코더(120)가 상기 인코딩된 데이터의 손실 디코딩을 제공하게 되는 것인, 디코더(120).

청구항 25

제21항 또는 제22항에 있어서, 상기 디코더(120)는 상기 인코딩된 데이터에 존재하는 상이한 데이터 구조들의 데이터를 디코딩하는 데 상이한 알고리즘들을 이용하도록 동작가능한 것인, 디코더(120).

청구항 26

제21항 또는 제22항에 있어서, 상기 적어도 하나의 Delta 디코딩 알고리즘 및 상기 적어도 하나의 엔트로피 디코딩 알고리즘 중의 적어도 하나는 DC 방법, 슬라이드 방법, 멀티레벨 방법, DCT 방법, 라인 방법, 스케일 방법, 데이터베이스 방법, 범위 코딩, Huffman 코딩, RLE 코딩, SRLE 코딩 중 적어도 하나를 이용하도록 동작가능한 것인, 디코더(120).

청구항 27

제21항 또는 제22항에 있어서, 상기 디코더(120)는 YUV 채널들, BGR 채널들 중 적어도 하나에 대응하는 데이터 구조들을 포함하는 상기 인코딩된 데이터를 디코딩하도록 동작가능한 것인, 디코더(120).

청구항 28

제27항에 있어서, 상기 디코더(120)는 상기 채널들의 데이터를 Y, U, V 순서로 또는 G, B, R 순서로 디코딩하도록 동작가능한 것인, 디코더(120).

청구항 29

대응하는 데이터를 발생시키기 위해, 디코더(120)로, 인코딩된 데이터를 디코딩하는 방법으로서, 상기 디코딩 방법은,

상기 인코딩된 데이터에 적어도 하나의 엔트로피 디코딩 알고리즘을 적용하는 것에 의해 상기 인코딩된 데이터를 처리하여 처리된 데이터를 발생시키는 단계, 및

적어도 하나의 Delta 디코딩 알고리즘과 결합하여 하나 이상의 예측자를 사용해서 상기 처리된 데이터를 디코딩하여, 디코딩된 데이터를 발생시키는 단계

를 포함하고,

상기 처리된 데이터는 상기 인코딩된 데이터가 발생되는 원래 데이터의 값 범위에 비해 증가되지 않은 값 범위를 사용하여 표현되는 델타 값들을 포함하고,

상기 적어도 하나의 Delta 디코딩 알고리즘은 역 ODelta, 역 DDelta, 역 IDelta, 및 역 PDelta의 유형들 중 하나이고,

상기 적어도 하나의 Delta 디코딩 알고리즘의 선택을 나타내는 정보는 상기 인코딩된 데이터에 포함되고,

상기 하나 이상의 예측자는,

(i) 하나 이상의 시간 예측자;

(ii) 양자화가 적용된 하나 이상의 로컬 공간 예측자; 및

(iii) 미리 계산된 값들을 이용하는 하나 이상의 로컬 공간 예측자

중 적어도 하나를 포함하는 것인, 디코딩 방법.

청구항 30

장치의 처리 수단에 의해 실행될 때, 상기 장치로 하여금, 제11항 내지 제13항, 및 제29항 중 어느 한 항의 방법을 실행하게 하도록 구성된 컴퓨터 판독가능 명령어를 포함하는, 비일시적 저장 매체에 저장된 컴퓨터 프로그램.

청구항 31

삭제

청구항 32

삭제

청구항 33

삭제

청구항 34

삭제

발명의 설명

기 술 분 야

[0001] 본 개시내용은 데이터를 인코딩하는 방법들에, 예를 들어, 하나 이상의 예측자(predictor)들을 이용하는 Delta 코딩(Delta coding)을 사용하여 데이터를 인코딩하는 방법에 관한 것이다. 더욱이, 본 개시내용은 데이터를 디코딩하는 방법들에, 예를 들어, 하나 이상의 예측자들을 이용하는 Delta 디코딩(Delta decoding)을 사용하여 데이터를 디코딩하는 방법에 관한 것이다. 더욱이, 본 개시내용은 앞서 언급된 방법들을 구현하기 위한 시스템들, 장치들 및 디바이스들에 관한 것이다. 게다가, 본 개시내용은 컴퓨터 판독가능 명령어들 - 컴퓨터 판독가능 명령어들은 상기한 방법들을 실행하기 위해 처리 하드웨어를 포함하는 컴퓨터화된 디바이스에 의해 실행가능함 - 이 저장되어 있는 비일시적 컴퓨터 판독가능 저장 매체를 포함하는 컴퓨터 프로그램 제품들에 관한 것이다.

배 경 기 술

[0002] 종래에는, 일반적으로, 많은 비디오 코덱들, 예를 들어, MPEG-4, H.264, VC-1, HEVC 및 VP9가 영상 블록들의 움직임 추정을 위해 이전 프레임들을 이용할 수 있고; 이 예시적인 코덱 이름들은 상표들을 포함한다. 움직임 추정 및 움직임 보상은 각각의 비디오 영상 프레임에 대해 블록 단위로 실행된다. 이와 유사하게, 데이터베이스들의 사용을 통한 중복 제거(de-duplication) 방법들 또는 처리는 주어진 현재 데이터 블록 또는 데이터 패킷을 인코딩할 때 이미 코딩된 데이터 블록들 또는 데이터 패킷들을 이용하기 위해 사용될 수 있다. Delta 코딩은 비디오 또는 유사한 유형들의 콘텐츠에 존재하는 데이터 심볼들의 엔트로피를 감소시키기 위해 사용될 수 있다. ODelta 코딩(ODelta coding)은, 이하에서 보다 상세히 설명될 것인 바와 같이, 또한 데이터 심볼들의 엔트로피를 추가로 감소시키기 위해 임의로 사용된다. 더욱이, ODelta 코딩은 개개의 비트들의 엔트로피를 감소시키는 것을 가능하게 만들고; ODelta 코딩과 연관된 방법들은 나중에 부록 1에서 보다 상세히 기술될 것이다. DPCM-스타일 방법들 - 즉, Delta 코딩 및 ODelta 코딩 - 둘 다는, 엔트로피 인코더에 대한 인코딩된 데이터 값을 생성할 때, 이전 데이터 값들을 이용한다.

[0003] 데이터의 양 및 그의 전송이 현재 급속히 증가하고 있기 때문에, 데이터 압축의 필요성도 증가하고 있으며, 데이터 압축의 효율을 개선시키기 위해 새롭고 보다 나은 방법들이 필요하다. 데이터 - 예를 들어, 영상들, 비디오, 오디오, 측정 데이터, 또는 다양한 유형들의 이진 데이터, ASCII 데이터 등 - 는, 예를 들어, 하나 이상의 센서들로부터 포착될 수 있고; 포착된 센서 데이터와 추상 데이터의 혼합이 또한 실현가능하다.

[0004] 데이터를 인코딩할 때 다수의 코딩 방법들이 현재 이용가능하지만, 그들 중 어느 것도 상이한 종류들의 데이터 모두에 대해 충분히 양호한 압축비를 제공하지 않는다. 주어진 현재 채널, 프레임, 데이터 블록 또는 데이터

패킷을 인코딩할 때, 이미 코딩된 채널들, 프레임들, 데이터 블록들 또는 데이터 패킷들의 정보를 여전히 이용하면서, 상이한 데이터 채널들 또는 프레임들, 예를 들어, 색, 영상 채널들, 오디오 채널들, 병렬 데이터 측정, 비디오에서의 개별 영상들, 오디오에서의 개별 패킷들, 3D 영상들, 3D 오디오 등을, 개별적으로, 코딩할 필요도 종종 있다. 공지된 데이터 인코딩 방법들은 이러한 아주 다양한 데이터 구조들을 가지고 있는 입력 데이터에 대처하기에 충분히 만능이지는 않다.

[0005] 이와 유사하게, 이미 인코딩되어 있는 공간적 정보가 또한 유리하게도 DPCM(http://en.wikipedia.org/wiki/DPCM)과 같은 공지된 방법을 사용하는 것보다 인코딩하는 데 더 효율적으로 이용된다. 데이터의 무손실 및 손실 코딩이 달성될 수 있게 하는 간단하지만 효율적인 인코딩 및 디코딩 방법들도 필요하다. 방법이, 나중에 부록 1에서 보다 상세히 기술될 것인 바와 같이, ODelta 방법을 통해 제공되지만, ODelta 방법들을 훨씬 더 효율적으로, 예를 들어, 전형적인 공지된 Delta 코딩을 수정하여, DPCM으로부터의 Delta 코딩이 상이한 예측자들에 대해서도 사용하기에 적합하게 만드는 방식으로 이용할 필요가 있다. 이러한 접근법은, 예를 들어, DPCM에서와 같이 종래의 공지된 Delta 코딩보다 마이너스 차이와 플러스 차이, 또는 합, 심볼들에 대한 값들에 대해 더 나은 엔트로피 감소를 가능하게 한다.

[0006] 양쪽 특성들 - 즉, 진보된 예측 방법들을 이용하는 것에 의한 데이터 재이용과, 즉, 프레임, 채널, 데이터 블록 또는 데이터 패킷에 대한 방법 선택, 및 인코딩된 잔차 값들 - 즉, 어떤 움직임 벡터들, 선택 심볼들 또는 데이터베이스 참조들도 갖지 않음 - 만을 전달하거나 저장하는 것에 의한, 양자화를 갖는 또는 갖지 않는 잔차 데이터의 효율적인 엔트로피 감소 - 을 겸비할 수 있는 공지된 방법들이 없다. 때때로, 심지어 잔차 코딩이 필요하지 않은데, 그 이유는,

[0007] (i) 예측이 완벽하거나;

[0008] (ii) 잔차가 모든 데이터 값들에 대해 일정하고, 그것을 전달하는 데 하나의 값으로 충분하거나;

[0009] (iii) 양자화를 갖거나 갖지 않는 잔차가 품질 파라미터에 기초한 오차 문턱값 미만이기 때문이다.

발명의 내용

[0010] 본 발명은, Delta 인코딩 알고리즘들의 사용에 기초하여, 데이터를 인코딩하는 개선된 방법을 제공하려고 한다.

[0011] 더욱이, 본 발명은, Delta 인코딩 알고리즘들의 사용에 기초하여, 데이터를 인코딩하는 개선된 인코더를 제공하려고 한다.

[0012] 게다가, 본 발명은, 역 Delta 인코딩 알고리즘들의 사용에 기초하여, 데이터를 디코딩하는 개선된 방법을 제공하려고 한다.

[0013] 게다가, 본 발명은, 역 Delta 인코딩 알고리즘들의 사용에 기초하여, 데이터를 디코딩하는 개선된 디코더를 제공하려고 한다.

[0014] 제1 양태에 따르면, 대응하는 인코딩된 데이터(E2)를 발생시키기 위해 입력 데이터(D1)를 인코딩하는 인코더가 제공되고, 인코더는 입력(D1)을 처리하고 적어도 일부분의 원래 값들을, 적어도 하나의 Delta 인코딩 알고리즘을 사용하여, 원래 값들의 값 범위에 비해 증가되지 않은 값 범위를 사용하여 표현되는 델타 값들로 인코딩하기 위해 그리고 입력 데이터(D1)의 하나 이상의 후속 부분들을 인코딩하는 데 사용하기 위한 하나 이상의 예측자들을 발생시키기 위해 동작가능하며, 여기서 인코더는 또한 인코딩된 데이터(E2)를 발생시키기 위해 적어도 하나의 Delta 인코딩 알고리즘(ODelta, DDelta, IDelta, PDelta)에 의해 발생된 데이터 및 하나 이상의 예측자들을 적어도 하나의 엔트로피 인코딩 알고리즘을 이용하는 것에 의해 인코딩하기 위해 동작가능하고, 여기서 하나 이상의 예측자들은

[0015] (i) 하나 이상의 시간 예측자(temporal predictor)들;

[0016] 삭제

[0017] (ii) 양자화가 적용된 하나 이상의 로컬 공간 예측자들; 및

[0018] (iii) 미리 계산된 값들을 이용하는 하나 이상의 로컬 공간 예측자들

[0019] 중 적어도 하나를 포함한다.

[0020] 본 발명은 Delta 인코딩, 하나 이상의 로컬 공간 예측자들의 발생, 각종의 상이한 델타 값 전달 방법들을 이용하는 것, 및 엔트로피 인코딩의 조합이 아주 효율적인 데이터 인코딩을 제공할 수 있다는 점에서 장점이 있다.

[0021] "로컬 공간 예측자"는 임의로 축약형으로서 "로컬 예측자" 또는 "공간 예측자"라고 지칭된다.

[0022] 미리 계산된 값들과 관련하여, 값들은, 공간 위치에 기초하여, 데이터에 대한 프로세스를 수행하기 전에 결정될 수 있는 임의의 값들일 수 있고; 즉, 부록 1에서 기술되는 바와 같이, ODelta 인코딩에서와 같이 로컬 공간 예측자들이 아니다. 이러한 미리 계산된 값들의 예들은 인코딩될 현재 데이터 블록에 대해 좌측에 또는 위쪽에 있는 이전 블록, 어떤 이전 채널, 뷰 또는 프레임에서의 동일한 데이터 블록의 위치, 또는 주어진 데이터 프레임 내에서의 내부 움직임 예측을 포함한다.

[0023] 본 방법에 따른 예측이 전형적으로 사전에 단 한번에 결정된다. 즉, 주어진 블록에 대한 예측이 시간 예측 또는 공간 예측 중 어느 하나에 의해 사전에 단 한번에 결정된다. 앞서 언급된 블록들, 채널들, 뷰들 및 프레임들에 부가하여, 움직임 추정이 또한 편리하게도 시간 예측인 것으로 간주된다. 움직임 추정이 또한 공간적으로 수행될 수 있고, 이 경우에 움직임 추정은 앞서 요소 (iv)에 정의된 공간 예측자의 문제이다. 즉, 움직임 추정 예측들이 항상 시간적으로 서로 상이할 필요가 없다. 본원에서의 공간 예측은, 예를 들어, 부록 1에 기술된, ODelta 방법에서 이용되는 "로컬 공간 예측"과 상이하다. "로컬 공간 예측"이 사용될 때, 예측은 방법 프로세스 동안 일어나고, 예측자 값들은 이전 값들이 처리되기 전에는 이용가능하지 않다. 이것은 시간 예측들 및 다른 공간 예측들과 비교하여 상이하다. 임의로, 예측은, ODelta(부록 1을 참조)에서와 같이, 프로세스가 진행될 때, 결정될 수 있다. 이러한 경우에, Delta 코딩에서의 로컬 공간 예측자들이 사용된다.

[0024] 임의로, 인코더에서, 적어도 하나의 Delta 인코딩 알고리즘은 하기의 것들에 의해 구현된다:

[0025] (a) 하나 이상의 대응하는 인코딩된 시퀀스들을 발생시키기 위해 차분 및/또는 합 인코딩의 형태를 입력 데이터(DA1)에 적용하기 위해 데이터 처리 장치(data processing arrangement)를 사용하는 것; 및

[0026] (b) 인코딩된 출력 데이터(DA2 또는 DA3)(= ODelta)를 발생시키기 위해 하나 이상의 대응하는 인코딩된 시퀀스들에 최대 값에서의 랩어라운드(wrap around a maximum value) 및/또는 최소 값에서의 랩어라운드(wrap around a minimum value)를 적용하기 위해 데이터 처리 장치를 사용하는 것.

[0027] 임의로, Delta 인코딩 알고리즘을 하기의 방식으로 사용하는 것이 또한 가능하다. 주어진 원래 값과 대응하는 예측된 값 사이의 차이가 항상 0이거나 플러스(positive)인 경우, 또는 주어진 원래 값과 대응하는 예측된 값 사이의 차이가 0이거나 마이너스(negative)인 경우, 양자화가 적용되든 그렇지 않든 간에, 코딩 알고리즘 정보와 함께 델타 값들의 부호, 예를 들어, 부호 비트만을 표현하고 전달하는 것이 가능하다. 이와 같이, 이러한 경우에, ODelta 알고리즘에서 이용되는 바와 같은 래핑(wrapping)이 전혀 필요하지 않을 것인데, 그 이유는, 래핑이 없더라도, 값들이 주어진 비트 카운트 범위에 항상 들어갈 것이기 때문이다. 이러한 방식으로 사용되는 이러한 알고리즘들은 편리하게도, 예를 들어, IDelta("Incremental Delta") 방법 및 DDelta("Decremental Delta") 방법이라고 지칭된다. IDelta 방법은 따라서 플러스 델타 값들만을 전달하고, DDelta 방법은 마이너스 델타 값들만을 전달하지만, 종종 유리하게도 부호 - 즉, 부호 비트 - 를 스와핑(swap)한다. 이 스와핑은 DDelta 값들에 대해서만 그리고 DDelta 값들을 전달할 때에만 행해진다. 앞서 기술된 방법들 둘 다는, 물론, 또한 변하지 않은 값들을 0 심볼/값과 함께, 이러한 변하지 않음이 양자화를 사용해 또는 양자화를 사용함이 없이 달성되었는지에 관계없이, 항상 전달할 수 있고; 즉, 이러한 경우에, 그 델타 값들 사이의 차이는 사용되는 양자화보다 더 크지 않다.

[0028] 더욱이, 임의로, 델타 값들을 페데스탈 값(pedestal value)과 함께 사용하는 것이 또한 가능하다. 이것은 마이너스 델타 값과 플러스 델타 값 둘 다가 있지만, 그들의 절대 값들이 코딩되는 데이터의 동적 범위 및 비트 깊이와 관련하여 작은 경우, 양자화되든 그렇지 않든 간에, 변화의 페데스탈 값 - 즉, 가장 큰 마이너스 변화 - 을 전달하는 것이 때때로 유익하다는 것을 의미한다. 그 후에, 앞서 언급된 IDelta 방법에서와 같이, 플러스 변화 값들만을 전달하는 것이 가능하다. 페데스탈 값을 이용하는 이러한 방법은 편리하게도 "PDelta 방법"이라고 지칭될 수 있다. 종종, PDelta 방법이 사용될 때는, 원래의 ODelta 알고리즘(부록 1을 참조)이 사용될 때보다 데이터의 동적 범위/비트 깊이가 더 많이 감소될 수 있다. 즉, 데이터가 보다 적은 비트들을 사용하여 표현될 수 있는데, 그 이유는 데이터의 최대 값이 보다 작을 것이고 따라서 가능한 가장 작은 값과 가능한 가장 큰 값 사이의 차이가 그에 대응하여 보다 작을 것이기 때문이며; 이러한 이점이 발생하는 이유는 원래의 ODelta 방법(부록 1을 참조)이 나오는 값들의 동적 범위 - 즉, 전달될 ODelta 값들의 값 범위 - 를 항상 표현하고 전달해야만 하기 때문이다.

[0029] 앞서 기술된 IDelta 방법, DDelta 방법 및 PDelta 방법은 사용되고 있는 예측이, 예를 들어, 이전 블록, 채널, 영상 또는 어떤 다른 결정된 값들의 세트이고, 이 방법들을 사용하기 전에 선언되는 본 개시내용의 실시예들에서 사용하기에 유익하다. 따라서, 코딩되고 있는 값과 예측 값 사이의 차이가 단 한번에 쉽게 결정되고, 따라서 값 또는 차이에 대한 예측이 코딩되고 있는 다른 값들에 전혀 의존하지 않는다. "시간 예측자" 또는 "공간 예측자"라고 지칭되거나 불리우는, 이러한 종류들의 예측 해결책들은 또한 양자화와 함께 사용되는 데 특히 아주 적합한데, 그 이유는 차이 값들의 양자화가 디코딩될 단일의 주어진 개별 데이터 값에만 영향을 줄 것이고 따라서 양자화에 의해 야기되는 오차가 디코딩될 다른 데이터 값들에 축적될 수 없기 때문이다. 양자화가 또한 로컬 공간 예측자들과 함께 사용될 수 있지만, 이러한 경우에, 알고리즘은, 양자화 오차가 누적되기 시작하지 않도록, 다음 값을 예측할 때 양자화 오차를 고려할 필요가 있다.

[0030] 공지된 유형들의 Delta 코딩은 주어진 현재 값으로부터 예측 값을 감소시키는 것 - 로컬 공간 예측자를 사용하는 것에 의해 달성됨 - 에 의해 구현되고, 이 값들의 차이로부터 플러스 값과 마이너스 값 둘 다가 생길 수 있고, 따라서 차이 값과 함께 부호가 항상 전달될 필요가 있다. ODelta 코딩(부록 1을 참조)에서는, 현재 값과 예측 값의 차이 또는 합이 사용될 수 있고, 더욱이 값들의 전달이 값들을 래핑하는 것을 사용하여 수행되며, 따라서 결과적으로 값들이 항상 플러스이다.

[0031] 앞서 언급된 바와 같이, IDelta 방법, DDelta 방법 및 PDelta 방법은 공지된 Delta 방법은 물론, 예측 값이 래핑을 사용하여 전달되는, 부록 1에 기술된 바와 같은 이전의 ODelta 방법과 상이하다. 게다가, 부록 1에 기술된 ODelta 방법에서는, 사용될 예측자가 하나의 값만을 이용하는 로컬 공간 예측자로 제한되고, 항상 양자화 없이 사용된다. 그렇지만, 다른 종류들의 예측자들 및/또는 양자화를 또한, 심지어 임의로 앞서 언급된 ODelta 방법과 결합하여, 이용하고 있는 본 개시내용의 실시예들에 대해서는 이러한 제한들이 필요하지 않다. IDelta 방법, DDelta 방법 또는 PDelta 방법에서의 래핑을 이용하는 원래의 ODelta 방법을 사용하는 것이 또한 실현가능한데, 그 이유는 그것이 종종 양호한 해결책이고, 어떤 경우에, 그것이 심지어 이 앞서 언급된 IDelta 방법, DDelta 방법 또는 PDelta 방법보다 더 나은 해결책이기 때문이다. 이러한 이유는 래핑을 이용하는 원래의 ODelta 방법이 종종 플러스 차이와 마이너스 차이 둘 다를 동일한 래핑된 값/심볼로 표현할 수 있기 때문이다. 래핑을 이용하는 원래의 ODelta 방법은 플러스 값과 마이너스 값이, 값 범위 및 예측 값에 기초하여, 나중에 서로 구별될 수 있는 방식으로 이것을 수행하는 반면, 이 앞서 언급된 IDelta 방법, DDelta 방법 및 PDelta 방법에서는, 0 값과 플러스 또는 마이너스 차이 값들만이 전달되고, 따라서 그와 관련하여 다른 구별이 필요하지 않다.

[0032] 임의로, 사용되는 예측 또는 어쩌면 사용되는 양자화에 관한 정보가 종종 방법 선택 정보와 함께 전달된다. 유익하게도, 양자화에 관한 정보가 종종 데이터 시퀀스 전체에 대해, 하나의 양자화 값 또는 하나의 품질 값을 사용하여, 한꺼번에 전달될 수 있다. 이러한 방법들의 예들은, 예를 들어, IDeltaBlockFromChannel0, PDeltaChannelR_2, DDeltaFrame_4, ODeltaBlockMode 및 DDeltaPacketPrevQ70이다. 이 방법들 중 첫 번째 방법 - 즉, IDeltaBlockFromChannel0 - 은 주어진 대응하는 블록에서의 채널 0 값들과 관련하여 플러스 차이 값들만을 전달한다 - 즉, 현재 코딩되고 있는 데이터 블록에서의 각각의 데이터 값이 예측으로서 사용되는 블록의 채널 0에서 대응하는 위치에 있는 값보다 크거나 적어도 그와 같다 -.

[0033] 이 방법들 중 두 번째 방법인 PDeltaChannelR_2는, 2에 의해 추가로 양자화되는, 페데스탈 값과 그에 뒤따르는 플러스 차이 값들 - 즉, 현재 채널과 R 채널 사이의 차이들 - 을 전달한다. DDeltaFrame_4라고 지칭되는 방법은, 예를 들어, 점차적으로 어두워지는 영상에 아주 적합하다. 이는, 4에 의해 양자화되는, 이전 프레임과 비교하여 마이너스 값 변화들을 전달한다. ODeltaBlockMode라고 지칭되는 방법은 현재 블록 영역에 대해, 모드 값과 비교하여, 래핑된 차이 값들을 전달한다. 임의로, 이 모드 값은 각각의 블록에 대해 개별적으로 송신/전달될 수 있거나, 예측 모드 값으로서, 데이터 채널 전체의 모드 값 또는 데이터 프레임 전체의 모드 값이 사용될 수 있다. DDeltaPacketPrevQ70은, 동일한 크기를 가졌던 이전 패킷과 비교하여, 현재 패킷에 대한 마이너스 데이터 값들을 전달하고, 델타 값들에 대한 양자화는 품질 인자 70을 사용하여 정의된다.

[0034] 앞서 제시된 예들은, 첨부된 청구항들에 의해 한정되는 바와 같은, 본 발명의 보호 범위를 제한하는 것으로 해석되지 않는데, 그 이유는 본 개시내용에 따른 방법의 다양한 다른 실시예들을 기술하기 위해 많은 다른 유사한 방법들이 사용될 수 있기 때문이다. 임의로, 데이터 값 한계들이 보통 최대 값들이기 때문에, 이 방법들 모두와 함께, 데이터 값 한계들이 전달될 수 있으며, 따라서 전달되는 데이터가 가능한 한 효율적으로 압축될 수 있다.

[0035] 본 개시내용은, 인코딩 알고리즘들 자체의 동작을 방해하지 않으면서, 데이터를 부분적 방식으로 인코딩하기 위

해 공지된 인코딩 방법들을 효율적으로 사용하는 대안의 방식을 정의하고 기술한다. 이와 같이, 본 개시 내용에 기술된 실시예들의 방법들은, 예를 들어, 유익하게도 다른 공지된 방법들과 함께 사용될 수 있거나, 공지된 코딩 방법들을 대체할 수 있다. 본 개시내용의 방법들은 본 개시내용의 실시예들의 이 방법들의 동작 후에 임의의 엔트로피 인코더에 대한 보다 낮은 엔트로피를 보장하기 위해 상이한 예측자들 그리고 임의로 양자화기들을 사용한다.

[0036] 임의로, 인코더는 인코딩된 데이터(E2)를 발생시키기 위해 입력 데이터(D1)를 인코딩할 때 적어도 하나의 양자화 알고리즘을 이용하기 위해 동작가능하고, 여기서 적어도 하나의 양자화 알고리즘으로 인해 인코더가 입력 데이터(D1)의 손실 인코딩을 제공하게 된다.

[0037] 임의로, 인코더는 입력 데이터(D1)에 존재하는 상이한 데이터 구조들의 데이터를 인코딩하는 데 상이한 알고리즘들을 이용하기 위해 동작가능하다.

[0038] 임의로, 인코더는 입력 데이터(D1)를 블록 단위로 인코딩할 때 RD 최적화를 이용하기 위해 동작가능하다. 게다가 임의로, 인코더에서, 하기 식의 값(V)을 최소화하기 위해 RD 최적화가 인코더 내에서 계산되고:

[0039] $$V = D + \lambda * R$$

[0040] 여기서 왜곡(D)은, 예를 들어, 입력 데이터(D1)와 인코딩된 데이터(E2)로 인코딩된 그리고 디코딩된 데이터(D3)로 디코딩된 입력 데이터(D1)의 표현 사이의 제곱 오차(SE)의 합이며, 여기서 레이트(rate)(R)는, 예를 들어, 비트로서 측정되는 인코딩된 데이터의 양을 나타낸다.

[0041] 임의로, 인코더에서, 적어도 하나의 Delta 인코딩 알고리즘(ODelta, DDelta, IDelta, PDelta) 및/또는 적어도 하나의 엔트로피 인코딩 알고리즘이 DC 방법, 슬라이드 방법, 멀티레벨 방법, DCT 방법, 라인 방법, 스케일 방법, 데이터베이스 방법, 범위(Range) 코딩, Huffman 코딩, RLE 코딩, SRLE 코딩 중 적어도 하나를 이용하기 위해 동작가능하다.

[0042] 임의로, 인코더는 YUV 채널들, BGR 채널들 중 적어도 하나에 대응하는 데이터 구조들을 포함하는 입력 데이터(D1)를 인코딩하기 위해 동작가능하다. 게다가 임의로, 인코더는 채널들의 데이터를 Y, U, V 순서로 또는 G, B, R 순서로 인코딩하기 위해 동작가능하다. 알파 채널 - 즉, 투명도 채널 - 이 또한 개별적으로 또는 다른 채널들과 함께 코딩될 수 있다. 그에 대응하여, 코딩되는 데이터가 오디오 데이터일 수 있고, 이 경우에, 예를 들어, 비트 깊이 8, 16 또는 24를 가질 수 있는 사운드 진폭 값들이 채널마다 개별적으로 또는 많은 채널들이 한꺼번에 코딩될 수 있다. 따라서, 코딩되는 데이터가 오디오, 영상들, 비디오, 게놈 데이터, 측정 결과들, 텍스트, 이진 또는 기타인지에 관계없이, 코딩되는 데이터의 비트 깊이가, 예를 들어, 데이터 요소마다 1 내지 256 비트에 대해 변할 수 있다.

[0043] 임의로, 인코더는 인코딩된 데이터(E2)를 발생시키기 위해 입력 데이터(D1)를 인코딩하는 데 인코더에 의해 이용되는 하나 이상의 인코딩 알고리즘들을 나타내는 이러한 데이터를 인코딩된 데이터(E2)에 포함시키기 위해 동작가능하다.

[0044] 제2 양태에 따르면, 대응하는 인코딩된 데이터(E2)를 발생시키기 위해 입력 데이터(D1)를 인코딩하는 데 인코더를 사용하는 방법이 제공되고, 본 방법은,

[0045] (i) 입력(D1)을 처리하고 적어도 그 일부분의 원래 값들을, 적어도 하나의 Delta 인코딩 알고리즘(ODelta, DDelta, IDelta, PDelta)을 사용하여, 원래 값들의 값 범위에 비해 증가되지 않은 값 범위를 사용하여 표현되는 델타 값들로 인코딩하기 위해 인코더를 사용하는 단계;

[0046] (ii) 입력 데이터(D1)의 하나 이상의 후속 부분들을 인코딩하는 데 사용하기 위한 하나 이상의 예측자들을 발생시키기 위해 인코더를 사용하는 단계; 및

[0047] (iii) 인코딩된 데이터(E2)를 발생시키기 위해 적어도 하나의 Delta 인코딩 알고리즘에 의해 발생된 데이터 및 하나 이상의 예측자들을 적어도 하나의 엔트로피 인코딩 알고리즘을 이용하는 것에 의해 인코딩하기 위해 인코더를 사용하는 단계를 포함하고, 여기서 하나 이상의 예측자들은

[0048] (i) 하나 이상의 시간 예측자들;

[0049] 삭제

[0050] (ii) 양자화가 적용된 하나 이상의 로컬 공간 예측자들; 및

[0051] (iii) 미리 계산된 값들을 이용하는 하나 이상의 로컬 공간 예측자들

[0052] 중 적어도 하나를 포함한다.

[0053] 임의로, 본 방법에서, 적어도 하나의 Delta 인코딩 알고리즘은 하기의 것들에 의해 구현된다:

[0054] (a) 하나 이상의 대응하는 인코딩된 시퀀스들을 발생시키기 위해 차분 및/또는 합 인코딩의 형태를 입력 데이터(DA1)에 적용하기 위해 데이터 처리 장치를 사용하는 것; 및

[0055] (b) 인코딩된 출력 데이터(DA2 또는 DA3)(= ODelta)를 발생시키기 위해 하나 이상의 대응하는 인코딩된 시퀀스들에 최대 값에서의 랩어라운드 및/또는 최소 값에서의 랩어라운드를 적용하기 위해 데이터 처리 장치를 사용하는 것.

[0056] 임의로, ODelta 방법(부록 1을 참조)을 하기의 방식으로 사용하는 것이 또한 가능하다. 주어진 원래 값과 대응하는 예측된 값 사이의 차이가 항상 0이거나 플러스인 경우, 또는 원래 값과 예측된 값 사이의 차이가 0이거나 마이너스인 경우, 양자화가 이용되든 그렇지 않든 간에, 코딩 방법 정보와 함께 델타 값들의 부호 - 즉, 부호 비트 - 만을 표현하고 전달하는 것이 가능하다. 이와 같이, 이러한 경우에, 래핑이 전혀 필요하지 않을 것인데, 그 이유는, 래핑이 없더라도, 값들이 주어진 비트 카운트에 항상 들어갈 것이기 때문이다.

[0057] 이러한 방식으로 사용되는 방법들은 편리하게도, 예를 들어, IDelta("Increment Delta") 방법 및 DDelta("Decrement Delta") 방법이라고 지칭될 수 있다. IDelta 방법은 따라서 플러스 델타 값들만을 전달하고, DDelta 방법은 마이너스 델타 값들만을 전달하지만, 종종 유리하게도 부호 - 즉, 부호 비트 - 를 스와핑한다. 앞서 기술된 방법들 둘 다는, 물론, 또한 변하지 않은 값들을 0 심볼/값과 함께, 이러한 변하지 않음이 양자화를 사용해 또는 양자화를 사용함이 없이 달성되었는지에 관계없이, 항상 전달하고; 즉, 이러한 경우에, 값들 사이의 차이는 사용되는 양자화보다 더 크지 않다.

[0058] 더욱이, 임의로, 델타 값들을 페데스탈 값과 함께 사용하는 것이 또한 가능하다. 이것은 마이너스 델타 값과 플러스 델타 값 둘 다가 있지만, 그들의 절대 값들이 코딩되는 데이터의 동적 범위 및 비트 깊이와 관련하여 작은 경우, 양자화되든 그렇지 않든 간에, 변화의 페데스탈 값 - 즉, 가장 큰 마이너스 변화 - 을 전달하는 것이 때때로 유익할 것임을 의미한다. 그 후에, 앞서 언급된 IDelta 방법에서와 같이, 플러스 변화 값들만을 전달하는 것이 가능하다. 이러한 페데스탈 방법은, 예를 들어, "PDelta 방법"이라고 지칭될 수 있다. 종종, PDelta 방법이 사용될 때는, 부록 1에 기술되는 바와 같은 원래의 ODelta 알고리즘이 사용될 때보다 데이터의 동적 범위/비트 깊이가 더 많이 감소될 수 있다. 즉, 데이터가 보다 적은 비트들을 사용하여 표현될 수 있는데, 그 이유는 데이터의 최대 값이 보다 작을 것이고 따라서 가능한 가장 작은 값과 가능한 가장 큰 값 사이의 차이가 보다 작을 것이기 때문이다. 이러한 이유는 ODelta 방법이 나오는 값들의 동적 범위 - 즉, 전달될 ODelta 값들의 값 범위 - 를 항상 표현하고 전달해야만 하기 때문이다.

[0059] 앞서 기술된 IDelta 방법, DDelta 방법 및 PDelta 방법은 사용되고 있는 예측이, 예를 들어, 이전 블록, 채널, 영상 또는 어떤 다른 결정된 값들의 세트이고, 이 방법들을 사용하기 전에 선언되는 이러한 해결책들에서 사용하기에 유익하다. 따라서, 코딩되고 있는 값과 예측 값 사이의 차이가 단 한번에 쉽게 결정되고, 따라서 값 또는 차이에 대한 예측이 코딩되고 있는 다른 값들에 전혀 의존하지 않는다. 이러한 종류들의 예측 해결책들은 또한 양자화와 함께 사용되는 데 특히 아주 적합한데, 그 이유는, 이 경우에, 차이 값들의 양자화가 디코딩될 하나의 개별 데이터 값에만 영향을 줄 것이고 따라서 양자화에 의해 야기되는 오차가 디코딩될 다른 데이터 값들에 축적될 수 없기 때문이다.

[0060] 임의로, 사용되는 예측 또는 어쩌면 사용되는 양자화에 관한 정보가 종종 방법 선택 정보와 함께 전달된다. 유익하게도, 양자화에 관한 정보가 종종 데이터 시퀀스 전체에 대해, 하나의 양자화 값 또는 품질 값을 사용하여, 한꺼번에 전달될 수 있다. 이러한 방법들의 예들은, 예를 들어, 편리하게도 IDeltaBlockFromChannel0, PDeltaChannelR_2, DDeltaFrame_4, ODeltaBlockMode 및 DDeltaPacketPrev라고 지칭된다. 이 방법들 중 첫 번째 방법 - 즉, IDeltaBlockFromChannel0 - 은 대응하는 블록에서의 채널 0 값들과 관련하여 플러스 차이 값들만을 전달한다. 즉, 현재 코딩되고 있는 데이터 블록에서의 각각의 데이터 값은 예측으로서 사용되는 블록의 채널 0에서 대응하는 위치에 있는 값보다 크거나 적어도 그와 같다.

[0061] 이 방법들 중 두 번째 방법 - 즉, PDeltaChannelR_2는, 2에 의해 추가로 양자화되는, 페데스탈 값과 그에 뒤따르는 플러스 차이 값들 - 즉, 현재 채널과 R 채널 사이의 차이들 - 을 전달한다. DDeltaFrame_4 방법은, 예를

들어, 점차적으로 어두워지는 영상에 대해 아주 적합하다. 이는, 4에 의해 양자화되는, 이전 프레임과 비교하여 마이너스 값 변화들을 전달한다. ODeltaBlockMode 방법은 현재 블록 영역에 대해, 모드 값과 비교하여, 래핑된 차이 값들을 전달한다. 임의로, 이 모드 값은 각각의 블록에 대해 개별적으로 송신/전달될 수 있거나, 예측 모드 값으로서, 데이터 채널 전체의 모드 값 또는 데이터 프레임 전체의 모드 값이 사용될 수 있다.

[0062] DDeltaPacketPrev 방법은, 유사한 크기를 갖는 이전 패킷과 비교하여, 현재 패킷에 대한 마이너스 데이터 값들을 전달한다.

[0063] 임의로, 본 방법은 인코더가 인코딩된 데이터(E2)를 발생시키기 위해 입력 데이터(D1)를 인코딩할 때 적어도 하나의 양자화 알고리즘을 이용하도록 어레인지하는 단계를 포함하고, 여기서 적어도 하나의 양자화 알고리즘으로 인해 인코더가 입력 데이터(D1)의 손실 인코딩을 제공하게 된다.

[0064] 임의로, 본 방법은 인코더가 입력 데이터(D1)에 존재하는 상이한 데이터 구조들의 데이터를 인코딩하는 데 상이한 알고리즘들을 이용하도록 어레인지하는 단계를 포함한다.

[0065] 임의로, 본 방법은 인코더(100)가 입력 데이터(D1)를 블록 단위로, 패킷 단위로, 채널 단위로, 뷰 단위로 또는 프레임 단위로 인코딩할 때 RD 최적화를 이용하도록 어레인지하는 단계를 포함한다. 게다가 임의로, 본 방법에서, 하기 식의 값(V)을 최소화하기 위해 RD 최적화가 인코더(100) 내에서 계산되고:

[0066] $$V = D + \lambda * R$$

[0067] 여기서 왜곡(D)은 입력 데이터(D1)와 인코딩된 데이터(E2)로 인코딩된 그리고 디코딩된 데이터(D3)로 디코딩된 입력 데이터(D1)의 표현 사이의 제곱 오차(SE)의 합이며, 여기서 레이트(R)는, 예를 들어, 비트로서 측정되는 인코딩된 데이터의 양을 나타낸다.

[0068] 임의로, 본 방법에서, 적어도 하나의 Delta 인코딩 알고리즘(ODelta, DDelta, IDelta, PDelta) 및/또는 적어도 하나의 엔트로피 인코딩 알고리즘이 DC 방법, 슬라이드 방법, 멀티레벨 방법, DCT 방법, 라인 방법, 스케일 방법, 데이터베이스 방법, 범위(Range) 코딩, Huffman 코딩, RLE 코딩, SRLE 코딩 중 적어도 하나를 이용하기 위해 동작가능하다.

[0069] 임의로, 본 방법은 인코더(100)가 YUV 채널들, BGR 채널들 중 적어도 하나에 대응하는 데이터 구조들을 포함하는 입력 데이터(D1)를 인코딩하도록 어레인지하는 단계를 포함한다. 게다가 임의로, 본 방법은 인코더(100)가 채널들의 데이터를 Y, U, V 순서로 또는 G, B, R 순서로 인코딩하도록 어레인지하는 단계를 포함한다. 알파 채널 - 즉, 투명도 채널 - 이 또한 개별적으로 또는 다른 채널들과 함께 코딩될 수 있다. 그에 대응하여, 코딩되는 데이터가 오디오 데이터일 수 있고, 이 경우에, 예를 들어, 비트 깊이 8, 16 또는 24를 가질 수 있는 사운드 진폭 값들이 채널마다 개별적으로 또는 많은 채널들이 한꺼번에 코딩될 수 있다. 따라서, 코딩되는 데이터가 오디오, 영상들, 비디오, 게놈 데이터, 측정 결과들, 텍스트, 이진 또는 기타인지에 관계없이, 코딩되는 데이터의 비트 깊이가, 예를 들어, 데이터 요소마다 1 내지 256 비트에 대해 변할 수 있다.

[0070] 임의로, 본 방법은 인코더가 인코딩된 데이터(E2)를 발생시키기 위해 입력 데이터(D1)를 인코딩하는 데 인코더에 의해 이용되는 하나 이상의 인코딩 알고리즘들을 나타내는 이러한 데이터를 인코딩된 데이터(E2)에 포함시키도록 어레인지하는 단계를 포함한다.

[0071] 제3 양태에 따르면, 대응하는 디코딩된 데이터(D3)를 발생시키기 위해 인코딩된 데이터(E2)를 디코딩하는 디코더가 제공되고, 디코더는 제1 양태에 따른 인코더에서 구현되는 인코딩 알고리즘들의 역을 수행하기 위해 동작가능하다.

[0072] 따라서, 대응하는 디코딩된 데이터(D3)를 발생시키기 위해 인코딩된 데이터(E2)를 디코딩하는 디코더가 제공되고, 디코더는 처리된 데이터를 발생시키기 위해 인코딩된 데이터(E2)에 적어도 하나의 엔트로피 디코딩 알고리즘을 적용하는 것에 의해 인코딩된 데이터(E2)를 처리하기 위해, 그리고 디코딩된 데이터(D3)를 발생시키기 위해 처리된 데이터를 디코딩하는 데 적어도 하나의 Delta 디코딩 알고리즘(역 ODelta, 역 DDelta, 역 IDelta, 역 PDelta)과 결합하여 하나 이상의 예측자들을 사용하기 위해 동작가능하며,

상기 처리된 데이터는 상기 인코딩된 데이터(E2)가 발생된 원래 데이터의 값 범위에 비해 증가되지 않은 값 범위를 사용하여 표현되는 델타 값들을 포함하고,

[0073] 여기서 하나 이상의 예측자들은

[0074] (i) 하나 이상의 시간 예측자들;

[0075] 삭제

[0076] (ii) 양자화가 적용된 하나 이상의 로컬 공간 예측자들; 및

[0077] (iii) 미리 계산된 값들을 이용하는 하나 이상의 로컬 공간 예측자들

[0078] 중 적어도 하나를 포함한다.

[0079] 미리 계산된 값들과 관련하여, 값들은 데이터에 대한 프로세스를 수행하기 전에, 공간 위치에 기초하여, 결정될 수 있는 임의의 값들일 수 있고; 즉, 부록 1에 기술된 바와 같은 ODelta 인코딩에서와 같이 로컬 공간 예측자들이 아니다. 이러한 미리 계산된 값들의 예들은 인코딩될 현재 데이터 블록에 대해 좌측에 또는 위쪽에 있는 이전 블록, 어떤 이전 채널, 뷰 또는 프레임에서의 동일한 데이터 블록의 위치, 또는 주어진 데이터 프레임 내에서의 내부 움직임 예측을 포함한다.

[0080] 임의로, 디코더는 방법을 나타내는 정보를 수신하기 위해 동작가능하고, 이어서 계속하여 방법에 따라 예측을 실행하고 다양한 다른 방법들에 따라 디코딩될 값들을 계산하며; 환언하면, 사용된 방법의 정보가 디코더에 수신되고; 그 정보에 기초하여, 페데스탈 방법을 사용할지(PDelta), 또는 플러스 차이 값들만이 나올지(IDelta 또는 PDelta), 또는 마이너스 차이 값들이 나올지(DDelta), 또는 그 마이너스 값들과 플러스 값들을 구분하기 위해 정규의 랩(wrap) 및 한계 값들이 사용되는지(ODelta)가 알려질 수 있다.

[0081] 임의로, 동작을 설명하면, 디코더는 방법을 나타내는 정보를 수신하고, 방법에 따라 예측을 실행하며, 다양한 다른 방법들에 따라 디코딩될 값들을 계산하고 - 즉, 사용된 방법의 정보가 수신됨 -, 그 정보에 기초하여, 페데스탈 방법을 사용할지(PDelta), 또는 플러스 차이 값들만이 나올지(IDelta 또는 PDelta), 또는 마이너스 차이 값들이 나올지(DDelta), 또는 그 마이너스 값들과 플러스 값들을 구분하기 위해 정규의 랩(wrap) 및 한계 값들이 사용되는지(ODelta)가 알려질 수 있다. 예측이 단 한번에 결정될 수 있거나, ODelta(부록 1을 참조)에서와 같이, 프로세스가 진행함에 따라 결정될 수 있다. 프로세스 동안 예측을 결정하는 원래의 ODelta 유형 절차는 "로컬 공간 예측"이라고, 또한 "로컬 예측"이라고 지칭되고, 블록에 대한 예측이 사전에 단 한번에 결정되는, 이 새로운 예측 해결책은 "시간 예측" 또는 "공간 예측"이라고 지칭된다. 앞서 언급된 블록들, 채널들, 뷰들 및 프레임들에 부가하여, 움직임 추정이 또한 "시간 예측"인 것으로 간주된다.

[0082] 제4 양태에 따르면, 대응하는 데이터(D3)를 발생시키기 위해 인코딩된 데이터(E2)를 디코더에서 디코딩하는 방법이 제공되고, 디코딩하는 방법은 제2 양태에 따른 방법의 역을 디코더에서 실행하는 단계를 포함한다.

[0083] 대응하는 데이터(D3)를 발생시키기 위해 인코딩된 데이터(E2)를 디코더(120)에서 디코딩하는 방법은 디코딩하는 방법이 처리된 데이터를 발생시키기 위해 인코딩된 데이터(E2)에 적어도 하나의 엔트로피 디코딩 알고리즘을 적용하는 것에 의해 인코딩된 데이터(E2)를 처리하는 단계, 및 디코딩된 데이터(D3)를 발생시키기 위해 처리된 데이터를 디코딩하는 데 적어도 하나의 Delta 디코딩 알고리즘(= 역 ODelta, 역 DDelta, 역 IDelta, 역 PDelta)과 결합하여 하나 이상의 예측자들을 사용하는 단계를 포함하고, 처리된 데이터는 인코딩된 데이터(E2)가 발생된 원래 데이터의 값 범위에 비해 증가되지 않은 값 범위를 사용하여 표현되는 델타 값들을 포함하고, 여기서 하나 이상의 예측자들은

[0084] (i) 하나 이상의 시간 예측자들;

[0085] 삭제

[0086] (ii) 양자화가 적용된 하나 이상의 로컬 공간 예측자들; 및

[0087] (iii) 미리 계산된 값들을 이용하는 하나 이상의 로컬 공간 예측자들

[0088] 중 적어도 하나를 포함하는 것을 특징으로 한다.

[0089] 미리 계산된 값들은, 로컬 예측자들과 마찬가지로, 이상에서 설명되었다.

[0090] 제5 양태에 따르면, 컴퓨터 판독가능 명령어들이 저장되어 있는 비일시적 컴퓨터 판독가능 저장 매체를 포함하는 컴퓨터 프로그램 제품이 제공되고, 컴퓨터 판독가능 명령어들은 제2 양태 또는 제4 양태에 따른 방법을 실행

하기 위해 처리 하드웨어를 포함하는 컴퓨터화된 디바이스에 의해 실행가능하다.

[0091] 본 발명의 특징들이 첨부된 청구항들에 의해 한정되는 바와 같은 본 발명의 범주를 벗어남이 없이 다양한 조합들로 결합될 수 있다는 것을 잘 알 것이다.

도면의 간단한 설명

[0092] 이제부터, 단지 예로서, 이하의 도면들을 참조하여 본 개시내용의 실시예들이 기술될 것이다.

도 1은 본 개시내용의 일 실시예의 예시적인 상위 레벨 구조의 개략도.

도 2는 본 개시내용의 일 실시예의 6개의 블록들의 예시적인 채널의 개략도.

도 3은 예시적인 블록과 그의 연관된 합성의 개략도.

도 4는 본 개시내용의 일 실시예에 따른, 예측을 위한 이웃 데이터 값들의 개략도.

도 5는 본 개시 내용에 따른 인코더, 디코더 및 코덱의 개략도.

도 6 내지 도 8은 ODelta 방법들에 관한 부록 1에서의 개시내용을 지원하는 것에 관한 도면.

첨부 도면들에서, 밑줄친 번호는 밑줄친 번호가 위치되어 있는 항목 또는 밑줄친 번호가 인접해 있는 항목을 나타내기 위해 이용된다. 밑줄이 없는 번호는 밑줄이 없는 번호를 항목에 연결시키는 라인에 의해 식별되는 항목에 관한 것이다. 번호가 밑줄이 그어져 있지 않고 연관된 화살표를 수반할 때, 밑줄이 없는 번호는 화살표가 가리키는 일반 항목을 식별하는 데 사용된다.

발명을 실시하기 위한 구체적인 내용

[0093] 이하에서 본 개시내용의 실시예들을 기술할 때, 표 1에 제공되는 바와 같은 약어들이 이용된다:

[0094] [표 1]: 실시예들을 기술하는 데 이용되는 약어들의 상세

표 1

[0095]

약어	상세
1D	1차원, 예를 들어, 신호 또는 데이터 패킷을 지칭함
2D	2차원, 예를 들어, 신호 또는 데이터 패킷을 지칭함
3D	3차원, 예를 들어, 신호 또는 데이터 패킷을 지칭함
블록	디지털 데이터로부터의 다수의 데이터 요소들, 즉 디지털 데이터의 일부
CRC	순환 중복 검사(cyclic redundancy check)
코덱	디지털 데이터에 대한 인코더 및 디코더
DB	RAM 기반 또는 ROM 기반 메모리에서의 데이터베이스
DC	영상의 DC 성분, 즉 평균 밝기에 대응하고 영상에 존재하는 가장 낮은 공간 주파수를 나타내는 영상 평균
Delta 코딩	Delta 코딩은 전체 데이터 파일들보다는 순차적 데이터 사이의 차분들의 형태로 데이터를 저장하거나 전송하는 방식임
ISP	내부 스위치 제공자(Internal Switch Provider)
LAN	근거리 통신망(Local Area Network)
패킷	예를 들어, 복수의 데이터 블록들을 포함하는 데이터의 보디
RAM	랜덤 액세스 메모리(Random Access Memory)
RD	레이트-왜곡(Rate-Distortion)
RLE	런 길이 인코딩(Run-Length Encoding)
ROI	관심 영역(Region of Interest)
ROM	판독 전용 메모리(Read Only Memory)
SRLE	분할 런 길이 인코딩(Split Run-Length Encoding)
VLC	가변 길이 코드(Variable-Length Code)
XOR	배타적 OR(Exclusive Or)(논리 함수)

[0096] 개요를 말하면, 본 개시내용의 실시예들은 향상된 형태의 인코더 및 디코더, 그리고 연관된 향상된 데이터 인코

딩 및 디코딩 방법들에 관한 것이다. 본 개시내용의 실시예들은 이하에서 보다 상세히 기술될 것이고 본 개시내용의 실시예들에서 추가로 향상된, 예를 들어, ODelta 인코딩 방법과 같은, Delta 인코딩 방법들에 기초한다. 다양한 서로 상이한 공간 및 시간 예측 방법들을 사용하는 것에 의해 오디오 패킷들, 영상 블록들, 인터넷 데이터 패킷들, 채널들, 비디오 프레임들 등을 코딩하기 위한 Delta 코딩 방법들이 제공되고, 임의로 양자화기가 이용된다. 본 개시내용의 인코딩 방법들은 무손실 코딩 및 손실 코딩 둘 다에 적당하고, 하기의 3개의 주요 기능 요소들을 포함한다:

[0097] (i) 예측;

[0098] (ii) ODelta, 또는 임의의 양자화기를 갖는 PDelta, IDelta 또는 DDelta와 같은 유사한 연산자; 및

[0099] (iii) 엔트로피 인코딩.

[0100] 본 개시내용의 인코딩 방법들 및 대응하는 디코딩 방법들은 도 5를 참조하여 나중에 보다 상세히 기술될 것인 바와 같이, 각각, 인코더들 및 디코더들에서 사용될 수 있다.

[0101] 첨부된 부록 1에서, DPCM 스타일의 사용을 위한 ODelta 연산자의 설명이 제공된다. 본 개시내용에서, 부록 1 이전의 텍스트에서, 그 ODelta 연산자들이 상이한 (로컬) 공간, 시간 또는 조합 예측 방법들을 사용하도록 수정된다. 본 개시내용의 방법들은 개개의 데이터 프레임들에 대한, 개개의 데이터 채널들에 대한, 개개의 데이터 블록들에 대한 또는 개개의 데이터 패킷들 등에 대한 데이터 시퀀스 전체에 대해 사용될 수 있도록 고안되었다. 더욱이, 본 개시내용의 방법들은 선택된 예측 방법들, 선택된 ODelta 연산자들, 그리고 선택된 잔차 코딩 및 압축 방법들에 기초한 몇 개의 서로 상이한 코딩 방법들을 제공한다.

[0102] 많은 다른 코딩 방법들이 본 개시내용의 방법들과 결합하여 임의로 사용되고, 이러한 방법들이 유리하게도 특허 문서 GB2503295 - 이로써 참고로 포함됨 - 에 기술되어 있는 부록 2에 기술된 바와 같은 블록 인코더, 및 특허 문서 GB2505169 - 이로써 참고로 포함됨 - 에 기술되어 있는 부록 3에 기술된 바와 같은 블록 디코더와 함께 사용된다. 예를 들어, 주어진 데이터 채널이 블록 단위로 코딩되어야 할 때, 주어진 데이터 블록을 인코딩하기 위해 이용할 최상의 코딩 방법의 선택은, 예를 들어, RD 최적화를 사용하여 행해진다. RD 최적화는 하기 식의 값(V)을 최소화하고:

[0103] $$V = D + \lambda * R$$ <수학식 1>

[0104] 여기서 왜곡(D)은, 전형적으로, 원래 값들과 디코딩된 값들 사이의 제곱 오차(SE)의 합이고, 여기서 인코딩된 데이터 값들의 레이트(R)는 전형적으로 비트에 의해 측정된다. 본 개시내용에 따른 방법들에서, 많은 다른 코딩 방법들 - 예를 들어, DC 방법, 슬라이드 방법, 멀티레벨 방법, DCT 방법, 라인 방법, 스케일링 방법 및 데이터베이스 방법 등 - 이 또한 임의로 사용된다.

[0105] 본 개시내용에 따른 방법들은 유익하게도 인코딩될 입력 데이터(D1)에 존재하는 서로 상이한 데이터 구조들에 대해 사용된다. 예를 들어, 데이터 채널 전체 - 예를 들어, 평면 영상의 루미넌스 채널(luminance channel) - 가 임의로 본 개시내용에 따른 방법들에 의해 코딩된다. 본 개시내용에 따른 방법들은 간단하고, 예를 들어, 휴대 전화, 카메라, 및 유사한 것들과 같은 저전력 휴대용 전자 디바이스들에서 현재 이용되는 RISC(reduced instruction set computer) 프로세서들을 사용하는 디바이스들 및 시스템들에서 낮은 복잡도로 구현될 수 있다. 따라서, 채널 전체에 대한 본 개시내용의 방법들에 의해 제공되는 결과들이 다른 채널 코딩 방법들 - 예를 들어, 블랙/모드 값 채널, 프리즈 채널(freeze channel), 엔트로피 코딩된 원래 채널, 및 블록 인코더에 의한 채널 코딩 - 과 쉽게 비교될 수 있다. 최상의 채널 코딩 방법의 선택은 또한 유익하게도 RD 최적화에 의해 행해진다. 블록 인코더에 의한 채널 코딩 방법은 데이터 채널이 서로 상이한 코딩 방법들을 사용하여 블록 단위로 코딩되고, 이 경우에, 하나의 코딩 방법이 데이터 채널 전체에 대해 사용되지 않는다는 것을 의미한다.

[0106] 본 개시내용의 방법들은 임의로 손실 코딩 - 즉, 양자화된 잔차 값 코딩을 사용하거나 잔차 코딩을 사용하지 않음 - 에서 또는 무손실 코딩 - 즉, 영 잔차(zero residual) 또는 양자화되지 않은 잔차 값 코딩을 사용함 - 에서 사용될 수 있다. 인코더에서 그리고 디코더에서의 이전의 코딩된 및 디코딩된 값들 모두는, 본 개시내용에 따르면, 현재 또는 장래 데이터 값들의 예측을 위해 사용될 수 있다. 본 개시내용의 인코딩 방법들을 데이터에 적용하는 것에 의해 데이터가 무손실 코딩될 때, 무손실 코딩에서의 디코딩된 값들과 동일한, 이미 처리된 소스 값들이 또한 인코더에서의 현재 또는 장래 데이터 값들의 예측을 위해 사용될 수 있다.

[0107] ODelta 연산자의 가장 중요한 파라미터들은 highValue, lowValue 및 wrapValue(즉, 적어도 highValue -

lowValue + 1)이고; 이것은 부록 1에서 보다 상세히 설명된다. highValue 및 lowValue가 양자화를 이용하도록 정의되는 것이 또한 가능하다. 예를 들어, 원래 데이터는 0부터 255까지의 값들을 포함하지만, 최종 결과가 선택된 품질 인자(예를 들어, 30, 품질 값들의 값들이 '1'부터 '100'까지의 범위에 있을 때, 품질 값 '100'은 무손실 압축을 지칭함)를 갖는 값들 - 예를 들어, 0부터 78까지(상대 양자화 78/255) - 로 양자화되는 것이 요망된다. 데이터 오프셋들이 또한 ODelta 연산자를 이용하기 전에(사전 오프셋(pre-offset)) 또는 그 이후에(사후 오프셋(post-offset)) 사용될 수 있다. 더욱이, 엔트로피 인코딩이 유익하게도 ODelta 연산자 이후에 실행되는데, 그 이유는 그렇지 않으면 감소된 엔트로피가 데이터 인코딩에서 충분히 이용되지 않을 것이기 때문이다.

[0108] 앞서 언급된 IDelta 방법, DDelta 방법 및 PDelta 방법을 참조하면, wrapValue이, 부록 1에 기술된 바와 같은 ODelta 방법에서 필요했던 것처럼, 더 이상 결정될 필요가 없다. 따라서, 앞서 언급된 IDelta 방법, DDelta 방법 및 PDelta 방법에 관해서, - highValue' 및 lowValue'은, 방법 선택 정보 및 최종 양자화 정보(eventual quantization information)와 함께, 훨씬 더 중요한 역할을 한다. 이 경우에, highValue' 및 lowValue'가 최종적인 실제 데이터 값들의 값 범위를 더 이상 지칭하지 않고, 그 대신에 전달되는 차이 값들의 값 범위를 지칭한다. lowValue'은 또한 PDelta 방법의 페데스탈 값을 결정하는 데 사용될 수 있고, 이 경우에 전달되는 최대 데이터 값은 그 자체로서 highValue' - lowValue'의 결과이거나, 어쩌면 양자화에 의해 제한된다. 이 양자화는 제수(divisor)에 의해, 품질 파라미터에 의해 또는 원래 동적 범위에 대한 상대 변화로서 결정될 수 있다. 이러한 구현의 일 예는 원래 highValue 또는 가장 높은 차이 highValue'이 255이었고 값 범위가 상대 양자화 값 78/255에 의해 제한되고 감소된 전술한 바에 제공되어 있다. 상대 양자화의 양자화 값은, 이 예에서, 예를 들어, 값 78로서 또는 값 0.3059(즉, 78/255 미만)로서 전달될 수 있다.

[0109] 사용된 엔트로피 인코딩 방법은 유익하게도 범위 코딩 또는 SRLE 범위 코딩으로 선택되지만, 다른 엔트로피 인코딩 방법들 - 예를 들어, Huffman 코딩, RLE 코딩, SRLE 코딩 - 이 또한 사용될 수 있다. 부록 1에 기술된 ODelta 방법이 사용될 때, 예측 값은 바람직하게는 항상 이전 데이터 값이고, 첫 번째 예측 값은 선택된 초기화 방법에 의해 초기화될 필요가 있다.

[0110] 본 개시내용에 따른 방법들에서, 예측 값들은 이전 데이터 값만을 사용하는 것과 상이한 방식으로 선택될 수 있다. 본 개시 내용의 방법들에 따르면, 하나의 선택된 데이터 값 또는 다수의 데이터 값들로부터 계산된 값인 예측 값을 사용하는 것이 가능하다. 예를 들어, 값이 2개 이상의 이전 데이터 값들로부터(1D), 이웃에 있는 2개 이상의 이전 데이터 값들로부터(2D, 3D, ...), 이전 데이터 블록 또는 데이터 패킷에 있는 데이터 값/값들로부터, 이전 데이터 채널/채널들에 있는 데이터 값/값들로부터, 이전 데이터 프레임/프레임들에 있는 데이터 값/값들로부터, 이전에 언급된 데이터 값들의 임의의 조합, 및 기타로부터 계산될 수 있다.

[0111] 현재 데이터 값에 대한 예측 값이 계산될 때, 원래 데이터 값과 예측된 데이터 값 사이의 차이 또는 합이 OValue로서 계산된다. OValue는 QOValue로 양자화되거나 복사될 수 있고, 이어서, QOValue이 lowValue보다 더 낮거나 highValue보다 더 높은 경우, 랩어라운드(즉, wrapValue의 가산 및/또는 감산)를 행하는 ODelta 연산자에 주어질 수 있다.

[0112] 특히 값이 양자화될 때, 래핑 연산(wrapping operation)이 방법의 결과를 플러스 가산(또는 감산)으로부터 마이너스 가산(또는 감산)으로 또는 그 반대로 변경하지 않도록 랩어라운드와 역 랩어라운드(inverse wrap around)가 정확하게 실행될 수 있도록 양자화 레벨들이 설계되어야만 한다는 것을 잘 알 것이다. 랩어라운드는 결과를 명확히 보다 작은 절대 값으로도 명확히 보다 큰 절대 값으로도 변경해서는 안된다. 이것은, 예를 들어, 절대 데이터 값들에 대해 2개의 상이한 양자화기들이 사용되는 경우, 역 양자화 및 랩어라운드에서 데이터 값의 잘못된 해석을 피하기 위해, 작은 데이터 값들과 큰 데이터 값들이 중간 데이터 값들보다 더 작은 양자화기 값으로 양자화되어야만 한다는 것을 의미한다.

[0113] 다음에 도 1을 참조하면, 3D 비디오 콘텐츠가, 프레임, 뷰, 채널, 데이터 블록, 데이터 패킷 및 개개의 데이터 값과 같은, 상이한 데이터 구조들로 분할되는 방식이 개략적으로 도시되어 있다. 부가의 구조들 - 예를 들어, 프레임들의 그룹, 데이터 블록들의 그룹, 초기 데이터 블록(init data block)들, 및 데이터 슬라이스들 - 을 사용하는 것이 또한 임의로 가능하다. 이러한 서로 상이한 구조들 모두는, 본 개시내용에 따르면, 대응하는 인코딩된 데이터(E2)를 발생시키기 위해 입력 데이터(D1)를 처리 및 인코딩하는 동안 분리되어 있을 필요가 없다. 데이터의 순서가 변할 수 있지만, 도 1의 예에서, 주어진 채널에서의 블록들이 좌에서 우로 그리고 위에서 아래로 처리된다. 이미 처리된 - 즉, 손실 코딩을 구현할 때 인코딩된, 그리고 임의로 디코딩된 - 값들 모두는 또한 현재 및 장래 값들의 예측을 위해 사용될 수 있는데, 그 이유는 본 개시내용에 따른 인코더 및 대응하는 디코더 둘 다가 인코딩된 값들로부터 디코딩된 값들에 관한 정보를 가질 수 있기 때문이다.

[0114] 본 개시 내용에 따르면, 본 개시내용에 따른 방법들에서 시간 채널 예측에 대한 보다 나은 대안들을 가능하게 하기 위해 YUV 채널들을 Y, U, V 순서로 그리고 또한 BGR 채널들을 G, R, B 또는 G, B, R 순서로 인코딩하는 것이 종종 유익하다. 시간 채널 예측은 주어진 영상이 RGB 색 공간으로 인코딩될 때 전형적으로 아주 양호한 방법이다. 시간 채널 예측은 채널 상관을 상당히 감소시킨다. YUV 색 공간이 사용될 때, 그 색 공간의 속성들로 인해 채널 상관이 이미 크게 감소되어 있다. 더욱이, YUV 색 공간이 사용될 때, Y 채널은 정보의 대부분을 포함하고, 이는 U 채널과 V 채널이 보다 효율적으로 코딩될 수 있다는 것을 의미한다.

[0115] 본 개시내용의 방법들은 임의로 서로 상이한 데이터 구조들을 처리하는 데 이용가능한 별개의 하위방법들 - 즉, 알고리즘들 - 을 갖는다. 예를 들어, 색상 채널은 공간 예측 또는 시간 예측을 사용해 코딩될 수 있다. 상이한 유형들의 시간 예측 하위방법들이 또한 사용될 수 있다. 하나의 하위방법은, 예를 들어, 이전 프레임에서와 유사한 위치에 있는 동일한 채널 값을 사용하는 반면, 다른 하위방법은 이전 뷰에서 유사한 위치에 있는 동일한 채널 값을 사용하고, 또 다른 하위방법은, 예를 들어, 채널 2에 대한 값을 예측하기 위해, 예를 들어, 채널 0으로부터의 유사한 위치 값을 사용한다.

[0116] 시간 예측이 사용될 때, 현재 채널에서의 모든 값이 다른 프레임들, 뷰들 또는 채널들에서 이용가능한 예측 값을 갖기 때문에, 임의의 첫 번째 예측 값을 정의할 필요가 없다. 주어진 채널에서 공간적으로 이전에 위치된 유사한 데이터 블록들에 대해 "시간 종류의 예측(temporal kind of prediction)"이 또한 사용될 수 있다. 전형적으로, 이 종류의 방법이 사용될 때, 상이한 하위방법들의 양이 과도하게 증가되지 않도록, 2개의 데이터 블록 대안들만이 이용가능하다. 이것은 또한 "움직임 벡터" 종류의 블록 설명자(block descriptor)를 전송할 필요가 없다는 것을 의미한다. 이 종류의 데이터 블록 설명자가 또한 임의로 사용될 수 있지만, 전형적으로 설명자의 정확도는, 예를 들어, 데이터 블록이고 개개의 데이터 값이 아니며, 따라서 상이한 조합들의 양이 인트라 움직임 벡터 추정(intra-motion vector estimation)을 이용하는 것에 비해 크게 감소될 수 있다.

[0117] 전형적인 공지된 인트라/인터 움직임 추정 방법이 제공하는 정확도와 유사한 정확도를 갖는 대안의 이용가능한 알고리즘들의 조합이 사용되더라도, 본 개시내용에 따른 방법들을 이용하는 데 여전히 큰 이점이 있을 것인데, 그 이유는 본 개시내용의 방법들에 의해 행해지는 선택이 ODelta 인코딩을 이용하는 것에 의한 효율적인 잔차 코딩도 포함하기 때문이다. 이와 같이, 본 개시내용에 따른 방법들에서, 예측을 위한 개별적인 인트라/인터 움직임 추정을 그리고 이어서 별도로, 예를 들어, 잔차 코딩을 위한 DCT 방법을 사용할 필요가 없다.

[0118] 본 개시내용의 실시예들에서, 공간 예측은 서로 상이한 예측 값들을 임의로 사용한다. 이전 값 예측(X에 대한 A)이 ODelta 기법들(부록 1을 참조)로부터의 공지된 기술이다. 본 발명의 실시예에서 이용할 하나의 유익한 방법은 값 X에 대한 P = A + B - C와 같은 예측 값들을 사용한다. 많은 다른 예측 값들 - 예를 들어, PNG 문서에 기술된 2A - D 또는 PAETH 예측 - 이 또한 임의로 사용된다. 예측 값은 유익하게도 가능한 값들의 범위로 제한되거나 다른 방식으로 절단(truncate)된다. 예를 들어, 채널 0에서의 주어진 값이 어쩌면 0(lowValue)과 63(highValue) 사이의 값들을 제공받고 A = 60, B = 61, C = 52이면, P = A + B - C = 69는 63(= highValue)으로 클리핑(clip), 절단 또는 포화(saturate)된다.

[0119] X = 62이면, OValue는, 수정된 ODelta 연산자 방법 1에 의해, 62 - 63 = -1이다. 이 값은 무손실 코딩에서 양자화될 필요가 없고, 따라서 QOValue도 -1이다. 이제, -1이 lowValue(0)보다 더 작고, 그러면 63을 ODelta 값으로서 얻기 위해 wrapValue(64)가 QOValue에 가산될 필요가 있다는 것을 잘 알 것이다. 이 값은 값 X를 이 ChannelSpatialODeltaCoded 방법으로 인코딩하기 위해 엔트로피 인코딩을 위한 버퍼에 설정되어 있다. 유사한 처리가 다른 데이터 값들에 대해서도 수행되고, 모든 채널 값들이 처리될 때, 채널 0에 대한 출력 인코딩된 데이터 값들을 생성하기 위해, 그 후에 ODelta 값들의 버퍼의 내용이, 예를 들어, 범위 코딩 또는 SRLE(serial run-length encoding) 범위 코딩에 의해 압축된다.

[0120] 첫 번째 행이 처리될 때, 값 A만이 예측을 위해 이용가능하고, A + B - C를 예측 값으로서 사용할 때, 따라서 값 A가 예측 값으로서 직접 사용된다는 것을 잘 알 것이다. 이와 유사하게, 첫 번째 열에 대해, B 값만이 이용가능하고, 따라서 값 B가 예측 값으로서 사용된다. 첫 번째 값 - 예를 들어, 채널 0에서의 가장 위쪽, 가장 왼쪽 값 - 은 예측을 위해 사용될 수 있는 어떤 공간 값도 갖지 않는다. 첫 번째 예측 값에 대해 시간 값들을 사용하는 것이 가능하다 - 즉, 예를 들어, 이전 뷰에서 또는 이전 프레임에서 시간 값이 이용가능한 경우 -. 어떤 적당한 시간 예측 값도 이용가능하지 않은 경우, 예를 들어, 값 0 또는 중간범위 값(midrange value)((63 - 0 + 1) div 2 = 32, 또는 채널/뷰/프레임의 별도로 전달된 모드 값이 채널 0에 대한 첫 번째 예측 값으로서 사용될 수 있다. 방법에서 상이한 예측자들을 사용하고 심지어 어느 예측자가 인코딩된 데이터(E2)에서의 어느 프레임, 채널, 블록 또는 심지어 데이터 값에 대해 사용되는지에 관한 정보를 전달하는 것이 또한 가능하다.

[0121] 유사한 방법들이 또한, 예를 들어, 데이터 블록들에 대해 사용될 수 있다. 이하의 예는, 명확히 보다 작은 품질 요구사항들로, 값 4에 의한 양자화가 대응하는 인코딩된 값들에 대해 사용될 수 있게 하는 BlockChannel00DeltaCoded 방법을 사용하여 채널 2에서의 블록 2가 어떻게 코딩되는지를 나타내고 있다. 값 4에 의한 이 양자화는 lowValue가 0이고, highValue가 15이며 wrapValue가 16이라는 것을 의미한다. 이제, 현재 채널 2에서의 블록 2가, 예를 들어, 다음과 같은 값들을 포함한다:

[0122] 45, 48, 50, 52

[0123] 46, 48, 50, 51

[0124] 46, 49, 49, 50

[0125] 채널 0에서의 블록 2가 다음과 같은 예측 값들을 포함한다:

[0126] 36, 39, 40, 42

[0127] 36, 37, 39, 41

[0128] 36, 39, 39, 41

[0129] 시간 예측이 사용되기 때문에, 채널 2의 디코딩된 값들을 수정하는, 인코딩된 값들의 양자화는 채널 0에서의 예측 값들에 영향을 주지 않고, 따라서 프로세스가 OValue들이 정의될 때 양자화를 고려하지 않는 것에 의해 단순화될 수 있다. OValue들은 그러면 다음과 같다:

[0130] 9, 9, 10, 10, 10, 11, 11, 10, 10, 10, 10, 9

[0131] 값들을 값 4로 나누는 것에 의해 값들이 양자화될 때, 그로써 발생된 QOValue들은 다음과 같다:

[0132] 2 2 2 2 2 2 2 2 2 2 2 2

[0133] 모든 값들이 범위 - 즉, 0부터 15까지 - 내에 있기 때문에, 임의의 QOValue들에 대해 랩어라운드가 필요하지 않고, ODelta 값들이 그러면 QOValue들과 동일하다. 이제, 모든 ODelta 값들이 동일하고, 그러면 사용되는 코딩 방법을 BlockChannel00DeltaCoded 방법으로부터 BlockChannel00DeltaSame 방법으로 또는 BlockChannel0IDeltaSame 방법으로 변경하는 것이 실현가능하다는 것을 또한 잘 알 것이다. 이 BlockChannel00DeltaSame 방법은 그와 함께 단지 하나의 값(2)이 전달되는 것을 필요로 하고, 이는 블록 값들을 디코더에서 적절히 - 즉, 인코더에서 역시 행해진 것과 유사한 방식으로 - 디코딩하는 것을 가능하게 한다.

[0134] 그 데이터 블록에 대해, 디코더는 이어서 인코딩 방법 BlockChannel00DeltaSame 및 값 2를 이용하도록 지시받는다. 이어서, 디코더는 다음과 같은 12개(4 x 3 블록) 값들을 포함하는 버퍼를 생성한다:

[0135] 2 2 2 2 2 2 2 2 2 2 2 2

[0136] 이 예에서, 역양자화(de-quantization)는 값들을 4와 곱하고, 양자화된 범위 내에서 오차의 보다 나은 추정을 가능하게 하기 위해 임의로 1을 가산한다. 이 값들이 역양자화될 때, 그로써 발생된 값들은 다음과 같다:

[0137] 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9

[0138] 이 역양자화된 값들이, 채널 0으로부터의 인코더에서의 것들과 유사한, 예측 값들에 가산될 때, 블록 값들은 다음과 같다:

[0139] 45, 48, 49, 51

[0140] 45, 46, 48, 50

[0141] 45, 48, 48, 50

[0142] 왜곡 - 즉, 인코딩 및 디코딩 동안 값들의 주어진 원래 데이터와 인코딩-디코딩 프로세스 이후의 대응하는 디코딩된 데이터 사이의 오차들 - 은 다음과 같을 것이다:

[0143] 0, 0, 1, 1

[0144] 1, 2, 2, 1

[0145] 1, 1, 1, 0

[0146] 이 왜곡은 양자화에 의해 야기되고 꽤 작다. 본 방법은 아주 효율적인데, 그 이유는 주어진 4x3 데이터 블록 전체에 대해, 선택된 방법 및 하나의 값만 전달되면 되기 때문이다. 채널에서 보다 많은 블록들이 동일하거나 실질적으로 유사한 인코딩 방법으로 코딩되어야 하는 경우, 그 값들 모두는, 예를 들어, 훨씬 더 높은 압축비 - 즉, 원래 또는 디코딩된 데이터(D1)의 양/인코딩된 데이터(E2)의 양의 비 - 를 가능하게 하기 위해 주어진 인코더에서 범위 인코딩되고 대응하는 주어진 디코더에서 범위 디코딩될 수 있다.

[0147] 앞서 언급된 ODelta를 사용할 때, 임의로, 채널 2가 채널 0 이전에 인코딩되고, 이어서 채널 0이 유익하게도 예측을 위해 채널 2를 사용한다. 이 경우에, QOValue들 전부는 유익하게도 -2이다. 그 값들은 lowValue보다 더 낮고, 따라서 랩어라운드(즉, wrapValue의 가산)가 필요하며, 그러면 모든 ODelta 값들은 14(-2 + 16)이다. 이와 같이, 본 개시내용의 실시예들을 구현하는 데 이용되는 앞서 언급된 ODelta 코딩은 범위를 증가시키지 않거나 임의의 부호 비트를 필요로 하지 않으며, 그 때문에, 유사한 공지된 Delta 코딩 방법보다 더 효율적이다. ODelta 코딩도 이 현재 데이터 블록에 대한 차이 채널에 대해 DC 방법을 이용하는 것보다 더 효율적이다.

[0148] 다음에, 채널 1에서 BlockSpatialODeltaCoded를 사용해 4x3 블록을 공간적으로 인코딩하는 일 예가 기술될 것이다. 이 예시적인 경우에, 이 방법은 0을 첫 번째 예측 값으로서 사용한다. 원래 값들은 0부터 255까지의 범위에 있고, 인코더에서 lowValue가 0이고, highValue가 35이며 wrapValue가 26이도록 품질이 설정된다. 물론, 대응하는 디코딩 단계에서, 값들은 유익하게도 여전히 0부터 255까지의 원래 범위에 있고, 이와 같이, 디코딩 단계에서, lowValue는 0이고, highValue는 255이며 wrapValue는 256이다. 원래 블록은 그러면 다음과 같다:

[0149] 141, 151, 148, 137

[0150] 159, 150, 152, 147

[0151] 159, 154, 153, 150

[0152] 처음 16개 레벨들 - 즉, 레벨 0 내지 레벨 15 - 및 마지막 16개 레벨들 - 즉, 레벨 20 내지 레벨 35 - 에 대한 양자화기는 7이고; 중간의 4개 레벨들 - 즉, 레벨 16 내지 레벨 19 - 에 대해, 양자화기는 8이다. 이것은 112 미만의 모든 절대 차이 값들 - 즉, 0부터 111까지 - 이 값 7에 의해 양자화된다 - 즉, 나누어진다 - 는 것을 의미한다. 144까지의 다음 값들 - 즉, 112부터 143까지 - 은 값 112에 의해 감산되고, 값 8에 의해 양자화되며, 값 16에 의해 가산된다. 마지막 값들 - 즉, 144부터 255까지 - 은 값 144에 의해 감산되고, 값 7에 의해 양자화되며, 값 20에 의해 가산된다. 이와 유사하게, 처음 16개 값들 - 즉, 값 0 내지 값 15 - 에 대한 역양자화는 각각의 값을 값 7과 곱하는 것에 의해 수행된다. 16부터 19까지의 중간 값들은 각각의 값으로부터 값 16을 감산하고 그 후에 이어서 각각의 값을 8과 곱하며 이어서 각각의 값에 값 112를 가산하는 것에 의해 디코딩된다. 마지막 값들 - 즉, 값 20 내지 값 25 - 은 각각의 값으로부터 값 20을 감산하고 이어서 각각의 값을 값 7과 곱하며 이어서 각각의 값에 값 144를 가산하는 것에 의해 디코딩된다.

[0153] 첫 번째 값은 141이다. 예측 값은 0으로서 초기화되고, 따라서 OValue는 141이고 QOValue는 그러면 19이다. 랩어라운드가 필요하지 않고, ODelta 값이 그러면 또한 19이다. 이 값은 랩어라운드를 사용하지 않고 값 136(16 * 7 + 3 * 8)으로 다시 디코딩되고, 그 값은 임의로 유익하게도 장래의 예측들을 위해 사용된다.

[0154] 두 번째 값은 151이고, A에 기초한 예측 값이 임의로 사용되고 그 값은 136이다. OValue는 15이고, QOValue는 2이다. 다시 말하지만, 랩어라운드가 필요하지 않고, ODelta 값은 2이다. 이 값은 랩어라운드를 이용하지 않고 값 150(136 + 2 * 7)으로 다시 디코딩되고, 그 값은 임의로 유익하게도 또한 장래의 예측들을 위해 사용된다.

[0155] 세 번째 값은 148이고, A에 기초한 예측 값은 이제 150이다. OValue는 -2이고, QOValue는 0이다. 다시 말하지만, OValue가 마이너스이더라도, 랩어라운드가 필요하지 않고, 양자화된 값 QOValue은 0이고, 범위 내에 있으며, 따라서 ODelta 값이 또한 0이다. 이 값은 랩어라운드가 이용되지 않고 값 150(150 + 0)으로 다시 디코딩되고, 그 값은 또한 임의로 유익하게도 장래의 예측들을 위해 사용된다.

[0156] 네 번째 값은 137이고, A에 기초한 예측 값은 또다시 150이다. OValue는 -13이다. QOValue는 -1이다. 이제는 랩어라운드가 필요하고, ODelta 값은 -1 + 36 = 35이다. 이 값이 다시 디코딩될 때, 값 150 + 249는 highValue - 즉, 값 255 - 보다 더 높고, 따라서 랩어라운드가 필요하게 되고, 결과는 그러면 150 + 249 - 256 = 143이고, 이는 임의로 유익하게도 장래의 예측들을 위해 사용된다.

[0157] 다섯 번째 값은 새로운 행에 있고, 159이다. 예측이 이제 136인 값 B를 사용하여 실행될 수 있다. OValue는 23이고, QOValue는 3이다. 랩어라운드가 필요하지 않고, 대응하는 ODelta 값은 따라서 또한 3이다. 디코딩된

값은 이제 136 + 21 = 157이고, 이 값이 또한 임의로 유익하게도 장래의 예측들을 위해 사용된다.

[0158] 여섯 번째 값은 150이고, A+ B - C를 예측 값으로서 사용할 수 있다. 예측 값은 이제 157 + 150 - 136 = 171이다. OValue는 -21이고, QOValue는 -3이다. 랩어라운드가 필요하고, ODelta 값은 따라서 33이다. 디코딩된 값이 또한 랩어라운드될 필요가 있고, 그러면 결과는 150(171 + 235 - 256)이며, 이는 또한 임의로 유익하게도 장래의 예측들을 위해 사용된다.

[0159] 처리가 유익하게도 유사한 방식으로 주어진 데이터 블록의 끝까지 계속되고, ODelta 코딩된 결과 전체는 그러면 다음과 같다:

[0160] 19, 2, 0, 35, 3, 33, 0, 0, 0, 0, 0, 1

[0161] 이 값들은 유익하게도 버퍼에 삽입되고, 예를 들어, 다른 유사하게 인코딩된 ODelta 값들과 함께, SRLE 범위 코딩을 사용하여 엔트로피 인코딩된다. 디코더는 방법 BlockSpatialODeltaCoded에 관한 정보 및 인코딩된 ODelta 값들을 인코딩된 데이터(E2)에서 제공받고, 디코더는 이어서 다음과 같이 값들을 재생성할 수 있다:

[0162] 136, 150, 150, 143

[0163] 157, 150, 150, 143

[0164] 157, 150, 150, 150

[0165] 왜곡 - 즉, 인코딩 및 디코딩 이후의 값들의 원래 데이터(D1)와 디코딩된 데이터(D3) 사이의 오차(여기서 인코딩과 디코딩은 서로 역임) - 은 다음과 같다:

[0166] 5, 1, -2, -6

[0167] 2, 0, 2, 4

[0168] 2, 4, 3, 0

[0169] 첫 번째 값이 어쩌면 임의의 값일 수 있고, 결과를 개선시키기 위해 양호한 초기화 추정치가 유익하게도 용이하게 이용된다는 것을 잘 알 것이다. 이 첫 번째 값들을 개별적으로 전달하고 다른 값들의 인코딩 결과를 개선시키는 것이 또한 가능하다. 다른 값들은 전형적으로 많은 0 그리고 또한, +1에 대응하는, 1에 가까운 그리고, -1에 대응하는, 35에 가까운 몇 개의 값들을 포함하며; 이 예에서, highValue는 35이었고 wrapValue는 36이었다.

[0170] 때때로, 모든 코딩된 값들이 0이고, 이러한 경우에, 예를 들어, 공간 예측만을 수행하고 ODelta 코딩을 위한 어떤 값도 전달하지 않는 BlockSpatialODeltaNotCoded 방법이 유익하게도 사용된다. 전형적으로, 공간 예측은 코딩된 값들을 필요로 하지만, 시간 예측은 종종 또한, 모든 코딩된 값들 - 즉, "Coded" 방법 - 대신에, 일정한 값을 사용해 - 즉, "Same" - 또는 값들을 코딩함이 없이 - 즉, "NotCoded" - 동작된다.

[0171] 본 개시내용에 따른 방법들이 사용될 때, 그들은 전형적으로 서로 상이한 방법들로서 사용된다. 어떤 기본 방법(base method)을 사용하고 이어서, 예를 들어, 주어진 사용된 시간 예측 소스, 코딩 해결책 등을 기술하는 하위방법을 이용하는 것이 또한 가능하다. 다음 예에서, 표 2는 서로 상이한 채널 및 블록 코딩 하위방법들의 리스트를 제공한다. 각각의 하위방법은 연관된 3 비트로 표현 - 즉, 정의 - 된다.

[0172] [표 2]: 채널 및 블록 코딩 방법들

표 2

[0173]

방법	기본 방법	하위 방법
Channel_Channel0ODeltaCoded	Channel0Delta	000
Channel_Channel1ODeltaCoded	Channel0Delta	001
Channel_ViewPreviousChannel0ODeltaCoded	Channel0Delta	010
Channel_ViewPreviousChannel1ODeltaCoded	Channel0Delta	011
Channel_ViewPreviousChannel2ODeltaCoded	Channel0Delta	100
Channel_FramePreviousODeltaCoded	Channel0Delta	101
Channel_ChannelPreviousODeltaNotCoded	Channel0Delta	110
Channel_SpatialODeltaCoded	Channel0Delta	111
Block_Channel0ODeltaCoded	BlockODelta	000
Block_Channel1ODeltaCoded	BlockODelta	001

Block_Channel0ODeltaSame	Block0Delta	010
Block_Channel0ODeltaNotCoded	Block0Delta	011
Block_FramePrevious0DeltaCoded	Block0Delta	100
Block_BlockLeft0DeltaCoded	Block0Delta	101
Block_BlockUp0DeltaCoded	Block0Delta	110
Block_Spatial0DeltaCoded	Block0Delta	111

[0174] 상이한 채널들이 임의로 서로 상이한 코딩 방법들을 사용하여 코딩된다. 예를 들어, 주어진 제1 채널(채널 0)은, 데이터 블록들을 인코딩하기 위해 Block0Delta 방법들을 임의로 또한 사용하는, 블록 인코더로 코딩되고, 주어진 제2 채널(채널 1)은 Channel_Channel0ODeltaCoded 시간 예측 기반 0Delta 방법으로 코딩되며, 주어진 제3 채널(채널 2)은 Channel_Spatial0DeltaCoded 공간 예측 기반 0Delta 방법으로 코딩된다. 모든 채널들이 동일한 방법으로 - 예를 들어, 공간 예측 기반 0Delta 방법으로 - 코딩되는 경우, 예를 들어, 대응하는 프레임에 대해 Frame_Spatial0DeltaCoded 방법이 유익하게도 전달되고, 그 프레임에 대해 어떤 다른 코딩 방법도 전달될 필요가 없다 - 즉, 어떤 채널 코딩 방법들 또는 블록 코딩 방법들도 사용될 필요가 없다 -.

[0175] 본 개시내용의 실시예에 관련되어 있는 데이터 구조들은 도 1 내지 도 4를 참조하여 다음에 기술될 것이다. 본 개시내용의 방법들에서 공간 예측을 위해 사용될 수 있는 데이터 구조들 및 데이터 값들이 나타내어져 있다. 도 1에는, 상위 레벨 구조 - 그의 데이터는, 예를 들어, 3개의 프레임들로 이루어져 있고, 여기서 각각의 프레임은, 예를 들어, 2개의 뷰들을 가지며, 각각의 뷰는, 예를 들어, 3개의 채널들을 가짐 - 의 일 예가 도시되어 있다. 임의로, 상위 레벨 구조의 데이터는 슬라이스들, 데이터 블록들의 그룹들, 또는 다른 데이터 구조들을 포함한다. 임의로, 일부 구조들 - 예를 들어, 패킷들 및/또는 뷰들 - 은 없다. 상이한 구조들이 존재하는 것으로 인해 또는 이용되는 값들 또는 구조들의 처리 순서의 선택으로 인해, 데이터 값들의 예측이 어쩌면 상당히 변한다. 도 2에는, 6개의 블록들의 예시적인 채널이 도시되어 있다. 인코딩을 위한 블록들의 순서는 좌에서 우로 그리고 위에서 아래로이다. 도 2에서, 블록은, 예를 들어, 3개의 패킷들을 포함하고, 여기서 각각의 패킷은, 예를 들어, 4개의 값들을 갖는다. 더욱이, 도 3에는, 예시적인 블록 및 그의 구성 요소들이 도시되어 있다.

[0176] 따라서, 예시적인 데이터 전체는 그러면 다음과 같다: -> 데이터 = 3 * 2 * 3 * 6 * 3 * 4개의 값들 = 1296개의 값들. 다음 예는 예측을 위해 사용되는 이웃 데이터 값들을 제공한다. 도 4에서, 값 A 내지 값 N은 위치 X에 대한 이전 데이터 값들이다. 값 'o' 내지 값 't'는, 예를 들어, 그들이 주어진 현재 블록보다 이전에 처리되는 하나 이상의 다른 블록들에 있는 경우, 이전 데이터 값들이다. 이전 데이터 패킷들, 데이터 블록들, 채널들, 뷰들 및 프레임들이 또한 어쩌면 예측을 위해 사용가능한 데이터 값들을 포함한다. 이와 같이, 도 4는, 본 개시내용에 따른, 어쩌면 예측을 수행하는 데 사용가능한 이웃 데이터 값들의 개략도이다.

[0177] 앞서 기술된 본 개시내용의 실시예들은, 앞서 언급된 바와 같이, 0Delta 연산자의 수정된 버전을 이용하는 것에 의해 데이터 압축 결과들을 개선시킬 수 있는 방법들을 제공한다. 본 개시내용의 방법들은 어쩌면 많은 서로 상이한 예측 방법 대안들 및 0Delta 래핑과 적절히 동작하는 임의의 양자화기를 이용한다. 본 개시내용의 방법들은 또한 0Delta 연산된 엔트로피 감소(0Delta-operated entropy reduction)에 의해 주어지는 이점들 전부를 이용하기 위해 엔트로피 인코딩을 이용한다. 본 방법들은 다양한 서로 상이한 종류들의 데이터 구조들 - 예를 들어, 프레임, 채널, 데이터 블록 및 데이터 패킷 - 에 대해 적당하다. 각각의 구조에 대해 잘 정의된 코딩 방법들이 이용되고, 이는 소량의 대응하는 0Delta 코딩된 데이터를 생성한다. 대응하는 인코딩되지 않은 데이터에 대해 필요하게 되는 것에 비해 인코딩된 데이터를 저장 및 전달하는 데 필요하게 되는 데이터 통신 대역폭의 절감이 어쩌면 상당하고 아주 유익하다.

[0178] 앞서 언급된 방법들 및 실시예들은 유익하게도 데이터 인코더들 및 데이터 디코더들에서 구현된다. 도 5를 참조하면, 본 개시내용의 실시예들은 다음과 같은 것들에 관한 것이다:

[0179] (i) 대응하는 인코딩된 데이터(E2)를 발생시키기 위해 입력 데이터(D1)를 인코딩하는 인코더(100), 및 인코딩된 데이터(E2)를 발생시키기 위해 입력 데이터(D1)를 인코딩하는, 앞서 언급된 바와 같은, 대응하는 방법들;

[0180] (ii) 대응하는 디코딩된 데이터(D3)를 발생시키기 위해 인코딩된 데이터(E2)를 디코딩하는 디코더(120). 임의로, 디코딩된 데이터(D3)는, 무손실 인코딩에서와 같이, 입력 데이터(D1)와 정확하게 유사하거나, 디코딩된 데이터(D3)는, 손실 디코딩에서와 같이, 입력 데이터(D1)와 거의 유사하거나, 데이터(D3)는, 예를 들어, 변환에 의해, 입력 데이터(D1)와는 상이하지만, 입력 데이터(D1)에 존재하는 정보, 및 디코딩된 데이터(D3)를 발생시키

기 위해 인코딩된 데이터(E2)를 디코딩하는 대응하는 방법들을 실질적으로 유지한다;

[0181] (iii) 적어도 하나의 인코더(100)와 적어도 하나의 디코더(120)의 조합을 포함하는 코덱(130) - 코덱(130)은 임의로 단일의 디바이스 내에 구현되거나 사실상 다수의 디바이스들 간에 구현됨 -; 예를 들어, 코덱(130)은 임의로 제1 공간 위치에 있는 인코더(100) 및 복수의 다른 공간 위치들에 있는 복수의 디코더들(130)이 있는 브로드캐스트 시스템으로서 구현된다.

[0182] 첨부된 청구항들에 의해 한정되는 바와 같은 본 발명의 범주를 벗어나지 않고 이상에서 기술된 본 발명의 실시예들에 대한 수정들이 가능하다. "포함하는(including)", "포함하는(comprising)", "포함하는(incorporating)", "~로 이루어져 있는", "갖는", "~인" 등의 표현들은 본 발명이 비배타적 방식으로 해석되어야만 한다 - 즉, 명시적으로 기술되지 않은 항목들, 컴포넌트들 또는 요소들이 또한 존재할 수 있다 - 는 것을 기술하고 청구하기 위해 사용된다. 단수를 언급하는 것이 또한 복수에도 관련되어 있는 것으로 해석되어야 한다. 첨부된 청구항들에서 괄호 안에 포함된 숫자는 청구항의 이해를 도우려고 의도되어 있고, 이 청구항들에 의해 청구되는 발명 요지를 결코 제한하는 것으로 해석되어서는 안된다.

[0183] **부록 1: ODelta 인코딩의 개요**

[0184] ODelta 인코딩의 개요가 이하에서 제공된다. 인코더(1010), 인코더(1010)를 사용하는 방법, 디코더(1020), 및 디코더(1020)를 사용하는 방법을 포함하는, ODelta 인코딩 및 연관된 기술이 제공된다.

[0185] 대응하는 인코딩된 출력 데이터(DA2 또는 DA3)를 발생시키기 위해 수치 값들의 시퀀스를 포함하는 입력 데이터(DA1)를 인코딩하는 인코더(1010)는 인코더(1010)가 하나 이상의 대응하는 인코딩된 시퀀스들을 발생시키기 위해 차분 및/또는 합 인코딩의 형태를 입력 데이터(DA1)에 적용하기 위한 데이터 처리 장치를 포함하고, 여기서 인코딩된 출력 데이터(DA2 또는 DA3)를 발생시키기 위해, 하나 이상의 대응하는 인코딩된 시퀀스들에 최대 값에서의 랩어라운드 및/또는 최소 값에서의 랩어라운드가 적용된다는 것을 특징으로 한다.

[0186] 대응하는 인코딩된 출력 데이터(DA2 또는 DA3)를 발생시키기 위해 수치 값들의 시퀀스를 포함하는 입력 데이터(DA1)를 인코딩하기 위해 인코더(1010)를 사용하는 방법은 본 방법이

[0187] (a) 하나 이상의 대응하는 인코딩된 시퀀스들을 발생시키기 위해 차분 및/또는 합 인코딩의 형태를 입력 데이터(DA1)에 적용하기 위해 인코더(1010)의 데이터 처리 장치를 사용하는 단계; 및

[0188] (b) 인코딩된 출력 데이터(DA2 또는 DA3)를 발생시키기 위해 하나 이상의 대응하는 인코딩된 시퀀스들에 최대 값에서의 랩어라운드 및/또는 최소 값에서의 랩어라운드를 적용하기 위해 데이터 처리 장치를 사용하는 단계를 포함하는 것을 특징으로 한다.

[0189] 대응하는 디코딩된 출력 데이터(DA5)를 발생시키기 위해 인코딩된 데이터(DA2, DA3 또는 DA4)를 디코딩하는 디코더(1020)는 디코더(1020)가 인코딩된 데이터(DA2, DA3 또는 DA4)의 하나 이상의 부분들을 오프셋 처리하는 데이터 처리 장치를 포함하고, 여기서 데이터 처리 장치는 차분 및/또는 합 디코딩의 형태를 하나 이상의 부분들의 하나 이상의 대응하는 인코딩된 시퀀스들에 적용하기 위해 동작가능하며, 여기서 디코딩된 출력 데이터(DA5)를 발생시키기 위해, 하나 이상의 인코딩된 시퀀스들에 최대 값에서의 랩어라운드 및/또는 최소 값에서의 랩어라운드가 적용된다는 것을 특징으로 한다.

[0190] 대응하는 디코딩된 출력 데이터(DA5)를 발생시키기 위해 인코딩된 데이터(DA2, DA3 또는 DA4)를 디코딩하기 위해 디코더(1020)를 사용하는 방법은 본 방법이

[0191] 인코딩된 데이터(D2, D3 또는 D4)의 하나 이상의 부분들을 처리하기 위해 데이터 처리 장치를 사용하는 단계를 포함하고, 여기서 데이터 처리 장치는 차분 및/또는 합 디코딩의 형태를 하나 이상의 부분들의 하나 이상의 대응하는 인코딩된 시퀀스들에 적용하기 위해 동작가능하며, 여기서 디코딩된 출력 데이터(D5)를 발생시키기 위해, 하나 이상의 인코딩된 시퀀스들에 최대 값에서의 랩어라운드 및/또는 최소 값에서의 랩어라운드가 적용된다는 것을 특징으로 한다.

[0192] 대응하는 디코딩된 출력 데이터(DA5)를 발생시키기 위해 인코딩된 데이터(DA2, DA3 또는 DA4)를 디코딩하기 위해 디코더(1020)를 사용하는 방법은 본 방법이

[0193] (a) 인코딩된 데이터(DA2, DA3 또는 DA4)가 변환된 데이터의 순차적 값들의 변화들을 나타내는 적어도 하나의 인코딩된 시퀀스를 포함하고 최대 값에서의 랩어라운드 또는 최소 값에서의 랩어라운드를 이용한다는 것을 고려하여, 인코딩된 데이터(DA2, DA3 또는 DA4)의 하나 이상의 부분들에 디코딩을 적용하기 위해 인코딩된 데이터

(DA2, DA3 또는 DA4)를 처리하기 위해 데이터 처리 장치를 사용하는 단계; 및

[0194] (b) 디코딩된 출력 데이터(DA5)를 발생시키기 위해, 대응하는 처리된 데이터를 발생시키고 적어도 하나의 사전 오프셋 값 및/또는 사후 오프셋 값을 사용하여 하나 이상의 부분들을 변환하기 위해 데이터 처리 장치를 사용하는 단계를 포함하는 것을 특징으로 한다.

[0195] 이제부터, 단지 예로서, 이하의 도면들을 참조하여 부록 1에 대한 본 개시내용의 실시예들이 기술될 것이다:

[0196] 도 6은 본 개시내용에 따라 기능하도록 구현되는 인코더와 디코더를 포함하는 코덱을 나타낸 것이고;

[0197] 도 7은 도 6의 인코더에서 실행되는 바와 같은 데이터를 인코딩하는 방법의 단계들을 나타낸 것이며;

[0198] 도 8은 도 6의 디코더에서 실행되는 바와 같은 데이터를 디코딩하는 방법의 단계들을 나타낸 것이다.

[0199] 본 개시내용의 실시예들을 기술할 때, 표 3에 제공되는 바와 같은 이하의 약어들 및 정의들이 사용될 것이다:

[0200] [표 3]: 약어들 및 정의들

표 3

[0201]

약어	정의
ADC	아날로그-디지털 변환기(Analog-to-digital converter)
코덱	디지털 데이터에 대한 인코더 및 대응하는 디코더
DAC	디지털-아날로그 변환기(Digital-to-analog converter)
DB	RAM(Random Access Memory) 또는 ROM(Read Only Memory) 내의 데이터베이스
DC	주어진 영상의 DC 성분, 즉 평균 밝기에 대응하고 영상의 가장 낮은 공간 주파수 성분을 나타내는 영상의 평균
RLE	런 길이 인코딩(Run-length encoding)
ROI	관심 영역(Region of interest)
ROM	판독 전용 메모리(Read Only Memory)
VLC	런 길이 코드(Variable-length code)

[0202] 개요를 말하면, 도 6을 참조하면, 본 개시내용은 인코더(1010) 및 그의 연관된 동작 방법에 관한 것이고; 유익하게도 인코더(1010)는 직접 ODelta 인코더(direct ODelta encoder)로서 구현된다. 더욱이, 본 개시내용은 또한 대응하는 디코더(1020)에 관한 것이고; 유익하게도 디코더(1020)는 역 ODelta 디코더로서 구현된다. 본 개시내용의 실시예들은 유익하게도 앞서 언급된 공지된 Delta 인코딩 방법의 비트 최적화된 버전은 물론 다른 데이터에 대한 범위 최적화된 버전인 직접 ODelta 연산자(direct ODelta operator)를 이용한다. ODelta 인코딩은 가변 길이 데이터 워드들 - 예를 들어, 8/16/32/64 비트 - 을 이용하고 그리고/또는 8/16/32/64 비트 데이터 요소들 - 그의 원래 값은 1부터 64 비트의 범위로 표현되고 대응하는 인코딩된 값은 1 내지 64 비트로 발생됨 - 의 가변 길이 인코딩을 이용하는 컴퓨팅 하드웨어 또는 전용 디지털 하드웨어에서 사용된다. 물론, 인코더(1010) 및 디코더(1020)는, 어쨌든, 어느 부류의 숫자 값들이 데이터(DA1) - 예를 들어, 원래 데이터 - 에 포함되는지를 인식하고, 따라서 그의 정의 또는 전송이 여기서 더 이상 설명되지 않을 것이다. 숫자 범위(MIN 및 MAX)가 알려져 있고 데이터(DA1)가 이용될 수 있는 것으로 가정될 뿐이다.

[0203] 공지된 Delta 코딩 방법들은 값들의 범위를 원래(MIN부터 MAX까지)로부터 결과(MIN-MAX부터 MAX-MIN까지)로 증대시킨다. 이것은 원래 데이터가 플러스 값들만을 포함할 때 그 방법이 또한 마이너스 값들을 생성할 것임을 의미한다. 본 개시내용에 따른 ODelta 연산자는 대응하는 원래 값들의 범위에 있지 않은 값을 결코 생성하지 않고, 따라서 사용된 데이터 범위를 증대시키지 않으며, 따라서 유익하게도, 예를 들어, 엔트로피 감소 및 연관된 데이터 압축을 실행할 때 이용된다. 예를 들어, 공지된 Delta 인코딩 방법들은 5-비트 데이터 - 즉, 0부터 31까지의 값들의 범위에 있음 - 의 스트림들에 대해 동작하며, 따라서 이러한 Delta 인코딩 방법들에 의해 발생된 데이터 값들은 -31부터 +31 - 즉, 6 비트(즉, 부호 비트 + 5 비트)를 사용하여 실질적으로 표현가능한 63개의 값들 - 까지의 범위에 있을 것이고; 이와 달리, 직접 ODelta 발생된 값들은, 상기한 5-비트 데이터의 스트림들로부터 발생될 때, 여전히 0부터 31까지의 범위에 있다. 더욱이, 공지된 Delta 인코딩 방법들이 재귀적으로 구현될 수 없는 반면, 본 개시내용에 따른 직접 또는 역 ODelta 연산자는 재귀적으로 구현될 수 있고 또한 여전히 사용된 값들의 범위를 유지한다. 값들의 범위가 비트 단위로 정확(bit-exact)할 필요가 없고 - 예를 들어,

0부터 31까지의 값들이 5 비트에 의해 정의됨 -; ODelta 연산자는 여전히 적절히 동작하면서 임의의 값들의 범위 - 예를 들어, 0부터 25까지의 값들의 범위 - 를 사용할 수 있다.

[0204] 원칙적으로, 본원에 기술된 ODelta 방법들은 항상 기존의 데이터 범위 - 그의 일 예가 이하에서 주어짐 - 에 직접 기초하여 기능할 수 있다. ODelta 방법들은 또한 데이터에 나오는 가장 낮은 숫자 값("lowValue") 및 데이터에 나오는 가장 높은 숫자 값("highValue")을 나타내는 정보를 전달하는 것에 의해 향상될 수 있다. 유의할 점은, lowValue >= MIN이고 highValue <= MAX이며, 이 값들이 임의적이라는 것이다.

[0205] 본 개시내용에 따른 직접 및 역 ODelta 연산자들의 2개의 예들이 이하에서 기술될 것이다. 직접 및 역 ODelta 연산자들의 제1 예는 효율적이고, 예를 들어, 비임시적(비일시적) 머신 판독가능 데이터 저장 매체 상에 기록된 하나 이상의 소프트웨어 제품들을 실행하기 위해 동작가능한 전자 하드웨어 및/또는 컴퓨팅 하드웨어에 구현하기가 비교적 간단하다.

[0206] 본 개시내용에 따른 직접 또는 역 ODelta 연산자들을 구현할 때, 유익하게도 원래 데이터 값들의 시퀀스 모두는 플러스이고 가장 낮은 값은 0이다. 임의로, 모두가 플러스 값이고 가장 낮은 값이 "0"이도록 데이터 값들을 시프트시키기 위해 어떤 오프셋 값 - 즉, 사전 오프셋 값 또는 사후 오프셋 값 - 이 이용될 수 있다. 본 개시내용에 따른 ODelta 연산자는 모든 유형들의 데이터에 대해 직접적인 방식으로 이용될 수 있고; 오프셋 값이 모든 값들에 가산되거나 모든 값들로부터 감산될 때, 데이터 값들의 범위가 보다 적은 비트들을 사용하여 정의될 수 있기 때문에, 전형적으로 데이터 압축 - 즉, 전달되는 데이터 레이트를 감소시키는 것 - 을 제공할 수 있다. 예를 들어, 직접 또는 역 ODelta 연산자의 적용 이전에, 원래 데이터 값들은 -11부터 +18까지의 범위에 있고; 이러한 범위는 +11의 오프셋 값을 사용하는 것에 의해 0부터 29까지의 범위로 변환되고 변환된 범위는 그 후에 5 비트에 의해 기술될 수 있다. 이러한 사전 오프셋 값 또는 사후 오프셋 값이 이용되지 않을 때, 원래 데이터 값들은 그들을 기술하는 데 적어도 6 비트를 필요로 하고, 종종, 실제로는, 편의상 완전 8-비트 부호있는 바이트(full 8-bit signed byte)가 이용된다.

[0207] 일반화된 직접 또는 역 ODelta 연산자를 사용할 때 데이터 범위의 유사한 최적화가 또한 가능하다. 이와 같이, 직접 또는 역 ODelta 연산자, 또는 어떤 다른 방법이 값들의 전체 범위보다 더 작은 오프셋 값을 제공받을 수 있는 데이터 값들을 생성하는 경우, 그 범위 최적화는 ODelta 인코딩 방법에서의 임의의 단계에서 구현될 수 있다. 오프셋 값 - 부호가 마이너스이든 플러스이든 관계없음 - 이 사용될 때, 도 6, 도 7 및 도 8을 참조하여 나중에 설명되는 바와 같이, 오프셋 값이 또한 인코더(1010)로부터 디코더(1020)로 전달되어야 한다.

[0208] 직접 ODelta 연산자는, 예를 들어, 원래 데이터(DA1)를 비트 단위 방식으로 인코딩하기 위해, 1-비트 방식으로 구현될 수 있고; 이러한 1-비트 방식에서, 방법 1 및 방법 3은, 이하에서 보다 상세히 기술될 것인 바와 같이, 도 6에서 원래 데이터(DA1)에서 비트 값들의 변화가 없을 때는 값 "0"을 그리고 원래 데이터(DA1)에서 비트 값들의 변화가 있을 때는 값 "1"을 생성한다. 원래 데이터에서의 첫 번째 비트에 대한 예측은 임의로 값 "0"이고, 따라서 원래 데이터(DA1)에서의 첫 번째 비트의 값은 보존된다. 원래 데이터에서의 첫 번째 비트의 예측된 값을 값 "1"로서 이용하는 것이 또한 대안적으로 임의로 가능하지만, 이러한 선택이 어떤 코딩 이점들도 제공하지 않으며; 이 때문에, 예측이 1-비트 데이터에 대해 기본적으로 값 "0"인 것으로 임의로 항상 가정될 때 선택이 전달될 필요가 없고 - 즉, 미리 정의된 값 "0"이 인코더(1010) 및 디코더(1020)에 의해 이용되고 -, 그로써 이 예측이 전달될 필요가 없게 되고 따라서 개선된 데이터 압축이 얻어진다.

[0209] 본 개시내용에 따른 직접 ODelta 인코딩의 일 예가 이제부터 기술될 것이다. 17개의 "1"과 20개의 "0"을 포함하는 예시적인 원래 비트 시퀀스 - 즉, 27 비트 - 가 다음과 같이 수학식 1에 제공되고:

수학식 1

[0210] **010101100100010100000000000111111111**

[0211] 그의 엔트로피(E)는 수학식 2로부터 계산가능하다:

수학식 2

$$E = 17 * \log_{10}\left(\frac{37}{17}\right) + 20 * \left(\frac{37}{20}\right) = 11.08523$$

[0212]

[0213] 수학식 2에서의 엔트로피(E)를 코딩하는 데 필요한 비트들의 수 - 즉, Min 비트 - 는, 수학식 3에 제공된 바와 같이, 표 4의 문서들 P7 및 P8에 기술된 바와 같은, Shannon의 소스 코딩 정리(source coding theorem)로부터 계산가능하다.

수학식 3

$$Min\_bits = \frac{E}{\log_{10}(2)} = 36.82$$

[0214]

[0215] 앞서 언급된 바와 같이 원래 비트 시퀀스에 직접 ODelta 연산자가 적용될 때 - 즉 방법 1 및 방법 3 -, 다음과 같이 37 비트 - 13개의 "1"과 24개의 "0"이 있음 - 를 포함하는 비트 시퀀스가 발생되고:

수학식 4

[0216] **0111110101100111100000000001000000000**

[0217] 그의 엔트로피(E)는 수학식 5로부터 계산가능하며:

수학식 5

$$E = 13 * \log_{10}\left(\frac{37}{13}\right) + 20 * \left(\frac{37}{24}\right) = 10.41713$$

[0218]

[0219] 수학식 5는 수학식 6에 따라 최소 비트 수 - 즉, Min 비트 - 로 표현가능하다:

수학식 6

$$Min\_bits = \frac{E}{\log_{10}(2)} = 34.60$$

[0220]

[0221] 수학식 4에서의 비트 시퀀스는 유익하게도 데이터 압축을 달성하기 위해, 예를 들어, RLE(run-length encoding), Huffman 코딩, 산술 인코딩, 범위 인코딩, EM(Entropy Modifier) 인코딩 또는 SRLE 인코딩 중 적어도 하나를 사용하여, 추가로 인코딩될 수 있다.

[0222] ODelta 연산자는, 예를 들어, 수학식 1에서와 같이, 원래 데이터 대신에, 예를 들어, 수학식 4에서와 같이, 그의 연관된 엔트로피 코딩 방법이 적용될 때 - 예를 들어, 연산된 데이터에 대해 RLE 또는 SRLE가 사용될 때 -, 원래 데이터(DA1)를 표현하는 데 필요한 비트들의 양을 감소시키고; 이 1-비트 직접 ODelta 연산자 - 즉, 방법 1 및 방법 3 - 는 수학식 1에서의 원래 비트 시퀀스에 많은 변화가 있을 때는 "1"을 생성하고, 수학식 1에서의 원래 비트 시퀀스에 서로 유사한 비트들의 긴 스트림이 있을 때는 "0"을 발생시킨다.

[0223] ODelta 연산자의 역 버전 - 즉, 방법 1 및 방법 3의 역 - 은, 인코딩된 데이터 스트림에 값 "1"이 있을 때는, 적절한 경우 비트 값을 값 "0"으로부터 값 "1"로 또는 값 "1"로부터 값 "0"으로 변경하고, 인코딩된 데이터(DA2) 스트림에 "0" 값이 있을 때는 비트 값을 변경하지 않는다. 직접 ODelta 연산된 데이터(DA2) 비트 스트림에 대해 이 ODelta 연산이 실행될 때, 원래 데이터(DA1) 스트림이 디코딩된 데이터(DA5)로서 재생성되지만; 앞서 언급된 바와 같이, 역시 고려될 필요가 있는, VLC 또는 Huffman 코딩과 같은 부가의 코딩이 유익하게도 이용되며; 이것은 데이터(DA3)가 엔트로피 인코더의 순방향 연산(forward operation)을 사용하여 데이터(DA2)로부터 발생되고 데이터(DA4)가 엔트로피 디코더의 역연산(inverse operation)을 사용하여 데이터(DA3)로부터 발생된다는 것을 의미한다.

[0224] 유익하게도, 원래 데이터(DA1) 스트림은, 그에 인코딩이 적용되기 전에, 2개 이상의 섹션들로 세분된다. 이러한 세분은 원래 데이터(DA1) 스트림을 인코딩할 때 보다 많은 최적화가 이용될 기회를 제공한다. 예를 들어, 이러한 세분은 유익한데, 그 이유는 데이터(DA1)에서의 변경가능한 시퀀스들이, 직접 ODelta 인코딩될 때 - 즉, 방법 1 및 방법 3을 이용할 때 -, 보다 많은 "1"을 발생시키는 반면, 단조로운 변하지 않은 시퀀스들 - 즉, "단조로운" 시퀀스들 - 이 보다 많은 "0" - 예를 들어, 차후의 VRL 인코딩 또는 Huffman 인코딩에 바람직함 - 을 생성하고, 따라서 데이터(DA1)를 앞서 언급된 바와 같이 개별적으로 인코딩될 수 있는 복수의 섹션들로 나누는 것에 의해 데이터(DA1)를 구성하는 비트스트림 전체에 대해 엔트로피(E)가 감소될 수 있기 때문이다.

[0225] 본 발명에 따른 직접 ODelta 인코딩의 일 예가 다음에 서로 개별적으로 인코딩되는 복수의 섹션들이 이용될 때 기술될 것이다. 원래 단일 비트들의 시퀀스를 포함하는 제1 섹션은 하기의 수학식 7에서의 총 16개의 비트들 - 즉, 7개의 "1"과 9개의 "0" - 을 포함하고:

수학식 7

[0226] **0101011001000101**

[0227] 여기서 H(X) = 4.7621이고 B = 15.82이며; "H"는 엔트로피를 나타내고 "B"는 Max 비트를 나타낸다. 원래 비트들의 수학식 7 시퀀스에 직접 ODelta 연산자가 적용될 때, 대응하는 변환된 비트들의 시퀀스가 수학식 8에서와 같이 제공되고:

수학식 8

[0228] **0111110101100111**

[0229] 여기서 H(X) = 4.3158이고 B = 14.34이다.

[0230] 원래 단일 비트들의 시퀀스를 포함하는 제2 섹션은 하기의 수학식 9를 포함하고:

수학식 9

[0231] **000000000001111111111**

[0232] 여기서 H(X) = 6.3113이고 B = 20.97이다. 원래 비트들의 수학식 9 시퀀스에 직접 ODelta 연산자가 적용될 때, 대응하는 변환된 비트들의 시퀀스가 수학식 10에서와 같이 제공되고:

수학식 10

[0233] **000000000001000000000**

[0234] 여기서 H(X) = 1.7460이고 B = 5.80이다. 이 예들에서, 앞서 언급된 바와 같이, H(X)는 엔트로피(E)를 나타내

고, B는 코딩을 위해 필요한 최소 비트 수를 나타낸다.

[0235] 이 예에서, 수학식 7 및 수학식 10으로부터의 최상의 압축은 양 섹션에 직접 ODelta 연산자가 적용될 때(즉, 총 14.34 비트 + 5.80 비트 = 20.14 비트로 인코딩할 때) 달성되고; 이것은 원래 필요했던 36.82 비트보다 더 적은 비트를 필요로 한다 - 즉, 직접 ODelta 연산된 비트는 34.60 비트, 또는 분할 이후에 필요한 원래 비트 수(= 15.82 비트 + 20.97 비트 = 36.79 비트)를 필요로 함 -. 유익하게도, 데이터(DA1)에서의 원래 비트 스트림을 섹션들로 분할하는 것은 원래 데이터(DA1)의 엔트로피(E) 및 수정된 데이터 - 즉, 데이터(DA2)에 포함되어 있음 - 의 대응하는 엔트로피(H)를 조금씩 분석하는 것에 의해 자동으로 실행된다.

[0236] 데이터(DA1)에 다수의 긴 섹션들이 있을 때, 비트 값들이 시퀀스를 따라 빠르게 변하는 충분히 큰 데이터 영역이 있기만 하다면, 데이터 압축은 단지 데이터(DA1)의 부분들을 인코딩될 새로운 섹션으로 나누는 것에 의해 대략적인 방식으로 임의로 구현된다. 임의로, 데이터(DA1)의 일부 섹션들은, 예를 들어, 긴 서로 유사한 비트들이 있고 그들 사이에 비교적 적은 개개의 상이한 비트들이 있는 경우, 직접 ODelta 연산자를 이용하지 않고 인코딩되며; 이러한 경우에, 직접 ODelta 연산자는 데이터 압축을 위해 그다지 이점을 제공하지 않는다.

[0237] 데이터(DA1)를 보다 작은 섹션들로 분할하는 것은 인코딩된 데이터(DA2)에 데이터를 제공하는 부가의 오버헤드를 발생시키는 단점이 있다. 이러한 오버헤드는, 예를 들어, 모든 새로운 섹션과 연관된 데이터 비트 또는 데이터 바이트의 양을 나타내는 정보를 포함한다. 그렇지만, 적어도 특정 양의 오버헤드 데이터 값들을 전송하는 것이 필요한 것으로 항상 밝혀졌고, 따라서 주어진 데이터가 2개의 데이터 섹션들로 분할될 때 하나의 추가 오버헤드 데이터 값이 있을 뿐이다.

[0238] 나중에 디코딩될 수 있는 인코딩된 비트스트림을 달성하기 위해, 엔트로피 인코딩이 유익하게도 직접 ODelta 연산자 - 예를 들어, VLC, Huffman 코딩, 산술 코딩, 범위 코딩, RLE, SRLE, EM 및 기타 - 이후에 구현된다. 실제 데이터 인코딩에 비해 계산된 엔트로피(E) 및 최소 비트 추정 값들에 기초하여 최적화 계산들을 실행하는 것이 보다 용이하고 계산상 보다 효율적이다. 이러한 실행 순서는 상당한 속도 최적화를 가능하게 하고, 인코딩된 데이터(DA2)에서 최적의 데이터 압축 결과를 종종 달성한다. 대안적으로, 엔트로피 최적화된 비트 스트림을 발생시키기 위해 원래 비트, 알파벳, 숫자, 바이트 및 워드 데이터 - 즉, 데이터(DA1)에 있음 - 가 어떤 다른 방법으로 먼저 코딩되고 그 후에 대응하는 인코딩된 데이터 - 즉, 데이터(DA2) - 를 제공하기 위해 엔트로피 최적화된 비트스트림을 수정하는 데 직접 ODelta 연산자가 사용되는 방식으로 엔트로피 최적화를 실행하는 것이 실현가능하다. 더욱이, 데이터(DA3)를 발생시키기 위해, 이 ODelta 연산된 데이터가 여전히 다른 인코딩 방법들로 데이터(DA2)로부터 인코딩될 수 있다.

[0239] 일반화된 직접 ODelta 연산자는 데이터(DA1)에서 사용되는 값들의 범위를 기술하는 파라미터 - 즉, 값들을 제시하는 데 필요한 비트들의 값 또는 수 - 를 이용한다. 더욱이, 플러스 오프셋 값과 마이너스 오프셋 값 - 환언하면, 플러스 "페데스탈" 값과 마이너스 "페데스탈" 값 - 의 사용을 가능하게 하는 방법에서 ODelta 연산자가 이용된다. 예를 들어, 데이터(DA1)가 7 비트 - 즉, "0"부터 "127"까지의 값들이 지원됨 - 로 제시되지만 "60"부터 "115"까지의 범위에 있는 값들만을 포함하는 경우, -60의 오프셋 값이 데이터(DA1)에 적용될 때, 그에 의해 발생되는 변환된 데이터는 단지 6 비트를 포함하는 값들로도 표현될 수 있는 "0"부터 "55"까지의 범위에 있는 값들을 갖는다 - 즉, 그로써 어느 정도의 데이터 압축을 달성하는 것이 실현가능하다 -. 일반화된 직접 ODelta 연산자는 따라서 데이터(DA1)에 전체 범위의 데이터 값들이 존재할 때 결과들을 개선시킨다 - 즉, 7 비트로 표현되고 종래에는 8-비트 바이트로 표현됨 -.

[0240] 본 개시 내용에 따르면, 직접 ODelta 값들 - 즉, 방법 1 - 은 플러스 값들(lowValue = MIN = 0이고 highValue = MAX = 127이며, wrapValue = 127 - 0 + 1 = 128임)만을 갖는 데이터에 대해 다음과 같은 예시적인 소프트웨어 코드의 발췌본에 의해 기술된 바와 같은 절차를 사용하여 계산될 수 있다:

```
wrapValue = power(2, bits) = power(2, 7) = 128
prediction Value = (lowValue + highValue + 1) div 2 = (wrapValue + 1) div 2 +
lowValue = 64
for all pixels

begin
  if(originalValue >= predictionValue) then
        ODeltaValue = originalValue – predictionValue
  else
        ODeltaValue = wrapValue + originalValue – predictionValue
  predictionValue = originalValue
end
```

[0241]

[0242]

[0243] 상기한 ODelta 연산자를 추가로 설명하기 위해 일 예가 이제부터 제공될 것이다. 원래 값 시퀀스는 하기의 수학식 11이다:

수학식 11

[0244] **65, 80, 126, 1, 62, 45, 89, 54, 66**

[0245] 대응하는 Delta 코딩 값들은 하기의 수학식 12이다:

수학식 12

[0246] **65, 15, 46, -125, 61, -17, 44, -35, 12**

[0247] 대응하는 직접 ODelta 코딩 값들은 하기의 수학식 13이고:

수학식 13

[0248] **1, 15, 46, 3, 61, 111, 44, 93, 12**

[0249] 여기서 파라미터 wrapValue 내에서의 랩어라운드가 이용된다.

[0250] 역 ODelta 연산자 - 즉, 방법 1 - 는, 예를 들어, 다음과 같은 예시적인 소프트웨어 코드에 의해 구현되는 바와 같이, 역 ODelta 값들을 발생시키기 위해 사용가능하다.

```
wrapValue = power(2, bits) = power(2, 7) = 128
predictionValue = (wrapValue + 1) div 2 + lowValue = 64
for all pixels

begin
  ODeltaValue = originalValue + predictionValue
  if (ODeltaValue >= wrapValue) then
        ODeltaValue = ODeltaValue – wrapValue
  predictionValue = ODeltaValue
end
```

[0251]

[0252] 이 소프트웨어 코드가 실행되어 수학식 13에 적용될 때, 수학식 14에 제공되는 바와 같은 값들을 발생시킨다:

수학식 14

[0253] **65, 80, 126, 1, 62, 45, 89, 54, 66**

[0254] 이 예는 2의 멱수 값인 wrapValue을 사용한다. 이것은 필수적인 것이 아니며, wrapValue은 또한 가장 높은 데이터 값보다 더 큰 임의의 값, 또는, 마이너스 값들이 또한 이용가능하거나 범위가 주어진 데이터 시퀀스에서의 사전 오프셋에 의해 수정되는 경우, 사용된 범위보다 더 큰 값일 수 있다. 이 특징을 보여주는 추가의 예가 나중에 있을 것이다.

[0255] 도 6을 참조하여 전술한 바를 요약하면, 본 개시 내용은 인코더(1010)와 디코더(1020)에 관한 것이다. 임의로, 인코더(1010)와 디코더(1020)는 결합되어 1030으로 일반적으로 표시된 코덱으로서 이용된다. 인코더(1010)는 대응하는 인코딩된 데이터(DA2 또는 DA3)를 발생시키기 위해, 예를 들어, 직접 ODelta 방법을 사용하여 인코딩되는 원래 입력 데이터(DA1)를 수신하기 위해 동작가능하다. 인코딩된 데이터(DA2 또는 DA3)는 임의로 통신 네트워크(1040)를 통해 전달되거나 데이터 저장 매체(1050) - 예를 들어, 광학 디스크 ROM(read-only-memory) 또는 기타와 같은 데이터 캐리어(data carrier) - 상에 저장된다. 디코더(1020)는 인코딩된 데이터(DA2 또는 DA3) - 예를 들어, 통신 네트워크(1040)를 통해 스트리밍되거나 데이터 저장 매체(1050) 상에 제공됨 - 를 수신하기 위해 그리고 대응하는 디코딩된 데이터(DA5) - 예를 들어, 원래 데이터(DA1)와 실질적으로 유사함 - 를 발생시키기 위해 역 방법 - 예를 들어, 역 ODelta 방법 - 을 적용하기 위해 동작가능하다. 인코더(1010)와 디코더(1020)는 유익하게도 디지털 하드웨어 - 예를 들어, 하나 이상의 소프트웨어 제품들 - 예를 들어, 이 설명에서 예시적인 실시예들로서 제공되는 바와 같은 코드들 - 을 실행하기 위해 동작가능한 컴퓨팅 하드웨어 - 를 사용하여 구현된다. 대안적으로, 인코더(1010) 및/또는 디코더(1020)는 전용 디지털 하드웨어를 사용하여 구현된다.

[0256] 인코더(1010)에서 실행되는 바와 같은 ODelta 방법은 도 7에 도시된 바와 같은 단계들을 이용한다. 임의의 제1 단계(1100)에서, 입력 데이터(DA1)가 그의 데이터 요소들의 값들의 범위를 알아내기 위해 처리된다. 임의의 제2 단계(1110)에서, 값들의 범위로부터, 대응하는 변환된 요소들의 세트를 발생시키기 위해 데이터 요소들을 플러스 영역으로 변환하기 위한 오프셋 - 즉, 사전 오프셋 - 이 계산된다. 제3 단계(1120)에서, 대응하는 ODelta 인코딩된 값들을 발생시키기 위해, 임의로 제2 단계(1110)에서 변환된, 요소들에 이어서 직접 ODelta 인코딩이 적용된다. 제4 단계(1130)에서, 데이터(DA2)로부터 데이터(DA3)를 발생시키기 위해, ODelta 인코딩된 값들 및 임의의 오프셋 값, 최소 값(lowValue), 및/또는 최대 값(highValue)이 이어서, 예를 들어, RLE(run-length encoding), 범위 코딩, 또는 Huffman 코딩을 사용하여 개별적으로 인코딩된다. 오프셋 값, 최소 값(lowValue), 및/또는 최대 값(highValue)이 항상 압축가능하지는 않으며, 따라서 적당한 양의 비트들을 사용하여 인코더(1010)로부터 디코더(1020)로 전달될 필요가 있다. 더욱이, 오프셋 값, 최소 값(lowValue) 및/또는 최대 값(highValue)은 직접 ODelta 연산자에 대한 임의의 특징들이고; 예를 들어, 오프셋 값은, 특정 상황들에서, 값 "0"을 갖고, lowValue는 값 MIN을 가지며, highValue는 값 MAX를 갖는다 - 즉, 어떤 변환도 적용되지 않고 전체 범위가 사용된다 -. 특히, 직접 ODelta 연산자가 1-비트 데이터를 위해 - 즉, 비트 단위로 인코딩하기 위해 - 구현될 때, 그는 오프셋 값을 전혀 필요로 하지 않고, 그러면 단계들(1100 및 1110)이 항상 무시된다. 오프셋 값이 또한 단계(1110)에서 사용될 때, 가장 높은 값과 가장 낮은 값을 제시하는 범위 값이 그 내에서 업데이

트되어야만 한다. 상이한 값들의 수 - 즉, wrapValue - 는 디코더(1020)도 알고 있어야만 하거나, 그렇지 않으면 인코더(1010)가 그것을 압축된 데이터 내에서 디코더(1020)에 전달해야만 한다. 임의로, 디폴트 wrapValue(= highValue - lowValue + 1)가 인코더에서 그리고 디코더에서 사용된다. 임의로, 인코더(1010)와 디코더(1020) 중 적어도 하나가, 예를 들어, 인코딩된 데이터(DA2)를 발생시키기 위해 데이터(DA1)의 최적의 압축을 제공하도록 입력 데이터(DA1)를 인코딩을 위한 섹션들로 세분하는 최적의 방식을 찾아내기 위해, 재귀적 방식으로 동작한다.

[0257] 디코더(1020)에서 실행되는 바와 같은 역 ODelta 방법은 도 8에 도시된 바와 같은 단계들을 이용한다. 제1 단계(1200)에서, 디코딩된 ODelta 데이터를 발생시키기 위해, 상기한 단계(1130)에서 이용된 것의 역 인코딩이 데이터(DA2/DA3 또는 DA4)에 적용되고, 여기서 디코딩된 ODelta 데이터는 ODelta 인코딩된 값들을 갖고 임의로 별도의 오프셋 값을 갖는다. 제2 단계(1210)에서, 데이터 요소들의 시퀀스를 발생시키기 위해 ODelta 인코딩된 값들이 디코딩된다. 제3 단계(1220)에서, 디코딩된 데이터(DA5)를 발생시키기 위해 임의의 사전 오프셋 값을 사용하여 데이터 요소들의 시퀀스가 변환되고; 특정의 상황에서, 이러한 변환은 값 "0"으로 설정된다 - 즉, 어떤 변환도 사실상 적용되지 않는다 -. 다시 말하지만, 예를 들어, 1-비트 인코딩 - 즉, 비트 단위 인코딩 - 을 수행할 때, 오프셋 값을 이용할 필요 없이 방법을 실행하는 것이 가능하고, 그로써 단계(1220)가 무시될 수 있다. 게다가, 디코더(1020)는 또한 그에 수신되는 데이터 요소들을 적절한 방식으로 디코딩할 수 있기 위해 wrapValue를 알고 있어야만 한다.

[0258] 플러스 값들만을 달성하기 위해 오프셋을 이용함으로써, 데이터(DA2 또는 DA3)에서의 보다 효율적인 데이터 압축이 달성될 수 있다. 데이터 값들 모두가 이미 플러스 값들인 경우, 어떤 오프셋 값도 가산할 필요가 없다. 물론, 다음 예에서 보여지는 바와 같이, 이용가능한 범위를 감소시키기 위해 마이너스 오프셋 값들이 임의로 이용되지만, 이는 필수적인 것이 아니다.

[0259] 도 7 및 도 8에서의 방법들이 ODelta 코딩된 이용가능한 값들만을 사용하는 것에 의해 임의로 추가로 최적화된다. 이러한 최적화는 사용되는 값들을 알고 있을 것을 필요로 한다. 예를 들어, 전술한 예에서, 1(= 원래 최소 값)부터 126(= 원래 최대 값)까지의 값들만이 원래 데이터 세트(DA1)에 존재한다. 오프셋 값은 그러면 1(-> lowValue = 원래 최소 값 - 오프셋 = 1 - 1 = 0이고 highValue = 원래 최대 값 - 오프셋 = 126 - 1 = 125임)이다. 사전 오프셋 값이 원래 데이터(DA1)로부터 감소될 때, 하기의 수학식 15에서의 값들이 그로써 발생된다:

수학식 15

[0260] **64, 79, 125, 0, 61, 44, 88, 53, 65**

[0261] 수학식 15로부터, 125의 최대 값이 결정되고(highValue = 원래 최대 값 - 오프셋 = 126 - 1 = 125임), 따라서 "number"(= 최대 Delta 값 = highValue - lowValue)가 이제 125일 수 있거나, wrapValue가 최소 126(= 숫자 + 1 = highValue - lowValue + 1)일 수 있다. 이제, 이 값들을 저장 및/또는 전달하는 것이 필요하고, 이어서 이전 예가 다음과 같이 프로세스 값들을 변경하는 것에 의해 수정될 수 있다.

wrapValue = 126 (“0” to “125” => 126 different values)

prediction Value = (highValue + lowValue + 1) div 2 = (wrapValue + 1) div 2 +

[0262] lowValue = 63

[0263] 대응하는 직접 ODelta 연산자 값들은 수학식 16에 제공된다:

수학식 16

[0264] **1, 15, 46, 1, 61, 109, 44, 91, 12**

[0265] 모든 "마이너스 Delta 값들"이 이제 2의 인자만큼 감소된다(즉 = 범위 변화 = 128 - 126)는 것을 잘 알 것이다. 이와 유사하게, 디코더(1020)에서, 프로세스 값들이 다음과 같이 변경되어야만 한다:

wrapValue = 126

[0266] predictionValue = (wrapValue + 1) div 2 + lowValue = 63

[0267] 대응하는 역 ODelta 값들은 하기의 수학식 17이다:

수학식 17

[0268] **64, 79, 125, 0, 61, 44, 88, 53, 65**

[0269] 사전 오프셋 값이 수학식 17에 가산될 때, 수학식 15에서의 원래 데이터에 대응하는 하기의 수학식 18의 결과가 획득된다, 즉:

수학식 18

[0270] **65, 80, 126, 1, 62, 45, 89, 54, 66**

[0271] 이 예에서, 값들의 범위는 거의 전체 범위이고, 따라서 오프셋 및 최대 값들(highValue)과 함께 직접 ODelta 연산자를 적용하는 것으로부터 비교적 무난한 이점이 도출된다. 그렇지만, 엔트로피(E)의 감소가 여전히 달성될 수 있다 - 즉, 값들이 적절히 전달될 때 도수분포표(frequency table)에 또는 코드표에 보다 적은 값들이 있다 -. 범위가 덜 사용될 때 최대 이점들이 달성될 수 있다.

[0272] 실제의 1-비트 직접 및 역 ODelta 방법 - 즉, 방법 1 또는 방법 3 - 의 예시적인 실시예가 이제부터 실행가능 컴퓨터 소프트웨어 코드에 의해 제공될 것이고; 본 방법은 상기한 직접 및 역 ODelta 연산자들 - 즉, 방법 1 또는 방법 3 - 을 이용한다. 소프트웨어 코드는, 컴퓨팅 하드웨어 상에서 실행될 때, 하나의 바이트 버퍼로부터의 비트들을 처리하여 다른 바이트 버퍼에 넣기 위해 동작가능하다. 소프트웨어 코드에서, GetBit, SetBit 및 ClearBit 함수들은 항상 HeaderBits 값을 업데이트한다. 다음 비트가 다음 바이트에 있게 될 때 HeaderIndex 값이 또한 업데이트된다. 임의로, 소스 및 목적지에 대해 HeaderIndex 및 HeaderBits 값들의 하나의 세트만이 사용되도록, 소프트웨어 코드가 최적화될 수 있고, 따라서 주어진 비트가 목적지 버퍼에 쓰여질 때에만 값들이 업데이트된다.

```
procedure EncodeODelta1u(APtrSrc : PByte; ASrcDstBitLen : PCardinal; APtrDst : PByte)
var
  iSrcHeaderIndex, iSrcHeaderBits, iIndex,
  iDstHeaderIndex, iDstHeaderBits : Cardinal;
  bBit, bLastBit : Boolean;
begin
  // Reset offsets
  iSrcHeaderIndex := 0;
  iSrcHeaderBits := 0;
  iDstHeaderIndex := 0;
  iDstHeaderBits := 0;

  // Initialise delta value
  bLastBit := False;

  // Go through all bits
  for iIndex := 0 to ASrcDstBitLen^-1 do
  begin
    // Read bit
    bBit := GetBit(APtrSrc, @iSrcHeaderIndex, @iSrcHeaderBits);

    // Set destination bit if current source bit is different than previous source bit
    if (bBit <> bLastBit) then
    begin
      SetBit(APtrDst, @iDstHeaderIndex, @iDstHeaderBits);
      bLastBit := bBit;
    end
    else ClearBit(APtrDst, @iDstHeaderIndex, @iDstHeaderBits);
  end;
end;

function DecodeODelta1u(APtrSrc : PByte; ASrcDstBitLen : PCardinal; APtrDst : PByte) :
Boolean;
var
```

[0273]

```
  iSrcHeaderIndex, iSrcHeaderBits, iIndex,
  iDstHeaderIndex, iDstHeaderBits : Cardinal;
  bBit, bLastBit : Boolean;
begin
  // Reset offsets
  iSrcHeaderIndex := 0;
  iSrcHeaderBits := 0;
  iDstHeaderIndex := 0;
  iDstHeaderBits := 0;

  // Initialise delta value
  bLastBit := False;

  // Go through all bits
  for iIndex := 0 to ASrcDstBitLen^-1 do
  begin
    // Read bit
    bBit := GetBit(APtrSrc, @iSrcHeaderIndex, @iSrcHeaderBits);

    // Change bit value if source bit is true
    if (bBit = True) then
    begin
      if (bLastBit = True) then
        bLastBit := False
      else bLastBit := True;
    end;

    // Set destination bit based on bit value (True or False)
    if (bLastBit) then
      SetBit(APtrDst, @iDstHeaderIndex, @iDstHeaderBits)
    else ClearBit(APtrDst, @iDstHeaderIndex, @iDstHeaderBits);
  end;
end;
```

[0274]

[0275] 앞서 언급된 직접 및 역 ODelta 연산자들 - 즉, 방법 1 또는 방법 3 - 은 유익하게도 디지털 포맷으로 되어 있는 임의의 유형의 데이터 - 예를 들어, 비디오 데이터, 영상 데이터, 오디오 데이터, 그래픽 데이터, 지진 데이터, 의료 데이터, 측정 값들, 참조 수치들 및 마스크들 - 를 압축하는 데 이용된다. 더욱이, 하나 이상의 아날로그 신호들이 또한, 대응하는 디지털 데이터로 최초로 변환될 때, 예를 들어, 압축 이전에 ADC를 사용하는 것에 의해, 직접 ODelta 연산자를 사용하여 압축가능하다. 역 ODelta 연산자가 사용될 때, 데이터가 다시 하나 이상의 아날로그 신호들로 변환되는 것이 요망되는 경우, 연산 이후에 DAC가 사용될 수 있다. 그렇지만, 직접 ODelta 연산자 자체가 데이터를 압축하는 데는 보통 효과적이지 않지만, 다른 인코딩 방법들 - 예를 들어, VLC(variable-length coding), 산술 코딩, 범위 코딩, RLE(run-length encoding), SRLE, EM(Entropy Modifier) 등 - 과 결합될 때 효과적인 데이터 압축을 제공할 수 있다는 것을 잘 알 것이다. 인코더(1010)에서 직접 ODelta 연산자가 이용된 후에 데이터(DA2)에 대해 이 인코딩 방법들이 사용된다. 그 결과 얻어지는 데이터가 디코더(1020)에 구현되는 역 ODelta 연산자로 전달되기 전에, 인코딩된 데이터(DA2)는 그에 대응하여 다시 디코딩되어야만 한다. ODelta 연산자는 또한 다른 유형들의 엔트로피 수정자(entropy modifier)들과 함께 사용될 수 있다. 특정의 상황에서, 직접 ODelta 연산자로 인해 엔트로피(E)가 증가될 수 있고, 데이터 압축 알고리즘들이 유익하게도 직접 ODelta 연산자가 유익한 데이터 압축 성능을 제공할 때에만 데이터를 인코딩하는 데 사용하기 위해 직접 ODelta 연산자를 선택적으로 이용하기 위해 동작가능하다 - 예를 들어, 직접 ODelta 연산자가 압축될 데이터의 성격에 기초하여 선택적으로 이용되고, 예를 들어, 앞서 언급된 바와 같이 입력 데이터(DA1)의 선택된 부분들에 선택적으로 적용됨 -.

[0276] 직접 ODelta 연산자는, 예를 들어, 미국 특허 출원 US 13/584,005 - 그 내용은 이로써 참고로 포함됨 - 에 기술

된 바와 같이 블록 인코더와 결합되어 이용되도록 고안되었고, 역 ODelta 연산자는 미국 특허 출원 US 13/584,047 - 그 내용은 이로써 참고로 포함됨 - 에 기술된 바와 같이 블록 인코더와 결합되어 이용되도록 고안되었다. 임의로, 직접 ODelta 연산자 및 역 ODelta 연산자는 유익하게도 미국 특허 출원 US13/657,382 - 그 내용은 이로써 참고로 포함됨 - 에 기술된 바와 같이 멀티레벨 코딩 방법과 결합되어 이용된다. 유익하게도, 대응하는 변환된 데이터 - 이에 대해 이어서 그 후에 인코딩된 데이터(DA2 또는 DA3)를 발생시키기 위해 실제 엔트로피 인코딩이 적용됨 - 를 발생시키기 위해, 이진 상태들을 포함하는, 예를 들어, 데이터(DA1)에 존재하는, 모든 유형들의 1-비트 데이터에 직접 ODelta 연산자의 1-비트 버전이 적용된다. 임의로, 앞서 언급된 바와 같이, 직접 ODelta 연산자가 원래 데이터(DA1)의 성격에 따라 선택적으로, 이용된다.

[0277] 임의로, 직접 ODelta 연산자 이전에 또는 그 이후에 데이터의 엔트로피를 수정하는 다른 방법들을 이용하는 것이 실현가능하다. 예를 들어, 직접 ODelta 연산자가 또한 직접 ODelta 연산자의 일반화된 버전 내에서 다중 비트 데이터에 대해 직접 사용될 수 있다. 더욱이, 사용된 비트들 모두가 직렬 비트 시퀀스에 먼저 넣어진 후에, 직접 ODelta 연산자의 상기한 1-비트 버전이 유익하게도 다중 비트 데이터에 대해 이용된다.

[0278] 인코더(1010)에서 직접 ODelta 연산자와 함께 데이터 압축을 위해 다수의 방법들이 이용될 때, 대응하는 역 연산들이, 예를 들어, 디코더(1020)에서 역순으로 수행된다.

[0279] 하기의 방법들의 시퀀스가 인코더(1010)에서 이용된다:

수학식 19

[0280]
```
[data DA1] => direct ODelta (method 2)
        => VLC-
        => EM
        => Arithmetic coding
        => [data DA3]
```

[0281] 하기의 방법들의 역 시퀀스가 디코더(1020)에서 이용되고:

수학식 20

[0282]
```
[data DA3] => inverse Arithmetic coding
        => inverse EM
        => inverse VLC
        => inverse ODelta (method2)
        => [data DA5]
```

[0283] 여기서 "VLC"는 가변 길이 코딩(variable-length coding)을 나타내고, "EM"은 엔트로피 수정(entropy modifying)을 나타낸다.

[0284] ODelta 연산자는 앞서 기술된 바와 같이 가역적이기 무손실이다. 더욱이, ODelta 연산자는 임의로 특히 1-비트 데이터 스트림들에 대해 - 예를 들어, 비트 단위 인코딩을 수행할 때 - 또한 다른 데이터에 대해서도 구현될 수 있다. 유익하게도, 모든 유형들의 데이터가 직접 ODelta 연산자의 일반화된 버전을 사용하여 처리될 수 있다. 유익하게도, 직접 ODelta 연산자는 데이터가 압축되어야 할 때 이용되고, 대응하는 역 ODelta 연산자는 압축된 데이터가 압축 해제되어야 할 때 이용된다. 임의로, ODelta 연산자가 이용될 때, 직접 ODelta 연산자 및 그의 대응하는 역 연산이 역순으로 이용되고; 환언하면, 원래 비트스트림을 재생성하기 위해, 역 ODelta 연산자가 원래 비트스트림에 대해 시간상 먼저 수행되고, 그후에 직접 ODelta 연산자가 뒤따른다. 하나의 ODelta 연산자는 엔트로피를 증가시키고, 다른 ODelta 연산자는 엔트로피를 감소시킨다. 직접 ODelta 연산자가 엔트로피를 전혀

수정해서는 안되고 이어서 역 ODelta 연산자도 엔트로피를 수정하지 않는 것은 아주 드문 경우이다. 유의할 점은, 예를 들어, 방법 1에 대해, 직접 ODelta 연산자와 역 ODelta 연산자가 사용될 때, 이 연산들의 역순은 방법 4의 정상 순서와 유사하다는 것이다. 방법 2와 방법 3에 대해서도 유사한 순서 변경이 가능하다.

[0285] 1-비트 버전에서 - 즉, 데이터를 비트 단위 방식으로 인코딩하는 것에 대해 -, 직접 ODelta 연산자는 유익하게도 예측 없이 시작한다 - 즉, 기본적으로 초기 "0" 값의 예측을 가정한다 -. 일반화된 버전에서, ODelta 연산자는 사용가능 데이터 범위의 1/2을 나타내는 예측으로 시작하고; 예를 들어, 데이터(D1)에서의 입력 데이터 값들에 대해 5 비트 - 즉, "0"부터 "31"까지의 범위에 있는 32개의 상이한 값들 - 가 사용되는 경우, 예측 값은 32/2 = 16이다. 유익하게도, ODelta 연산자는 연산자를 사용하여 처리될 데이터 요소들에 대한 사용가능 데이터 범위에 관한 정보를 제공받을 필요가 있다.

[0286] 이상에서 기술된 본 개시내용의 실시예들은 비트들 또는 임의의 디지털 값들로서 데이터(DA1)에 제공되는 엔트로피(E)를 감소시키는 것을 가능하게 만든다. 직접 ODelta 연산자는 Delta 코딩에 비해 개선된 엔트로피 감소를 거의 항상 제공한다. Delta 코딩이 바이트 랩어라운드와 함께 사용되고 원래 예측(방법 1)과의 차이 ODelta 연산이 값들 wrapValue=256, lowValue=MIN=0, 및 highValue=MAX=255를 사용하는 경우만 그 내에 동일한 출력 결과를 생성한다. 다른 직접 ODelta 방법이 사용되는 경우, 또는 데이터 범위 전체가 입력 데이터에서 이용가능하지는 않은 경우, ODelta 연산자는 선택된 방법 또는 lowValue 및/또는 highValue - 즉, 이는 또한 wrapValue를 자동으로 수정함 - 을 송신하는 것에 의해 보다 나은 결과들을 생성한다. 보다 작은 엔트로피는 데이터가 보다 높은 데이터 압축 비로 압축될 수 있게 한다. 보다 높은 데이터 압축 비는 보다 작은 용량의 데이터 저장소가 이용될 수 있게 하고, 또한 보다 느린 데이터 대역폭이 압축된 데이터를 전달할 때 이용될 수 있게 하며, 그에 대응하여 에너지 소비가 감소된다.

[0287] 이상에서, 차이 및 합 계산의 형태가 인코더(1010)에서 실행되고 대응하는 역 계산이 디코더(1020)에서 수행된다는 것을 잘 알 것이다. 인코더(1010)에서 사용되는 다른 예측 방법을 사용하는 것이 또한 가능하고, 대응하는 역 예측이 이어서 디코더(1020)에서 수행된다. 이것은 실제로 적어도 4개의 상이한 직접 ODelta 방법들은 물론 적어도 4개의 대응하는 역 ODelta 방법들이 있다는 것을 의미한다. 이 방법들의 상세하고 정확한 설명은 다음과 같다. 임의로, 인코딩된 데이터(DA2)(또는 DA3)에서 보다 높은 정도의 데이터 압축을 달성하기 위해 계산들이 재귀적 방식으로 수행된다. 이러한 재귀적 계산들을 실행할 때, 몇 개의 재귀적 계산들이 이용되었는지의 함수로서 변하는 숫자 범위가 이용된다. 예를 들어, 인코더(1010)에서, 인코딩된 데이터(DA2)(또는 DA3)를 발생시키기 위해 하기의 계산 시퀀스가 데이터(DA1)에 대해 수행되고:

수학식 21

[0288]
$$
\begin{aligned}
&[\text{Data DA1}]\ \text{edirect ODelta (method 3)} \Rightarrow \\
&\quad \text{edirect ODelta (method 3)} \Rightarrow \\
&\quad \text{eEM} \Rightarrow \\
&\quad \text{edirect ODelta (method 1)} \Rightarrow \\
&\quad \text{eVLC} \qquad [\text{Data DA3}]
\end{aligned}
$$

[0289]

[0290] 대응하는 역 연산들이 디코더(1020)에서 수행된다:

수학식 22

[0291]
$$
\begin{aligned}
&[\text{Data DA3}]\ \text{dVLC} \Rightarrow \\
&\quad \text{dinverse ODelta (method 1)} \Rightarrow \\
&\quad \text{dEM} \Rightarrow \\
&\quad \text{dinverse ODelta (method 3)} \Rightarrow \\
&\quad \text{dinverse ODelta (method 3)} \qquad [\text{Data DA5}]
\end{aligned}
$$

[0292] 수학식 21(수학식 21은 방법 1에 대응함), 수학식 22(수학식 22는 방법 2에 대응함), 수학식 23(수학식 23은 방법 3에 대응함), 및 수학식 24(수학식 24는 방법 4에 대응함)로 나타낸 바와 같은, 이 4개의 방법들에서 데이터가 처리될 때마다, 모든 방법들을 사용하려고 시도하는 것이 임의로 가능한데, 그 이유는 이 방법들 중 하나가 처리되는 데이터의 엔트로피를 다른 방법들보다 더 많이 감소시킬 수 있기 때문이다. 인코더(1010) 및/또는 디코더(1020) 내에서의 방법들의 사용을 최적화할 때, 동일하거나 상이한 방법들을 선택된 방법 또는 방법들과 같은 횟수로 그리고 그만큼 오래 사용하여, 요구된 데이터에서의 정보의 양에 비해, 엔트로피를 감소시키는 것이 유리하다. 이와 같이, 방법 1 내지 방법 4는 수치 값들을 인코딩하는 데 사용가능하고, 여기서 "수치 값들"은 그의 정의 내에 1-비트 데이터는 물론, 비트 단위 방식으로 인코딩된 비트 스트림에서와 같은, 비이진 숫자들은 물론, 다중 비트 값들을 포괄한다.

[0293] 차이 연산은 연속적인 숫자 값들의 나머지를 나타내고; 그에 대응하여, 합 연산은 연속적인 숫자 값들의 합을 나타낸다. 인코더(1010)에서 실행되는 바와 같은 이 연산들은 디코더(1020)에서의 그 자신의 대응하는 역 연산들을 갖는다. 차이 또는 합은 현재 입력 값 및 예측 값으로서 사용되는 이전 입력 또는 결과 값에 기초하여 계산될 수 있다. 다른 예측 값들이 또한 사용될 수 있을 것이고, 예를 들어, 디코더(1020)에서도 그렇게 하는 것이 가역적인 한, 예측을 생성하기 위해 인코더(1010)에서 보다 이전의 입력 및 출력 값들을 사용할 수 있다.

[0294] 이 방법들 중 어느 것도 인코더(1010) 및 디코더(1020) 내에서 데이터를 그다지 압축하지 않지만, 모든 방법들이 유익하게도 엔트로피를 감소시키는 데 이용되며, 따라서 다른 압축 방법이 이어서 엔트로피 감소된 데이터를 보다 효율적으로 압축할 수 있다. 이러한 다른 압축 방법은 임의로 Huffman 코딩, 산술 코딩, 범위 코딩, RLE 코딩, SRLE 코딩, 엔트로피 수정자 코딩 중 적어도 하나이다. 그렇지만, 모든 방법들에 대해, 예를 들어, 데이터의 무손실 압축 및 차후의 무손실 압축 해제가 달성되어야 하는 경우, 연산 및 그의 역 연산이 항상 정확하게 실행될 수 있는 몇 개의 숫자 값들을 전달하는 것이 필요하다. 물론, 인코더(1010) 및 디코더(1020)는 어느 종류의 숫자 값들이 입력 데이터(DA1)에 포함되어 있는지에 관한 정보를 갖는다. 유익하게도, 숫자 범위 - 즉, MIN과 MAX에 의해 정의됨 - 가 알려져 있는 것으로 가정된다. 원칙적으로, 방법들은 항상 기존의 데이터 범위에 직접 기초하여 기능할 수 있다. 연산들이 필요로 하는 숫자 값들은 나오는 가장 낮은 숫자 값(lowValue) 및 나오는 가장 높은 숫자 값(highValue)이고; lowValue는 MIN보다 크거나 같고 highValue는 MAX보다 작거나 같다.

[0295] 이 값들에 기초하여, 다른 필요한 숫자 값들이 도출될 수 있다. 유익하게도, 이 값들은 다양한 형태들로 전달되고, 여기서 누락된 값들은 유익하게도 계산된다. 예를 들어, 세트 ["lowValue", "highValue", "number"]로부터의 2개의 값들이 알려져 있고, "number"가 [highValue - lowValue]인 경우, 제3 값은 그로부터 계산될 수 있다. 데이터(DA2)에서의 특정 값들을 생략하고 이어서 그들을 디코더(1020)에서 도출하는 것은 데이터(DA2)에서의 보다 큰 데이터 압축을 제공할 수 있다.

[0296] 이 값들에 부가하여, 첫 번째 값의 계산에서 이전 값으로서 사용될 수 있는 숫자 P - 즉, "prediction" - 가 필요하다. "0"과 "number" 값 사이의 값이 숫자 P - 즉, "prediction" - 에 대해 항상 선택될 수 있다. 더욱이, 상기한 연산들은 디코더(1020)에서 데이터(DA2/DA3 또는 DA4)를 디코딩할 때 복구가능하게 기능하기 위해 - 즉, 연산들이 발생시키는 값 범위를 가능한 한 작게 축소시키기 위해 - 값 "wrapValue"를 제공받을 필요가 있다. 그렇지만, 이 "wrapValue"는 "number"보다 더 커야만 하고, 유익하게도 값 "number" + 1을 가질 것이다. 임의로, 데이터(DA1)의 성격에 따라, 첫 번째 "prediction" 값은, 예를 들어, 데이터(DA1)가 보다 큰 값들보다 더 많은 보다 작은 값들을 포함하는 것으로 가정되는 경우, 앞서 언급된 바와 같이 "0"으로 선택될 수 있고; 대안적으로, 데이터(DA1)가 보다 작은 값들보다 더 많은 보다 큰 값들을 포함하는 것으로 가정되는 경우, 첫 번째 "prediction" 값이 "number"와 같도록 선택될 수 있다. 값들의 크기에 대해 가정이 행해지지 않은 경우에, "prediction" 값에 대해 값 "(wrapValue + 1) div 2 + lowValue"를 사용하는 것이 바람직하다.

[0297] 본 개시내용의 실시예들을 구현할 때 컴퓨팅 하드웨어에서 수행되는 연산들의 예들이 이제부터 기술될 것이다.

[0298] 인코더(1010)에서, 첫 번째 직접 차이 연산(direct difference operation) - 즉, 방법 1 - 은 유익하게도 다음과 같이 구현되고; 데이터 값들 모두에 대해, 입력 값 - 즉, "original" 값 - 에 대응하는 출력 값 - 즉, "result" - 은 소프트웨어 루프에서 계산된다:

result = original – prediction

[0299] if result < lowValue then result = result + wrapValue

[0300] 마지막으로, 다음 입력에 대한 예측 값은 현재 입력과 같게 설정되고, 즉:

[0301] prediction = original

[0302] 디코더(1020)에서, 첫 번째 역 차이 연산(inverse difference operation) - 즉, 방법 1 - 은 유익하게도 다음과 같이 구현되고; 데이터 값들 모두에 대해, 입력 값 - 즉, "original" 값 - 에 대응하는 출력 값 - 즉, "result" - 은 소프트웨어 루프에서 계산된다:

result = original + prediction

[0303] if result > highValue then result = result - wrapValue

[0304] 마지막으로, 다음 입력에 대한 예측 값은 현재 결과와 같게 설정되고, 즉:

[0305] prediction = result

[0306] 인코더(1010)에서, 두 번째 직접 차이 연산 - 즉, 방법 2 - 은 유익하게도 다음과 같이 구현되고; 데이터 값들 모두에 대해, 입력 값 - 즉, "original" 값 - 에 대응하는 출력 값 - 즉, "result" - 은 소프트웨어 루프에서 계산된다:

result = original – prediction

[0307] if result < lowValue then result = result + wrapValue

[0308] 마지막으로, 다음 입력에 대한 예측 값은 현재 결과와 같게 설정되고, 즉:

[0309] prediction = result

[0310] 디코더(1020)에서, 두 번째 역 차이 연산 - 즉, 방법 2 - 은 유익하게도 다음과 같이 구현되고; 데이터 값들 모두에 대해, 입력 값 - 즉, "original" 값 - 에 대응하는 출력 값 - 즉, "result" - 은 소프트웨어 루프에서 계산된다:

result = original + prediction

[0311] if result > highValue then result = result - wrapValue

[0312] 마지막으로, 다음 입력에 대한 예측 값은 현재 입력과 같게 설정되고, 즉:

[0313] prediction = original

[0314] 인코더(1010)에서, 첫 번째 직접 합 연산(direct sum operation) - 즉, 방법 3 - 은 유익하게도 다음과 같이 구현되고: 데이터 값들 모두에 대해, 입력 값 - 즉, "original" 값 - 에 대응하는 입력 값 - 즉, "result" - 은 다음과 같이 소프트웨어 루프에서 계산된다:

result = original + prediction

[0315] if result > highValue then result = result – wrapValue

[0316] 마지막으로, 다음 입력에 대한 예측 값은 다음과 같이 현재 입력과 같게 설정되고:

[0317] prediction = original

[0318] 디코더(1020)에서, 첫 번째 역 합 연산(inverse sum operation) - 즉, 방법 3 - 은 유익하게도 다음과 같이 구현되고: 데이터 값들 모두에 대해, 입력 값 - 즉, "original" 값 - 에 대응하는 입력 값 - 즉, "result" - 은 다음과 같이 소프트웨어 루프에서 계산된다:

result = original - prediction

[0319] if result < lowValue then result = result + wrapValue

[0320] 마지막으로, 다음 입력에 대한 예측 값은 현재 결과와 같게 설정되고, 즉:

[0321] prediction = result

[0322] 인코더(1010)에서, 두 번째 직접 합 연산 - 즉, 방법 4 - 은 유익하게도 다음과 같이 구현되고: 데이터 값들 모두에 대해, 입력 값 - 즉, "original" 값 - 에 대응하는 입력 값 - 즉, "result" - 은 다음과 같이 소프트웨어 루프에서 계산된다:

result = original + prediction

[0323] if result > highValue then result = result – wrapValue

[0324] 마지막으로, 다음 입력에 대한 예측 값은 다음과 같이 현재 결과와 같게 설정된다:

[0325] prediction = result

[0326] 디코더(1020)에서, 두 번째 역 합 연산 - 즉, 방법 4 - 은 유익하게도 다음과 같이 구현되고: 데이터 값들 모두에 대해, 입력 값 - 즉, "original" 값 - 에 대응하는 입력 값 - 즉, "result" - 은 다음과 같이 소프트웨어 루프에서 계산된다:

result = original - prediction

[0327] if result < lowValue then result = result + wrapValue

[0328] 마지막으로, 다음 입력에 대한 예측 값은 현재 입력과 같게 설정되고, 즉:

[0329] prediction = original

[0330] 이러한 합 및 차이 연산들 - 4개의 방법들 모두 - 이 또한, 즉 인코더(1010) 및 디코더(1020)의 ODelta 버전들을 구현할 때, 1-비트 데이터에 - 즉, 비트 단위로 - 적용가능하다. 1-비트 데이터의 상황에서, 다음 값들은 인코더(1010) 및 디코더(1020) 둘 다가 이미 알고 있다 - 즉, MIN = 0이고, MAX = 1임 -. 더욱이, 유익하게도 lowValue = MIN = 0이고, highValue = MAX = 1인 것으로 가정된다. 게다가, 이러한 경우에, "number"는 따라서 [highValue - lowValue = 1 - 0 = 1]이고, wrapValue는 유익하게도 "number" + 1 = 1 + 1 = 2로 선택된다. 유익하게도, 예측 값은 값 "0"으로 선택되는데, 그 이유는 lowValue = MIN = 0으로부터 시작하는 플러스 값들만을 가질 수 있는 1-비트 데이터만이 고려되기 때문이다. 1-비트 데이터에 대해, 방법 1 및 방법 3은 서로 유사한 코딩 결과들을 산출한다. 이와 유사하게, 방법 2 및 방법 4는 서로 유사한 코딩 결과들을 산출한다. 이러한 지식을 가지고 있는 것은 유익하게도 데이터(DA2)에서 송신될 필요가 있는 정보를 단순화시키는데, 그 이유는 다양한 기본값들이 가정될 수 있기 때문이다 - 즉, 차이 연산들(방법 1 또는 방법 2 중 어느 하나)의 실행 횟수, 및 선택된 예측(입력 값(방법 1) 또는 결과 값(방법 2))에 관한 정보를 송신하기만 하면 되고, 따라서 디코더(1020)는 디코딩된 데이터(DA5)를 발생시키기 위해 데이터(DA2, DA3 또는 DA4)를 디코딩할 때 올바른 역 차이 연산을 필요한 횟수만큼 실행할 수 있다 -.

[0331] 유사한 출력을 생성하는 방법 1 또는 방법 3을 사용하여 생성된 제1 예는 또한, 유사한 출력을 역시 생성하는, 방법 2 또는 방법 4 중 어느 하나를 사용하여 처리될 수 있다. 이하에 보여지는 결과는 수학식 1의 데이터에 적용될 때 그 방법들에 의해 달성될 수 있다:

[0332] **0 1 1 0 0 1 0 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0 1**

[0333] 이 때, 처리된 데이터는 24개의 "1"과 13개의 "0"을 갖지만 - 즉, 엔트로피는 제1 예에서와 동일하지만 -, "1"의 개수와 "0"의 개수가 자리를 바꾼다. 이것이 항상 일어나지는 않으며, 그 대신에 종종 이 상이한 방법들 간에 엔트로피도 변한다. 예를 들어, 처음 4개의 데이터 요소들 후에, 방법 1 및/또는 방법 3은 3개의 "1"과 하나의 "0"을 생성하는 반면, 원래 데이터와 방법 2 및/또는 방법 4로 처리된 데이터는 2개의 "1" 및 2개의 "0"을 갖는다. 따라서, 방법 1 및/또는 방법 3은, 이러한 경우에, 방법 2 및/또는 방법 4보다 더 작은 엔트로피를 그

리고 또한 당초보다 더 작은 엔트로피를 생성한다.

[0334] 다중 비트 구현에서, 데이터(DA1)가 -64부터 +63까지의 범위에 있는 값들을 포함하는 경우, MIN = -64이고 MAX = 63이다. lowValue = MIN이고 highValue = MAX인 것으로 가정하는 것에 의해, "number" = 127이고 wrapValue는 유익하게도 128로 선택된다. 그렇지만, 데이터(DA1)가 랜덤하게 변할 때, "prediction" 값은 유익하게도 값 [(wrapValue + 1) div 2 + lowValue = 64 + -64 = 0]으로 설정된다.

[0335] 첫 번째 값이, 예를 들어, -1인 경우, 직접 ODelta 방법 1 및/또는 방법 2에 의한 첫 번째 코딩된 값은 -1 - 0 = -1일 것이고, 그에 대응하여, 직접 ODelta 방법 3, 및/또는 방법 4에 의한 첫 번째 코딩된 값은 -1 + 0 = - 1일 것이다. 다음 값들은 그러면 데이터가 어떻게 진행되느냐에 따라 변할 것이다 - 예를 들어, 두 번째 값이 5인 경우, 직접 ODelta 방법 1은 5 - -1 = 6을 생성할 것이고, 직접 ODelta 방법 2는 5 - -1 = 6을 생성할 것이며, 직접 ODelta 방법 3은 5 + -1 = 4를 생성할 것이고, 직접 ODelta 방법 4는 5 + -1 = 4를 생성할 것이다. 디코더(1020)는, 이 경우에, 첫 번째 값으로서 역 ODelta 방법 1 및/또는 방법 2를 사용할 때는 -1 + 0 = -1을 그리고 역 ODelta 방법 3, 및/또는 방법 4에 의해서는 -1 - 0 = -1을 생성할 수 있을 것이다. 그에 대응하여, 두 번째 값은 역 ODelta 방법 1에 의해서는 6 + -1 = 5일 것이고, 역 ODelta 방법 2에 의해서는 6 + -1 = 5일 것이며, 역 ODelta 방법 3에 의해서는 4 - -1 = 5일 것이고, 역 ODelta 방법 4에 의해서는 4 - -1 = 5일 것이다.

[0336] 이러한 해결책은 그러면 숫자 범위가 실제로 -20부터 +27 사이의 값들만을 포함하는 경우 최적화될 수 있다. 이 예시적인 경우에, 예를 들어, lowValue = -20 및 highValue = 27을 전송하는 것이 실현가능하다. 둘 다가 전송되는 경우, number = 47인 것으로 계산하는 것이 실현가능하고, wrapValue가 그러면 유익하게도 48로 선택된다. 이제, 예측을 위해 값 48 div 2 + -20 = 4를 계산하는 것이 실현가능하다. 이어서, 이전 예는 , 예를 들어, 값 -1에 대해 ODelta 방법 1 또는 방법 2가 사용될 때는 -1 - 4 = -5를 그리고 ODelta 방법 3 또는 방법 4에 의해서는 -1 + 4 = 3을 산출할 것이다. 이와 유사하게, ODelta 방법들에 대한 두 번째 값은 (5 - -1) = 6, (5 - -5) = 10, (5 + -1) = 4, 및 (5 + 3) = 8일 것이다. 디코더(1020)에서의 디코딩은 또다시 올바르게 기능하고 방법 1 및/또는 방법 2에 대한 첫 번째 값을 -5 + 4 = -1로서 산출하고 방법 3 및/또는 방법 4에 대해서는 3 - 4 = -1로서 산출한다. 그에 대응하여, 상이한 방법들에 대한 두 번째 값들은 (6 + -1) = 5, (10 + -5) = 5, (4 - -1) = 5, 및 (8 - 3) = 5로서 디코딩될 것이다.

[0337] 상기 예들에서의 값들 모두가 범위 - 즉, -64부터 +63까지 또는 -20부터 +27까지 - 내에 있고, 따라서 이 예시적인 값들 내에서 보정 항(correction term)을 수행하는 것이 필요하지 않지만, 임의의 마이너스 또는 플러스 변화가 충분히 큰 경우, 결과 값들을 범위 내에 유지하기 위해 데이터 값들에 대한 보정이 주어진 수학식 21 내지 수학식 24에 의해 행해져야만 한다는 것을 잘 알 것이다. 유의할 점은 보정 항이 여기서 랩어라운드 값을 지칭한다는 것이다.

[0338] lowValue를 알고 있는 경우, 엔트로피 인코딩된 데이터(DA3)와 함께 인코더(1010)로부터 디코더(1020)로 송신되어야만 하는 코딩표(coding table)를 단순화시키기 위해, 코딩된 값들이 유익하게도 0부터 시작하여 값 "number"로 끝나도록 배열된다. 이 연산은 사후 오프셋이라고 불리우고, 이 사후 오프셋 값은 엔트로피 디코딩 후에 그리고 데이터(DA4)에 대한 역 ODelta 연산 이전에 코딩된 데이터 값들로부터 삭제되어야만 한다.

[0339] 앞서 언급한 바와 같이, 원래 입력 데이터(DA1)가 ODelta 방법의 실제 실행 이전에 이미 0부터 "number"까지의 값들을 포함할 수 있는 플러스 요소들로 변환되는 사전 오프셋 기능으로 오프셋을 구현하는 것이 또한 가능하다. 또한 이 상황에서, 이 연산이 필요로 하는 정보 전송은 "사전 오프셋" 및 ODelta 방법이 동일한 정보를 반복하여 전송하지 않거나 어떤 다른 방법을 통해 이미 알고 있는 것은 무시하는 방식으로 수행하는 데 유익하다. 이 사전 오프셋은 적절한 DA5 출력 데이터를 생성하기 위해 역 ODelta 연산 이후에 디코딩된 데이터로부터 삭제되어야만 한다.

[0340] [표 4]: 공지된 기술

표 4

[0341]

이전 문서	상세
P1	"Variable-length code", Wikipedia (2012년 11월 28일 접속함) URL: http://en.wikipedia.org/wiki/Variable-length_code

P2	"Run-length encoding", Wikipedia (2012년 11월 28일 접속함) URL: http://en.wikipedia.org/wiki/Run-length_encoding
P3	"Huffman coding", Wikipedia (2012년 11월 28일 접속함) URL: http://en.wikipedia.org/wiki/Huffman_coding
P4	"Arithmetic coding", Wikipedia (2012년 11월 28일 접속함) URL: http://en.wikipedia.org/wiki/Arithmetic_coding
P5	"A Mathematic Theory of Communication", Shannon, Claude E. (1948) (2012년 11월 28일 접속함) URL: http://cm:bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf
P6	"Delta encoding", Wikipedia (2012년 11월 28일 접속함) URL: http://en.wikipedia.org/wiki/Delta_encoding
P7	Shannon's source coding therorem; Wikipedia (2012년 11월 28일 접속함) URL: http://en.wikipedia.org/wiki/Source_coding_therorem
P8	"Entropy" - Wikipedia (2012년 11월 28일 접속함) URL: http://en.wikipedia.org/wiki/Entropy

[0342] **부록 2: 블록 인코더의 개요**

[0343] 블록 인코더는 특허문헌 GB2503295에 개시되어 있다. 그 내용은 참조에 의해 여기에 포함된다. 입력 데이터를 인코딩하는 방법 및 입력 데이터를 인코딩하도록 동작가능한 인코더가 제공된다.

[0344] 대응하는 인코딩된 출력 데이터를 생성하기 위해 입력 데이터를 인코딩하는 방법은,

[0345] (a) 상기 입력 데이터를 복수의 블록 또는 패킷 - 상기 블록 또는 패킷은 상기 블록 또는 패킷의 콘텐츠의 성질에 의존하는 크기를 가지며, 상기 블록 또는 패킷은 하나 이상의 크기를 가짐 - 으로 세분하는 단계;

[0346] (b) 대응하는 변환된 데이터를 생성하기 위해 상기 블록 또는 패킷의 콘텐츠에 복수의 변환을 적용하는 단계;

[0347] (c) 상기 변환된 데이터의 표현의 품질이 하나 이상의 품질 기준을 만족시키는지 여부를 결정하기 위해, 상기 변환을 적용하기 전의 상기 블록 또는 패킷의 콘텐츠와 비교하여 상기 블록 또는 패킷의 상기 변환된 데이터의 표현의 품질을 검사하는 단계;

[0348] (d) 상기 하나 이상의 블록 또는 패킷의 상기 변환된 데이터의 표현의 품질이 상기 하나 이상의 품질 기준을 만족시키지 않는 경우, 상기 하나 이상의 블록 또는 패킷을 추가로 세분 및/또는 결합하고, 단계 (b)를 반복하는 단계; 및

[0349] (e) 상기 하나 이상의 블록 또는 패킷의 상기 변환된 데이터의 표현의 품질이 상기 하나 이상의 품질 기준을 만족시키는 경우, 인코딩될 상기 입력 데이터를 나타내는 상기 인코딩된 출력 데이터를 제공하기 위해 상기 변환된 데이터를 출력하는 단계

[0350] 를 포함한다.

[0351] 대응하는 인코딩된 출력 데이터를 생성하기 위해 입력 데이터를 인코딩하도록 동작가능한 인코더는,

[0352] (a) 입력 데이터를 복수의 블록 또는 패킷 - 상기 블록 또는 패킷은 상기 블록 또는 패킷의 콘텐츠의 성질에 의존하는 크기를 가지며, 상기 블록 또는 패킷은 하나 이상의 크기를 가짐 - 으로 세분하고;

[0353] (b) 대응하는 변환된 데이터를 생성하기 위해 상기 블록 또는 패킷의 콘텐츠에 적어도 하나의 변환을 적용하며;

[0354] (c) 상기 변환된 데이터의 표현의 품질이 하나 이상의 품질 기준을 만족시키는지 여부를 결정하기 위해, 상기 변환을 적용하기 전의 상기 블록 또는 패킷의 콘텐츠와 비교하여 상기 블록 또는 패킷의 상기 변환된 데이터의 표현의 품질을 검사하고;

[0355] (d) 상기 하나 이상의 블록 또는 패킷의 상기 변환된 데이터의 표현의 품질이 상기 하나 이상의 품질 기준을 만족시키지 않는 경우, 상기 하나 이상의 블록 또는 패킷을 추가로 세분 및/또는 결합하고, 상기 (b)를 반복하며;

[0356] (e) 상기 하나 이상의 블록 또는 패킷의 상기 변환된 데이터의 표현의 품질이 상기 하나 이상의 품질 기준을 만족시키는 경우, 인코딩될 상기 입력 데이터를 나타내는 상기 인코딩된 출력 데이터를 제공하기 위해 상기 변환된 데이터를 출력하도록,

[0357] 동작가능한, 데이터 프로세싱 하드웨어를 포함한다.

[0358] **부록 3: 블록 디코더의 개요**

[0359] 블록 디코더의 개요도 특허문헌 GB2505169에 개시되어 있다. 그 내용은 참조에 의해 여기에 포함된다. 인코딩된 입력 데이터를 디코딩하는 방법 및 입력 데이터를 디코딩하도록 동작가능한 디코더가 제공된다.

[0360] 대응하는 디코딩된 출력 데이터를 생성하기 위해 인코딩된 입력 데이터를 디코딩하는 방법은,

[0361] (a) 인코딩된 입력 데이터에 포함되어 있는 블록 및/또는 패킷에 관한 인코딩된 데이터를 나타내는 헤더 정보 - 상기 헤더 정보는 상기 블록 및/또는 패킷에 관한 상기 인코딩된 데이터로서 포함시키기 위한 원래의 블록 및/또는 패킷 데이터를 인코딩하고 압축하는 데 이용되는 하나 이상의 변환을 나타내는 데이터를 포함함 - 를 상기 인코딩된 입력 데이터로부터 추출하기 위해, 상기 인코딩된 입력 데이터를 프로세싱하는 단계;

[0362] (b) 디코딩된 블록 및/또는 패킷 콘텐츠를 수신하기 위해 데이터 저장 장치에 데이터 필드를 어레인지하는 단계;

[0363] (c) 상기 데이터 필드를 채우기 위한 대응하는 디코딩된 블록 및/또는 패킷 콘텐츠를 생성하기 위해, 상기 하나 이상의 변환을 기술하는 정보를 검색하고 이어서 상기 인코딩되고 압축되는 원래의 블록 및/또는 패킷의 디코딩을 위한 상기 하나 이상의 변환의 역을 적용하는 단계;

[0364] (d) 상기 인코딩된 입력 데이터에 포함되어 있는 분할 및/또는 결합 정보에 따라 상기 데이터 필드에 있는 블록 및/또는 패킷을 분할 및/또는 결합하는 단계; 및

[0365] (e) 상기 인코딩된 입력 데이터가 적어도 부분적으로 디코딩되었을 때, 상기 데이터 필드로부터의 데이터를 상기 디코딩된 출력 데이터로서 출력하는 단계

[0366] 를 포함한다.

[0367] 대응하는 디코딩된 출력 데이터를 생성하기 위해 입력 데이터를 디코딩하도록 동작가능한 디코더는,

[0368] (a) 인코딩된 입력 데이터에 포함되어 있는 블록 및/또는 패킷에 관한 인코딩된 데이터를 나타내는 헤더 정보 - 상기 헤더 정보는 상기 블록 및/또는 패킷에 관한 상기 인코딩된 데이터로서 포함시키기 위한 원래의 블록 및/또는 패킷 데이터를 인코딩하고 압축하는 데 이용되는 하나 이상의 변환을 나타내는 데이터를 포함함 - 를 상기 인코딩된 입력 데이터(20)로부터 추출하기 위해, 상기 인코딩된 입력 데이터를 프로세싱하고;

[0369] (b) 디코딩된 블록 및/또는 패킷 콘텐츠를 수신하기 위해 데이터 저장 장치에 데이터 필드를 어레인지하며;

[0370] (c) 상기 데이터 필드를 채우기 위한 대응하는 디코딩된 블록 및/또는 패킷 콘텐츠를 생성하기 위해, 상기 하나 이상의 변환을 기술하는 정보를 검색하고 이어서 상기 인코딩되고 압축되는 원래의 블록 및/또는 패킷의 디코딩을 위한 상기 하나 이상의 변환의 역을 적용하고;

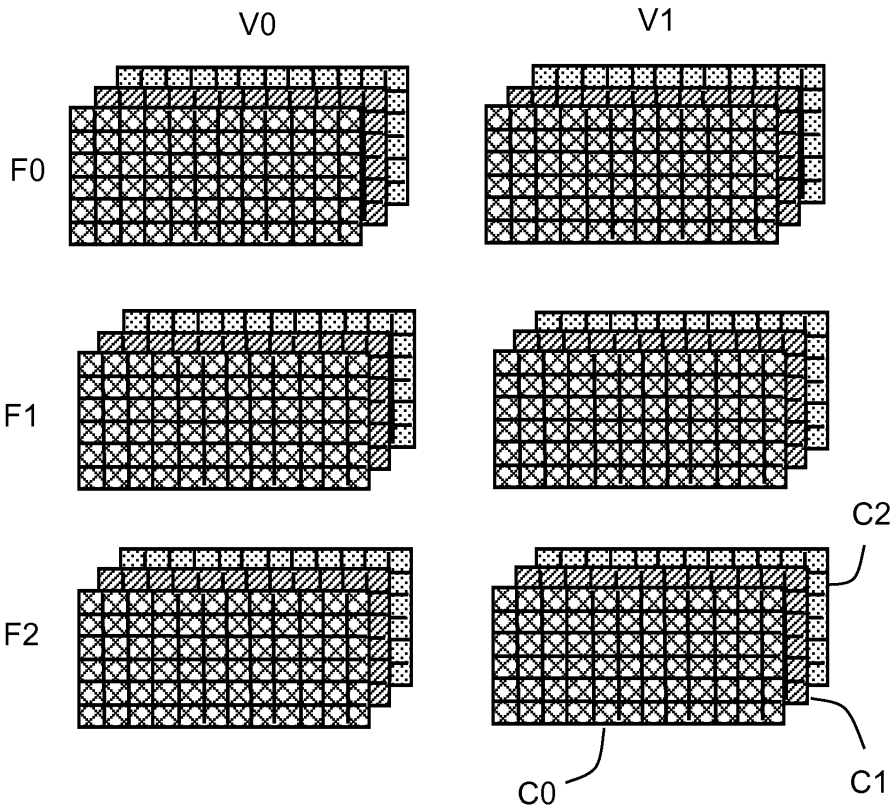
[0371] (d) 상기 인코딩된 입력 데이터에 포함되어 있는 분할 및/또는 결합 정보에 따라 상기 데이터 필드에 있는 블록 및/또는 패킷을 분할 및/또는 결합하며;

[0372] (e) 상기 인코딩된 입력 데이터가 적어도 부분적으로 디코딩되었을 때, 상기 데이터 필드로부터의 데이터를 상기 디코딩된 출력 데이터로서 출력하도록

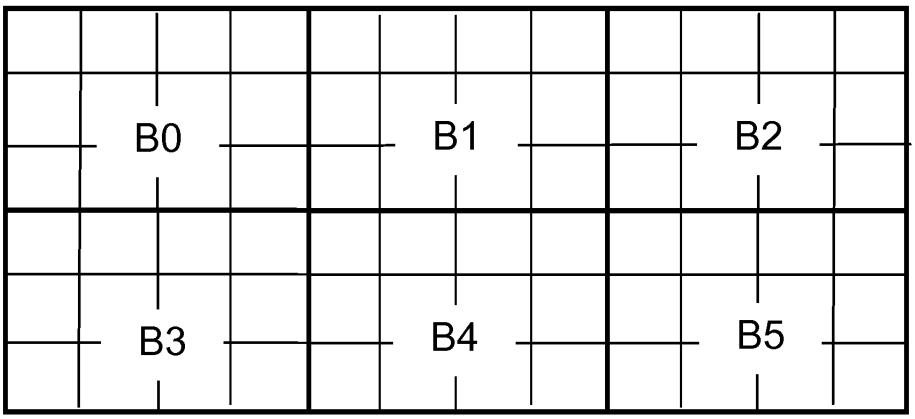
[0373] 동작가능한 데이터 프로세싱 하드웨어를 포함한다.

도면

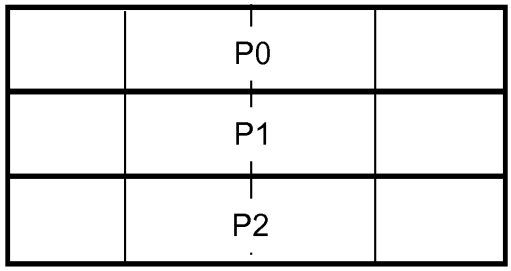
도면1

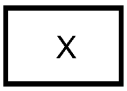


도면2

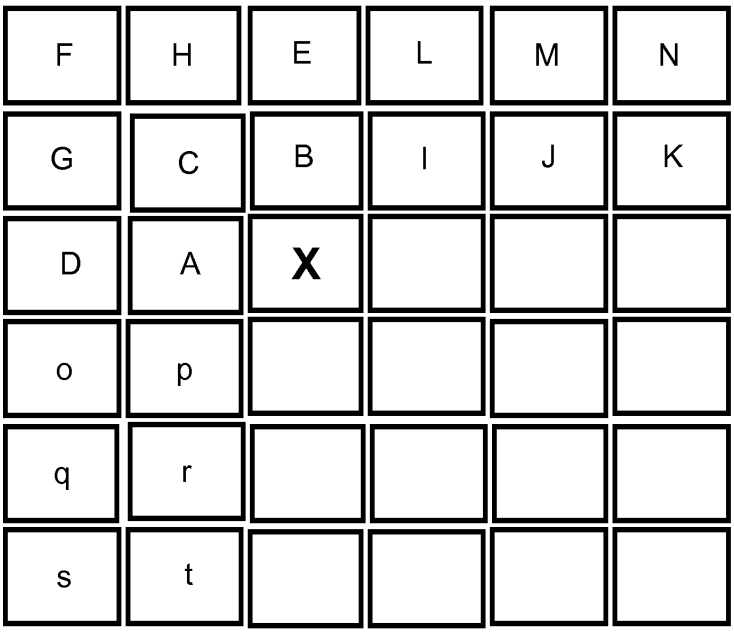


도면3

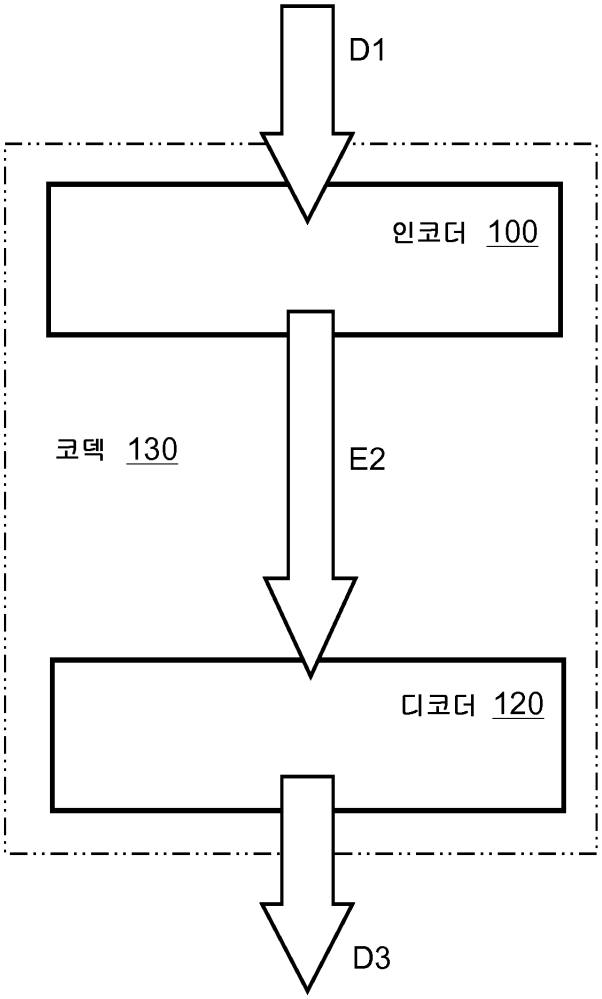




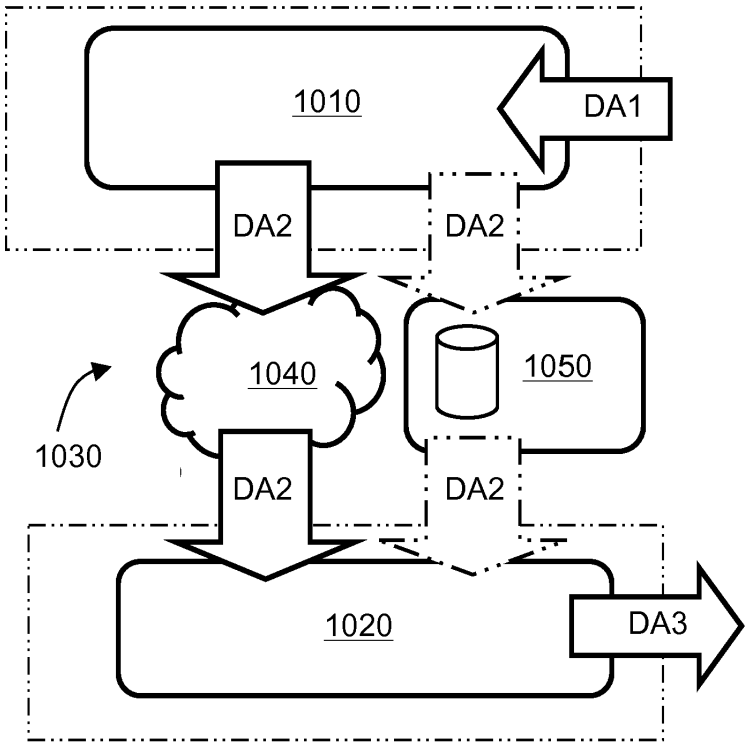
도면4



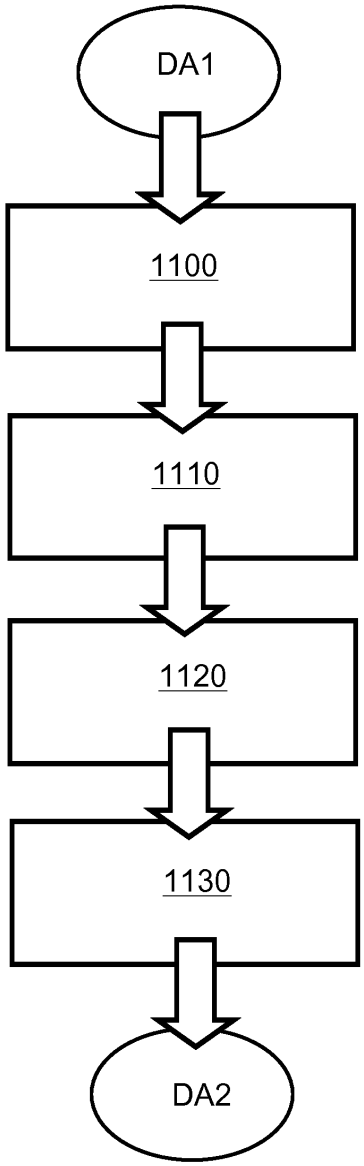
도면5



도면6



도면7



도면8