

(11) 特許出願公開番号

特開2010-4548

(P2010-4548A)

(43) 公開日 平成22年1月7日(2010.1.7)

(51) Int. Cl.
H04N 1/41

F I
H O 4 N 1/41

テーマコード (参考)
5C178

審査請求 有 請求項の数 22 O L (全 60 頁)

(21) 出願番号	特願2009-184108 (P2009-184108)	(71) 出願人	599163366
(22) 出願日	平成21年8月7日 (2009.8.7)		エレクトロソニック リミティド
(62) 分割の表示	特願2008-527499 (P2008-527499) の分割		イギリス国、ケント、ディーユー2 7エ スワイ、ダートフォード、ハウレー ロー ド、ハウレー ミル
原出願日	平成18年8月11日 (2006.8.11)	(74) 代理人	100067828
(31) 優先権主張番号	0517501.3		弁理士 小谷 悦司
(32) 優先日	平成17年8月26日 (2005.8.26)	(74) 代理人	100115381
(33) 優先権主張国	英国 (GB)		弁理士 小谷 昌崇
		(74) 代理人	100127797
			弁理士 平田 晴洋
		(72) 発明者	ピーター ライオネル、 スミス
			イギリス国 ケント ビーアール2 8ピ ービー ブロムリー オークリー ドライ ブ 14

最終頁に続く

[最終頁に続く](#)

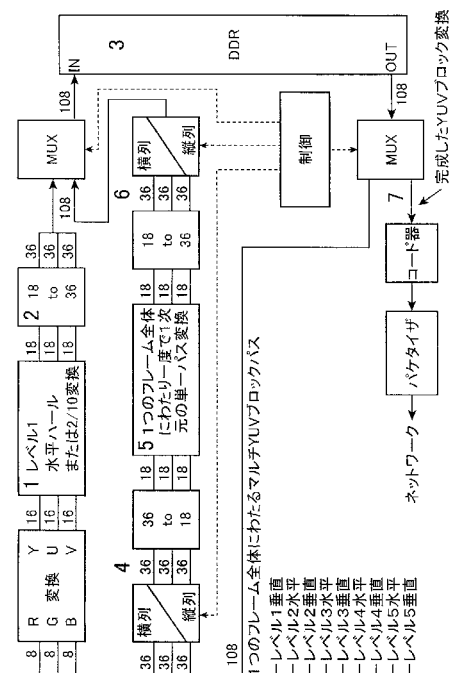
(54) 【発明の名称】 画像データ処理方法

(57) 【要約】 (修正有)

【課題】データの有効な符号化を可能とし、損失なし、または損失ありのいずれかである得る空間的および時間的両方の画像データ圧縮の方法を提供する。

【解決手段】第1および第2のデータグループのそれぞれに対して第1の変換を行うことで第1および第2の変換データグループをそれぞれ生成し、前記第1および第2の変換データグループのそれぞれに対して複数のその後の変換を行う方法であって、順次、第1の変換エンジンを使って前記第1のデータグループに前記第1の変換を行う工程、および、前記第1の変換エンジンが前記第2のデータグループに対して前記第1の変換を行う期間と少なくとも部分的に重複する期間内に、第2の変換エンジンを使って前記変換された第1のデータグループに前記後続の変換をすべて行う工程、を包含する。

【選択図】 図 1 2



【特許請求の範囲】

【請求項 1】

第 1 および第 2 のデータグループのそれぞれに対して第 1 の変換を行うことで第 1 および第 2 の変換データグループをそれぞれ生成し、前記第 1 および第 2 の変換データグループのそれぞれに対して複数のその後の変換を行う方法であって、順次、

第 1 の変換エンジンを使って前記第 1 のデータグループに前記第 1 の変換を行う工程、および

前記第 1 の変換エンジンが前記第 2 のデータグループに対して前記第 1 の変換を行う期間と少なくとも部分的に重複する期間内に、第 2 の変換エンジンを使って前記変換された第 1 のデータグループに前記後続の変換をすべて行う工程、
を包含することを特徴とする方法。

10

【請求項 2】

変換されたデータグループに対して前記後続の変換をすべて行うのにかかる時間は、第 1 の変換をそれに対して行うのにかかる時間と等しいまたはより短いことを特徴とする、請求項 1 に記載の方法。

【請求項 3】

第 1 のメモリ保存領域に前記第 1 のデータグループに対するそれぞれの変換から得られた変換データを保存する工程、および第 2 のメモリ保存領域に前記第 2 のデータグループに対するそれぞれの変換から得られた変換データを保存する工程をさらに包含することを特徴とする、請求項 1 または 2 に記載の方法。

20

【請求項 4】

さらに別の複数のデータグループに適用される方法であって、

前記第 1 のメモリ保存領域に前記またはさらに別の奇数番号のデータグループそれぞれに対するそれぞれの変換から得られた変換データを保存する工程と、

前記第 2 のメモリ保存領域に前記またはさらに別の偶数番号のデータグループそれぞれに対するそれぞれの変換から得られた変換データを保存する工程と、
を包含することを特徴とする、請求項 3 に記載の方法。

【請求項 5】

前記後続の変換のそれぞれが終わった後に、得られた変換データがそれぞれのメモリ保存領域内に保存されて、そこにすでに保存されている、1 つ以上の以前の変換から得られたデータのうち少なくともいくつかを上書きすることを特徴とする、請求項 3 または 4 に記載の方法。

30

【請求項 6】

第 1 および第 2 のデータグループに対して複数の変換を行う方法であって、順次、

(a) 第 1 の変換エンジンを使って前記第 1 のデータグループに第 1 の変換を行って、第 1 の 1 回変換データグループを生成する工程、

(b) 第 1 のメモリ保存領域に前記第 1 の 1 回変換データグループを保存する工程、

(c) 前記第 1 のメモリ保存領域から前記第 1 の 1 度変換データグループを読み出す工程、

(d) それに対して第 2 の変換エンジンを使って第 2 段階の変換を行って、第 1 の 2 回変換データグループを生成する工程、および

40

(e) 前記第 1 の 2 回変換データグループを前記第 1 のメモリ保存領域に書き込んで前記第 1 の 1 回変換データグループを上書きする工程、

を包含し、

前記方法は、順次、

(f) 前記第 1 の変換エンジンを使って前記第 1 の変換を前記第 2 のデータグループに行うことで、第 2 の 1 回変換データグループを生成する工程、

(g) 前記第 2 の 1 回変換データグループを第 2 のメモリ保存領域に保存する工程、

(h) 前記第 2 の 1 回変換データグループを前記第 2 のメモリ保存領域から読み出す工程、

50

(i) それに対し前記第2の変換エンジンを使って前記第2段階の変換を行い、第2の2回変換データグループを生成する工程、および

(j) 前記第2の2回変換データグループを前記第2のメモリ保存領域に書き込むことで前記第2の1回変換データグループを上書きする工程、
をさらに包含し、

工程(f)は工程(a)が完了した後であって、且つ、工程(e)の前に始まることを特徴とする方法。

【請求項7】

工程(f)は、工程(c)が完了する前に始まることを特徴とする、請求項6に記載の方法。

【請求項8】

工程(a)から工程(j)はデータグループの多様性を使って繰り返され、工程(a)から(e)は奇数番号のデータグループに適用され、工程(f)から(j)は偶数番号のデータグループに適用されることを特徴とする、請求項6または7に記載の方法。

【請求項9】

工程(e)の後に、順次、

(e1) 前記第1のメモリ保存領域から前記第1の2回変換データグループのサブグループを読み出す工程、

(e2) それに対し前記第2の変換エンジンを使って第3段階の変換を行うことで、第1の3回変換データサブグループを生成する工程、

(e3) 前記第1の3回変換データサブグループを前記第1のメモリ保存領域に書き込むことで、前記第1の2回変換データグループの前記サブグループを上書きする工程、
をさらに包含し、そして工程(j)の後に、

(j1) 前記第2の2回変換データグループのサブグループを前記第2のメモリ保存領域から読み出す工程、

(j2) それに対し前記第2の変換エンジンを使って第3段階の変換を行うことで、第2の3回変換データサブグループを生成する工程、および

(j3) 前記第2の3回変換データサブグループを前記第2のメモリ保存領域に書き込むことで、前記第2の2回変換データグループの前記サブグループを上書きする工程、
をさらに包含することを特徴とする、請求項6から8のいずれかに記載の方法。

【請求項10】

工程(c)から工程(e3)の組み合わせを行うのにかかる時間は、工程(a)を行うのにかかる時間に等しいまたはそれより短く、

工程(h)から(j3)の組み合わせを行うのにかかる時間は、工程(f)を行うのにかかる時間に等しいまたはそれより短いことを特徴とする、請求項9に記載の方法。

【請求項11】

工程(e3)の後に、順次、

(e4) 前記第1の3回変換データサブグループを前記第1のメモリ保存領域から読み出す工程、

(e5) それに対して前記第2の変換エンジンを使って第4段階の変換を行うことで、第1の4回変換データサブグループを生成する工程、

(e6) 前記第1の4回変換データサブグループを前記第1のメモリ保存領域に書き込むことで、前記第1の3回変換データサブグループを上書きする工程、
をさらに包含し、

工程(j3)の後に、

(j4) 前記第2の2回変換データグループを前記第2のメモリ保存領域から読み出す工程、

(j5) それに対して前記第2の変換エンジンを使って第4段階の変換を行うことで、第2の4回変換データサブグループを生成する工程、および

(j6) 前記第2の4回変換データサブグループを前記第2のメモリ保存領域に書き込

10

20

30

40

50

むことで前記第 2 の 3 回変換データサブグループを上書きする工程、
をさらに包含することを特徴とする、請求項 9 または 10 に記載の方法。

【請求項 12】

工程 (c) から工程 (e 6) の組み合わせを行うのにかかる時間は工程 (a) を行うのにかかる時間と等しいかまたはより短く、

工程 (h) から工程 (j 6) の組み合わせを行うのにかかる時間は工程 (f) を行うのにかかる時間と等しいかまたはより短いことを特徴とする、請求項 11 に記載の方法。

【請求項 13】

工程 (e 3) から工程 (e 6) および工程 (j 3) から工程 (j 6) を複数回、それぞれのメモリ保存領域に保存された変換サブグループを使って繰り返す工程をさらに包含し、

それぞれの偶数番号の変換はメモリに保存されているデータのサブグループだけに対して行われ、それぞれの奇数番号の変換は先行の偶数番号の変換工程で生成されたすべてのデータに対して行われることを特徴とする、請求項 11 または 12 に記載の方法。

【請求項 14】

データグループは 10 回の変換を受けることを特徴とする、請求項 13 に記載の方法。

【請求項 15】

第 2 回目又はそれより高位の変換のすべてをデータのそれぞれのグループに行うのにかかる時間は、第 1 目の変換を行うのにかかる時間よりも短いことを特徴とする、請求項 14 に記載の方法。

【請求項 16】

第 1 および第 2 のメモリ保存領域は、二重データ速度メモリの 2 つの保存領域をそれぞれ備えることを特徴とする、請求項 13 から 15 のいずれかに記載の方法。

【請求項 17】

複数の変換はマルチレベルウェーブレット変換を含むことを特徴とする、請求項 1 から 16 のいずれかに記載の方法。

【請求項 18】

マルチレベルウェーブレット変換はハールウェーブレット変換を含むことを特徴とする、請求項 17 に記載の方法。

【請求項 19】

前記マルチレベルウェーブレット変換は 2 / 10 ウェーブレット変換を含むことを特徴とする、請求項 17 に記載の方法。

【請求項 20】

それぞれのデータグループは、画像データのフレームを備えることを特徴とする、請求項 1 から 19 のいずれかに記載の方法。

【請求項 21】

請求項 1 から 5 のいずれかに記載の方法に従って変換された第 1 および第 2 のデータグループに対して複数の逆変換を行う方法であって、順次、

第 1 の逆変換エンジンを使って、前記第 1 の変換データグループの最後の一つを除くすべてについて逆変換を行う工程と、

第 1 の逆変換エンジンを使って、前記第 2 の変換データグループの最後の一つを除くすべてについて逆変換を行う期間と少なくとも部分的に重複する期間内に、第 2 の逆変換エンジンを使って、前記第 1 の変換データグループの最後の一つを逆変換する工程と、を包含することを特徴とする方法。

【請求項 22】

前記第 1 の変換データグループの最後の一つを除くすべてについて逆変換を行うのにかかる時間は、前記第 1 のデータグループに対して最後の逆変換を行うのにかかる時間と等しいかまたはより短いことを特徴とする、請求項 21 に記載の方法。

【発明の詳細な説明】

【技術分野】

【0001】

本発明はデータ処理の方法に関し、より詳細には、画像に関するデータの処理方法に関する。本発明は特に、但しそれに限定されるわけではないが、データ圧縮の方法に関する。

【背景技術】

【0002】

〔圧縮の問題〕

電子画像がすでにデジタル形式であると仮定すると、画像圧縮は、画像を表現するのに必要とされるデータビットの数を大幅に削減する手段である。典型的なパラメータは次の通りである。

【0003】

色パラメータ 3 (例えば、RGBまたはYUV)

色当たりのビット 8 (例えば、4、10、12)

水平画素 1400 (または、例えば、720、1024、1600、1920)

垂直画素 1050 (または、例えば、588、768、1200)

秒当たりのフレーム 60 (または、例えば、24、30、85)

従って、60Hzで稼動するSXGA+ (1400×1050) 画像の場合、非圧縮データ速度は例えば次の通りであり、これは最上位モデル製品の典型的な要件であり得る。

【0004】

$3 \times 8 \times 1400 \times 1050 \times 60 = 2,116,800,000$ ビット/秒

元々はアナログ時代に導入されたものであるが、実際には現在にも等しく適用できる帯域幅縮小の「典型的な」方法としては、(a) 代替の色空間を使うことによる色の縮小 (例えば、縮小された彩度情報)、および (b) フレームごとのデータ縮小 (例えば、それぞれが全フレーム解像度の半分である「インターレースされた」画像を使うことによって行うが、それでも動作を行うための高いフレーム速度が可能である) が挙げられる。

【0005】

従って、(いわゆる1920iフォーマットに基づく) 「高品位」画像データ速度は次のようになり得る。

【0006】

$16 \times 1920 \times 1024 \times 30 = 943,718,400$ ビット/秒

しかしながら、かかる構成は、せいぜいよくても部分的に問題点を緩和するにすぎない。明らかに、さらにもっと積極的な方法が必要である。映像からUXGAにわたる画像のための目標ビット速度は、動作量変動する状態で、0.5から100Mb/秒にわたるが、10Mb/秒未満の速度に重点を置く。

【0007】

〔圧縮の基礎〕

画像圧縮には、空間的圧縮と時間的圧縮の2つのタイプがある。空間的圧縮は、単一画像フレームを記述するのに必要とされる情報量を削減するものであり、時相的圧縮は、フレームごとに全フレームデータを送る必要を少なくしつつも圧縮されていない画像における動きは維持したままとするものである。

【0008】

空間圧縮を使って単一画像フレームを圧縮するための望ましい方法は次の通りである。

(a) 画像をより効率的または「簡便な」やり方で記述できる方法を見出す。例えば、広い領域が緑色で着色されている場合、ただ単にこの領域を限られた数の座標で定義し、これを画素ごとに記録するのではなく「緑」とコード化する。

(b) 必要に応じて、人間の視覚の公知の特徴を利用し、見る人物には見えないかもしれない画像の局面に関するデータを排除するかまたは少なくする、および、

(c) 得られた数値データを取りこれを、例えば冗長なゼロを抑制するまたはランレングス符号化のような標準的な損失なしのデータ圧縮技術によって、より効率的に記録する。

【0009】

10

20

30

40

50

時間的圧縮のための主な方法は、連続的な画像を比較し、1つの画像と他の画像との変化に対して送信される情報を制限することである。かかる方法を使用する場合、全フレーム画像またはその等価物を周期的に送って画像の復元が正しいデータから確実に起こるようにする方法がなければならない。

【0010】

〔圧縮システムの必要な属性〕

特定の用途向けの圧縮システムを開発する際、数多くの優先すべき事項を次のように特定できる。

(a) システムは最小限の待ち時間でリアルタイムに作動しなければならない。

(b) システムはさまざまな「ビット深さ」に適さなければならない。8ビット画素が典型的に使用されるが、特定の用途ではシステムを10または12ビット画素まで拡張するのが望ましいであろう。

(c) システムは空間的解像度の点でスケーラブルでなくてはならない。最も高い現行の解像度は 1600×1200 (UXGA) であるが、原則としてシステムはより高い解像度が導入された場合これらにも対応できるべきである。

(d) ハードウェアの点で、システムは「対称的」である必要がある、すなわち、符号器を実現する費用は復号器を実現する費用と大幅に異なるべきではない、(ただし用途によってはソフトウェアベースの復号器の場所もあることが認識される)。

(e) システムは標準的なコンポーネントを使って実現できなければならない(ただし大容量用途向けにはASICバージョンが考えられるであろう)。

(f) 高解像度画像データ全体を処理する必要なしに、高解像度画像の低解像度バージョンを抽出できるまたは高解像度画像の一部を抽出できなければならない。かかる特徴は非常に重要である。

【0011】

〔ウェーブレット変換の選択〕

実用的な空間的画像圧縮システムは、画像情報中の冗長性を簡単に識別でき排除できる方法を必要とする。理論的には元の画素数値データを分析することは可能であるが、現実にはこれは非効率的であり計算的に集約されてしまう。

【0012】

現状の手法は、元の画素データを他のフォーマットに「変換」することである。新しいフォーマット自身は画像を表現するのに必要なデータ量を少なくすることはしないが、これが行うのは、冗長な情報を簡単に識別し排除できるようにデータを表示するというものである。この新しいフォーマットはまた、効率的に符号化できるやり方でのデータ表示もする。

【0013】

変換の考え方は、フーリエ法則によって例示される。この法則は、調和して関連し合う振幅の変動する数多くの正弦波を一緒にすることによって、どのような複雑な波形でも再生できることを示している。使用する高調波の数が多いほど、得られる結果は元の波形に近くなる。従って、例えば、「矩形波」は、その振幅が等間隔でゼロから最大値まで変化しそして再び戻るものとして示される「時間」ドメインか、または元の波形の基本周波数のそれぞれの高調波に当てはまる1組の係数が規定される「周波数」ドメインのいずれかで記述できる。

【0014】

変換の概念は、矩形波の例によりほぼ近い形で図示される。高調波の振幅を周波数に対して描く場合、結果は $\text{Sin } x / x$ 関数である。

【0015】

図1は、このような「変換対」を示している。左側の波形が時間に対する振幅を表す場合、右側のものは周波数の分布を示しているが、これが逆であっても同じことが当てはまる。すなわち、左側のものが周波数の分布を示す場合、右側のものはその結果得られる、時間に対する振幅である。

10

20

30

40

50

【0016】

図示される変換対の特徴は、左側の関数が狭くなるほど右側の関数が広くなるということである。このことは、もし狭い範囲の周波数だけがある場合、得られる振幅分布は「広く平坦になる」であろうと言うのに等しい。周波数分布がゼロになる場合の限界点では、結果は長さが無限大の平坦な線、すなわち「直流」になる。

【0017】

この変換対の例について興味深い点は、これがどのように時間／周波数変換を作成することができるかについての手がかりを提供することである。左側の関数がフィルタの帯域幅を表す場合は、右側の関数はフィルタのインパルス応答を表す。

【0018】

画像圧縮は、実際に、まさに周波数ドメインの考え方をを使うものである。通常の構成では、画像を複数のブロックに分割し、ブロックはそれぞれが画素のアレーからなる。そして、周波数分布についてそれぞれのブロックを検討する。画像の「縁」では「高い」周波数が生じるが、均一な「輝度」の領域は「低い」周波数を示す。

【0019】

画像圧縮のためのもっとも公知の変換は離散コサイン変換（DCT）であり、これはフーリエ変換の空間版である。その原理は、広い範囲の周波数に対して画像データを「試験」し、それぞれについて係数を生成するというものである。このプロセスは、それ自体が原則的に無限の正弦波（ただし実際には必ず上が切り取られた状態になる）である基底関数の使用を必要とする。DCTの特徴は周波数ドメインを等しい増分に分割する点であり、このことの当然の帰結として、基底関数は周波数に従った異なるサイクル数を含む。

【0020】

DCTに代わるものとしてウェーブレット変換が普及しており、これは入力帯域幅を半分に分割する、カスケード状に配列された一連の補完的な高いおよび低いパスフィルタを使って実行されるものである。それぞれのフィルタの出力は図2に図示されるように2分の1にダウンサンプルされるため、カスケードの出力データは入力データと同じ大きさである。ハイパスフィルタのインパルス応答は「ウェーブレット」である。

【0021】

ウェーブレットの特徴は、この文脈では、ウェーブレット基底関数が、周波数に関わりなくすべて同じサイクル数を含んでいることであり、このことはこれらが異なる長さであることを意味する。図示されるカスケード構成では、ウェーブレットのセットは、それぞれのステージで2分の1に変倍された1つの単一のウェーブレットから導き出されている。

【0022】

カスケードの終わりには、嚴重に帯域が制限された信号がある。前回の周波数帯域から係数を追加すると利用できる解像度が2倍になり、処理が繰り返されると、解像度が再び2倍になる。このことはウェーブレット変換の3つの属性を表している、すなわち、

(a) 当然ながら変倍可能である。

(b) 周波数ドメインが等しい増分ではなくオクターブに分割される。

(c) 画像処理の状況では、利用可能なデータの一部だけを使うことによって画像の低解像度版を得ることが可能である。

【0023】

簡単な例を挙げると、もし16個の入力サンプルが4つのステージのカスケードに供給されると、最初のステージでは差分の8つのサンプルを作成することになる。次のステージでは4つの、その次の2つの、そして最終的には差分の1つのサンプルを、単一の値とともに生成することになる。この単一の値はローパスフィルタのシーケンスから由来するもので、16個すべてのサンプルの平均的な「輝度」とみなすことができる（直流成分）。出力サンプルの総数は入力総数と同じである、すなわち、

$$(8 + 4 + 2 + 1) \text{ 個の差分} + (1) \text{ 個の平均値} = 16$$

画像の高周波数成分は数多くの短いウェーブレットによって記述され、低周波数部分は

10

20

30

40

50

数少ない長いウェーブレットによって記述される。

【0024】

ウェーブレット変換は、時期が合う一連の離散的信号とみなすことができ、これらの離散的信号はそれぞれが画像の複数の解像度分析を提供するものである。変換の目的は、画像をより重要性の大きな係数またはより重要性の低い係数に分解することである。そして、あまり重要でない係数は量子化または排除できる。ウェーブレット変換は（他の変換と比べると）有意の係数をもっともよい形でコンパクト化するものである。

【0025】

電子画像は「1次元」ではなく、2次元画素アレーからなる。従って、画像圧縮では、変換プロセスを2次元で行う必要がある。単一段ウェーブレット分解のプロセスを図3に示す。元の画像はろ過されて4つの周波数帯となる、すなわち、LLは、水平および垂直方向の両方にローパスフィルタ化されサブサンプル化された元の画像である。HLは、残りの垂直周波数、すなわち、元の画像とLL画像との間の差の垂直成分からなる。同様に、LHは残りの水平周波数からなり、垂直および水平両方のフィルタリングの高周波数成分からなるHHは、残りの斜め周波数を表している。

【0026】

実際には、多段分解が生じる。LLは解像度が低くされた全体の画像（または画像の一部）を表しているため、ここでLL画像に対しフィルタリングプロセスを行って第2レベルの分解を行う。損失なしの変換（すなわち、情報内容が失われることのない変換）を行うためには、個々の画素の空間的な等価物までプロセスを繰り返す必要がある。

【0027】

従って、例えば、プロセスを4×4画素ブロックに適用する場合、「レベル1」変換を想定できる。この場合、水平および垂直方向の両方にフィルタ対を適用することによって4つの係数が導かれる。そして、「レベル2」変換でローパスフィルタの出力を表している情報の4分の1に対し同じプロセスを行う、これは空間的な意味では画素レベルである。（ブロックがより大きかった場合は、損失なしの変換を行うのにより大きな「レベル」が必要になると思われる。）

【0028】

変換の復号化（「再構築」）は符号化（「分解」または「破壊」）プロセスの逆であり、どのような現実の実行でも高度に対称的であることを指す。簡素なフィルタ対レベルでは、もし2つの入力の流れが2倍にアップサンプル化され、その後、ろ過されそして再度組み合わせられると、得られる結果は元の空間的なデータである。完全な再構築が起こるためには、復号化フィルタは符号化フィルタの応答と正確に一致しなければならない。「レベル」数が同じでなければならない。

【0029】

好ましい圧縮法の基礎としてウェーブレット変換が選択された。なぜなら、

(a) 固有のスケーラビリティを持つ、

(b) 有意な変換係数をもっともよい形でコンパクト化する、

(c) イメージデータ全体を処理せずして画像の低解像度バージョンを簡単に導き出せる、

(d) 高速平行処理に変更可能である、

(e) 効率的な符号化に変更可能である、そして

(f) （変換へのまたは変換からの）符号化および復号化プロセスが対称的であるからである。

【0030】

ウェーブレット変換に基づいて圧縮システムを実現する際、システムが標準的な機材を使って実現するよう実行可能でありまた市場のニーズを満たすことを確実にするための、多数の重要な実行上の点を考慮に入れるべきである。

【0031】

設計に多大な影響を及ぼすいくつかの特定の点としては次のものが挙げられる、

10

20

30

40

50

(a) システムは損失なしの（典型的な実現される割合は2：1）から視覚的に損失なしの（おそらく30：1と高い）にわたり損失ありの（50：1以上）までに及ぶ広い範囲の空間的圧縮比に対応できなければならない。

(b) 符号化および復号化プロセスの作業は、画像の複雑さに関わりなくこれらが定義された時間サイクル内で作動しなければならない意味で決定的でなければならない。明らかに、これらは「リアルタイム」に作動しなければならない。

(c) フルモーション「ビデオ」画像および高解像度「グラフィック」画像の異なるニーズを全面的に考慮しなければならない。

【0032】

システム全体の説明はセクションに分割でき、次のように要約される。

10

- (a) 入力システム、色空間の選択、
- (b) ウェーブレット変換エンジン、
- (c) 得られたデータの符号化
- (d) 一時的圧縮のための符号化
- (e) ネットワーク接続、および
- (f) 復号化

画像入力からデータストリーム出力までの異なるステージを図5に図示する。

【0033】

実際のシステムは、実行可能である限り、既存の標準に基づいていなければならない。従って、好ましいシステムへの「入力」は現在のデジタル画像標準、主にコンピュータグラフィック画像のためのDVI標準、およびビデオおよび高品位ビデオ画像のためのSDIおよびHDSDI標準に基づいている。DVIそれ自体は事実上1600×1200（画素当たり24ビット）の画像解像度に制限されているが、複数のDVI信号をいくつかの組に編成してより大きな画像を記述することが可能である。明らかに、いずれの実際のシステムもより高い解像度および新しいトランスポート基準に、それらが利用可能となると同時に適合するように設計されていなければならない。

20

【0034】

電子画像は通常、色パラメータRGB（赤、緑、青）によって記述される。従って、原則的にはいずれの圧縮システムも3部構成で作動しなければならない。すなわち、色パラメータそれぞれ対して1つの「チャンネル」である。これは、一般にYUVと呼ばれる他の色空間に使うのに都合がよいもので、この場合、Yは「輝度」または「白色明るさ」値であり、UおよびVは一括して「クロミナンス」または「色差」値と呼ばれる2つの「色差」値である。RGBからYUVへの基本的な変換ではデータ量が減ることはないが、実際には、人間の目は輝度空間的解像度に比べるとクロミナンス空間的解像度に対する感度が鈍い。従って、この事実がカラーテレビ帯域軽減の手段としてその発足当初から使用されてきた。

30

【0035】

YUVの使用に限られるわけではないが、好ましいシステムはこれに基づいている、なぜならこれを使用することでクロミナンスおよび輝度情報について微分符号化速度が可能となり、その結果、人間の目の応答性を利用して圧縮効率が改善されるからである。RGBとYUVとの間の変換は簡単な演算のように見受けられるが、画像の劣化かデータ量の増加のいずれかを招く可能性のある落とし穴がある。

40

【0036】

CCIR601標準は以下のマトリックスによって成分ビデオを定義している。

【0037】

【数 1】

$$\begin{bmatrix} Y \\ C_R \\ C_B \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.500 & -0.419 & -0.081 \\ -0.169 & -0.331 & 0.500 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

【0038】

10

このマトリックスは、それ自体は損失なしの可逆転換には適さない、なぜなら転換要因として非整数を使用するからである。従って、好ましいシステムは以下の数式を使用する。この数式はC C I Rマトリックスの近似を表しており、損失なしの可逆転換を実現するものである。ここで、Y、U、およびVは可逆輝度およびクロミナンス値である。

【0039】

【数 2】

$$\begin{aligned} U_r &= R - G & G &= Y_r - \lfloor (U_r + V_r)/4 \rfloor \\ V_r &= B - G & R &= U_r + G \\ Y_r &= G + \lfloor (U_r + V_r)/4 \rfloor & B &= V_r + G \end{aligned}$$

20

【0040】

上述の数式で、記号exuは「床」関数と呼ばれるもので、x以下の最大整数として定義されるものである。上述の数式は以下の属性を持つ。

(a) 入力がNビットのRGBであれば、損失なしの変換もまたNビットを持つY_rとなるが、U_rおよびV_r成分はN+1ビットを持つ。

(b) その後、成分が逆転されると、結果はNビットのRGB信号である。

30

【0041】

本発明の背後には開発の一部として2つの問題が生じる。すなわち、

(a) ウェーブレット変換に適用した場合、クロミナンス成分における余分なビットの生成を損失なしの性能を失うことなく吸収することは可能であるか。

(b) 損失ありの圧縮が必要な場合、数式を改変して性能を最適化すべきか。

【0042】

1つの重要な発見は、損失なしの動作を行うための色変換およびウェーブレット変換ビット成長は両方とも、組み合わせられた結果を精度保存の特性(P P P)と呼ばれる特性に適用することによって省けるであろうということであった。この技術のさらなる詳細は、非特許文献1に見出すことができる。しかしながら、上述の数式およびP P P技術は両方とも、損失なしの変換だけに適用されるものである。

40

【0043】

損失ありの変換が必要な場合は、他の技術を使用する。ここで、その目的はただ単に元の範囲を保存し確実にビット成長がないようにするだけである。これは以下の数式を使って実現される。

【0044】

【数 3】

$$\begin{aligned} U_r &= \lfloor (R - G)/2 \rfloor & G &= Y_r - \lfloor (U_r + V_r)/2 \rfloor \\ V_r &= \lfloor (B - G)/2 \rfloor & R &= 2U_r + G \\ Y_r &= G + \lfloor (U + V)/2 \rfloor & B &= 2V_r + G \end{aligned}$$

【先行技術文献】

【非特許文献】

10

【0045】

【非特許文献1】1997年12月Hongyang Chao, Paul Fisher, Zeyi Hua著「An Approach to the Integer Wavelet Transformations for Lossless Image Compression (損失なしの画像圧縮のための整数ウェーブレット変換の方法)」

【発明の概要】

【発明が解決しようとする課題】

【0046】

データの有効な符号化を可能とし、損失なし、または損失ありのいずれかである得る空間的および時間的両方の画像データ圧縮の方法を提供することにある。

【課題を解決するための手段】

20

【0047】

本発明の一の側面によると、第1および第2のデータグループのそれぞれに対して第1の変換を行うことで第1および第2の変換データグループをそれぞれ生成し、前記第1および第2の変換データグループのそれぞれに対して複数のその後の変換を行う方法が提供される。この方法は、順次、第1の変換エンジンを使って前記第1のデータグループに前記第1の変換を行う工程、および前記第1の変換エンジンが前記第2のデータグループに対して前記第1の変換を行う期間と少なくとも部分的に重複する期間内に、第2の変換エンジンを使って前記変換された第1のデータグループに前記後続の変換をすべて行う工程を包含する。

【0048】

30

これにより、マルチレベル変換を行うための特に効率的な方法が提供される、なぜなら変換エンジンがたった2つだけしか必要ないからである。

【0049】

変換されたデータグループに対して前記後続の変換をすべて行うのにかかる時間は、第1の変換をそれに対して行うのにかかる時間と等しいまたはより短い。これにより、その後の変換工程は全体のマルチレベル変換方法においてどのような遅延も引き起こさないという利点が提供される。

【0050】

本方法は、好ましくは、第1のメモリ保存領域に前記第1のデータグループに対するそれぞれの変換から得られた変換データを保存する工程、および第2のメモリ保存領域に前記第2のデータグループに対するそれぞれの変換から得られた変換データを保存する工程をさらに包含する。そして、さらに別の複数のデータグループに適用されると、前記またはさらに別の奇数番号のデータグループそれぞれに対するそれぞれの変換から得られた変換データが前記第1のメモリ保存領域に保存され、前記またはさらに別の偶数番号のデータグループそれぞれに対するそれぞれの変換から得られた変換データが前記第2のメモリ保存領域に保存される。このようなアレンジメントは、たとえマルチレベル変換を実行する場合であってもメモリ保存領域を2つだけしか必要としない。

40

【0051】

前記後続の変換のそれぞれが終わった後、好ましくは、得られた変換データがそれぞれのメモリ保存領域内に保存されて、そこにすでに保存されている、1つ以上の以前の変換

50

から得られたデータのうち少なくともいくつかを上書きする。

【0052】

本発明の他の側面によると、第1および第2のデータグループに対して複数の変換を行う方法が提供される。この方法は、順次、

(a) 第1の変換エンジンを使って前記第1のデータグループに第1の変換を行って、第1の1回変換データグループを生成する工程、

(b) 第1のメモリ保存領域に前記第1の1回変換データグループを保存する工程、

(c) 前記第1のメモリ保存領域から前記第1の1度変換データグループを読み出す工程

、
(d) それに対して第2の変換エンジンを使って第2段階の変換を行って、第1の2回変換データグループを生成する工程、および

(e) 前記第1の2回変換データグループを前記第1のメモリ保存領域に書き込んで前記第1の1回変換データグループを上書きする工程、

を包含し、

前記方法は、順次、

(f) 前記第1の変換エンジンを使って前記第1の変換を前記第2のデータグループに行うことで、第2の1回変換データグループを生成する工程、

(g) 前記第2の1回変換データグループを第2のメモリ保存領域に保存する工程、

(h) 前記第2の1回変換データグループを前記第2のメモリ保存領域から読み出す工程

、
(i) それに対し前記第2の変換エンジンを使って前記第2段階の変換を行い、第2の2回変換データグループを生成する工程、および

(j) 前記第2の2回変換データグループを前記第2のメモリ保存領域に書き込むことで前記第2の1回変換データグループを上書きする工程、

をさらに包含し、

工程(f)は工程(a)が完了した後であって、且つ、工程(e)の前に始まる。

【0053】

工程(f)は、好ましくは工程(c)が完了する前に始まる。

【0054】

工程(a)から工程(j)はデータグループの多様性を使って繰り返してよい。この場合、工程(a)から(e)は奇数番号のデータグループに適用され、工程(f)から(j)は偶数番号のデータグループに適用される。

【0055】

本方法は、好ましくは、工程(e)の後に、順次、

(e1) 前記第1のメモリ保存領域から前記第1の2回変換データグループのサブグループを読み出す工程、

(e2) それに対し前記第2の変換エンジンを使って第3段階の変換を行うことで、第1の3回変換データサブグループを生成する工程、

(e3) 前記第1の3回変換データサブグループを前記第1のメモリ保存領域に書き込むことで、前記第1の2回変換データグループの前記サブグループを上書きする工程、

をさらに包含し、そして工程(j)の後に、

(j1) 前記第2の2回変換データグループのサブグループを前記第2のメモリ保存領域から読み出す工程、

(j2) それに対し前記第2の変換エンジンを使って第3段階の変換を行うことで、第2の3回変換データサブグループを生成する工程、および

(j3) 前記第2の3回変換データサブグループを前記第2のメモリ保存領域に書き込むことで、前記第2の2回変換データグループの前記サブグループを上書きする工程、

をさらに包含する。

【0056】

工程(c)から工程(e3)の組み合わせを行うのにかかる時間は好ましくは工程(a

10

20

30

40

50

）を行うのにかかる時間に等しいまたはそれより短く、工程（h）から（j 3）の組み合わせを行うのにかかる時間は、工程（f）を行うのにかかる時間に等しいまたはそれより短い。

【0057】

本方法は、好ましくは、工程（e 3）の後に、順次、
（e 4）前記第 1 の 3 回変換データサブグループを前記第 1 のメモリ保存領域から読み出す工程、

（e 5）それに対して前記第 2 の変換エンジンを使って第 4 段階の変換を行うことで、第 1 の 4 回変換データサブグループを生成する工程、

（e 6）前記第 1 の 4 回変換データサブグループを前記第 1 のメモリ保存領域に書き込むことで、前記第 1 の 3 回変換データサブグループを上書きする工程、
をさらに包含し、

工程（j 3）の後に、

（j 4）前記第 2 の 2 回変換データグループを前記第 2 のメモリ保存領域から読み出す工程、

（j 5）それに対して前記第 2 の変換エンジンを使って第 4 段階の変換を行うことで、第 2 の 4 回変換データサブグループを生成する工程、および

（j 6）前記第 2 の 4 回変換データサブグループを前記第 2 のメモリ保存領域に書き込むことで前記第 2 の 3 回変換データサブグループを上書きする工程、

をさらに包含する。

【0058】

工程（c）から工程（e 6）の組み合わせを行うのにかかる時間は好ましくは工程（a）を行うのにかかる時間と等しいかまたはより短く、工程（h）から工程（j 6）の組み合わせを行うのにかかる時間は工程（f）を行うのにかかる時間と等しいかまたはより短い。

【0059】

本方法は、好ましくは、工程（e 3）から工程（e 6）および工程（j 3）から工程（j 6）を複数回、それぞれのメモリ保存領域に保存された変換サブグループを使って繰り返す工程をさらに包含し、それぞれの偶数番号の変換はメモリに保存されているデータのサブグループだけに対して行われ、それぞれの奇数番号の変換は先行の偶数番号の変換工程で生成されたすべてのデータに対して行われる。

【0060】

データグループは 10 個の変換を受け得るが、これはレベルのそれぞれが 2 つの別々の変換工程を必要とする 5 レベルウェーブレット変換を行う場合のことである。

【0061】

複数の変換は、ハールウェーブレット変換または 2 / 10 ウェーブレット変換のようなマルチレベルウェーブレット変換を一括して含み得る。

【0062】

それぞれのデータグループは画像データのフレームを含み得る。

【0063】

本発明のさらなる側面によると、上で定義した方法に従って変換された第 1 および第 2 のデータグループに対して複数の逆変換を行う方法であって、順次、第 1 の逆変換エンジンを使って、前記第 1 の変換データグループの最後の一つを除くすべてについて逆変換を行う工程と、第 1 の逆変換エンジンを使って、前記第 2 の変換データグループの最後の一つを除くすべてについて逆変換を行う期間と少なくとも部分的に重複する期間内に、第 2 の逆変換エンジンを使って、前記第 1 の変換データグループの最後の一つを逆変換する工程と、を包含する。

【0064】

前記第 1 の変換データグループの最後の一つを除くすべてについて逆変換を行うのにかかる時間は、前記第 1 のデータグループに対して最後の逆変換を行うのにかかる時間と等

10

20

30

40

50

しいかまたはより短い。

【発明の効果】

【0065】

本発明によれば、変換エンジンを2つだけしか必要としないので、マルチレベル変換を行うための特に効率的な方法が提供される。

【図面の簡単な説明】

【0066】

【図1】「変換対」の一例を図示している。

【図2】カスケードになった1セットのフィルタを使ってどのようにウェーブレット変換が実行されるかを図示している。

【図3】単一段のウェーブレット分解を示している。

【図4】復号器における単一段のウェーブレット再構築を示している。

【図5】好ましい符号器内のプロセスを図示している。

【図6】5レベル分解を図示している。

【図7】ブロックベースのハール変換を図示している。

【図8】変換の動作を図示するための画素アレーの一例を示している。

【図9】数式1および2を図8のアレーに適用した結果を図示しており、これら数式は $r = 0 \cdots 9$ および $c = 0 \cdots 4$ の範囲で解かれている。

【図10】数式5、6、9および10を図9示す水平変換データに適用した結果を図示しており、範囲は $r = 0 \cdots 4$ および $c = 0 \cdots 4$ であり、例示的な画素アレーのためのハール変換を表している。

【図11】2/10変換数式をサンプル画素アレーに適用した結果を図示している。

【図12】符号化システム全体のアーキテクチャを示すブロック図である。

【図13】好ましい変換エンジンを詳細に図示している。

【図14】メモリにおける連続的な書き換えを図示している。

【図15】どのようにして係数データが分析されるかを示すCKLツリーを図示している。

【図16】64LタイプからなるLツリーを図示しており、24が図上の「Lボックス」として示されており、残りの40はレベル3LHおよびHHLタイプの子供として暗示されている。

【図17】8つすべてのデータ面を複合物として検討することにより符号化損失を省く概念を図示している。

【図18】5レベル変換に適用される重み付け要因の概念を図示しており、典型的には $a = 2$ である。

【図19】どのようにして1ビット面の係数データの組織がウェーブレット変換に直接関連するかを図示している。

【図20】好ましいL符号器を図示している。

【図21】好ましいCS符号器を図示している。

【図22】LKCSパスを図示している。

【図23】好ましい一時的符号化プロセスの図式化である。

【図24】標準的なイーサネット（登録商標）で使用されるIEEE802.3フレームフォーマットを図示している。

【図25】圧縮された画像ストリームのIPパケットへの変換を図示している。

【図26】ビットストリームの一部を示しており、2つの連続的な画像ブロックを、互いに同期言語、および画像内のブロック位置を特定する指標番号が関連し合った状態で示している。

【発明を実施するための形態】

【0067】

ここで本発明の好ましい実施態様を、添付の図面を参照しながら詳細に説明する。

【0068】

〔ウェーブレット変換エンジン〕

好ましい設計の大きな利点は、画像の破壊および再構築に同じ「エンジン」を使用することができる点である。この設計は、リアルタイムに2次元で5レベル変換を実行する単一フィルタバンクからなるアーキテクチャを使用する。

【0069】

上述から、5レベル変換を使用することにより 32×32 画素ブロックを記述するデータが得られることがわかる。しかしながら、もしこれが文字通り符号化段階における場合であったなら、最終的な結果は「濃淡のむらのある(blocky)」画像（特に高い圧縮割合では）となるであろう。ブロックの端にある画素に関連するデータが隣接するブロックの画素のエネルギーを完全に考慮することを確実にするには、変換プロセスは画素アレー全体を「掃引」しなければならない。従って、得られるデータは、実際には一連の 32×32 ブロックを表すものとしてフォーマットされるが、このようにして得られた画像情報それ自体はブロックベースではない。

【0070】

図6は、5レベル変換プロセスの目的を示している。レベル3には16個の係数があり、レベル4には4つの係数があり、レベル5には1つの係数がある。ウェーブレット変換が信号をさまざまなサブ帯域に分解できる方法はいくつかある。かかる方法としては、均一な分解、オクターブ帯域分解、および適応またはウェーブレットパケット分解が挙げられる。これらのうち、オクターブ帯域分解がもっとも広く使用されている。これは不均一帯域分裂法であり、より周波数の低い部分をより狭い帯域に分解し、それぞれのレベルのハイパス出力にはさらなる分解を全くせずしてそのまま残すものである。

【0071】

システムをさまざまな供給源材料に最適なものにするために、好ましいシステムは2つの異なるウェーブレット変換を使用するようセットアップされる。ハール変換は鮮明な切れ目または「エッジ」の定義が正確である必要がある場合に材料に使用され、「ツェー／テン」またはTT変換は、「円滑な」結果がより喜ばしい場合にビデオ画像を移動するための別の方法として提供されるものである。

【0072】

ハール変換は、鮮明な切れ目（実際には細い線など）の一体性を維持することが非常に重要なグラフィック画像の圧縮に最良である。移動しているビデオ画像をとまなう場合、異なる変換を使用すると恩恵があり、好ましいシステムは、使用されている画像のタイプに従ってハール変換をとるか「ツェー／テン」（またはTT、または $2/10$ ）変換をとるかの選択が可能である。

【0073】

過酷な圧縮を行うと、画像を再構築した場合、画像人工物がブロック境界に現れる傾向がある。 $2/10$ 変換はより多くの画素をハイパスフィルタで処理するがこれには画像を「円滑化する」効果があり、ビデオ内容において視覚的により許容される結果を与える。

【0074】

ブロックベースのハール変換では、画像は 32×32 画素ブロックで処理され、1つのブロックがY、UおよびVそれぞれのためのものである。このことを図7に図面によって示す。実際には、画素はそれぞれが2画素 $\times$ 2画素の 16×16 重複なしブロックとして処理される。実際の処理、およびそれが $2/10$ 変換に必要とされる処理に類似する点を以下に説明する。

【0075】

両方の変換で、2段処理を使用する。第1段では、1次元変換で画素データから2つの係数LおよびHが導き出され、そして第2段では、2次元変換によりLL、LH、HLおよびHH値が導き出される。実際のところ、当初のローパスフィルタリングのための数式は両方の変換について同じである。ハイパスフィルタリングも似通ってはいるが、 $2/10$ 変換の場合、既存の導き出されたローパス値を観察した結果導き出されるさらに別の「予測」値がある。このことは得られる画像を「平滑化」する効果がある。

【 0 0 7 6 】

以下の数式では、Pを使ってもとの画素データを表している。接尾辞 r、c は縦および横の座標をそれぞれ表しており、p は予測子を示している。

【 0 0 7 7 】

数式 1. ハールおよび 2 / 1 0 変換両方のための L の誘導

【 0 0 7 8 】

【数 4】

$$L_{r,c} := \text{floor} \left[\frac{P_{(r,2 \cdot c)} + P_{r,(2 \cdot c)+1}}{2} \right] \quad 10$$

【 0 0 7 9 】

数式 2. ハール変換における H の誘導

【 0 0 8 0 】

【数 5】

$$H_{r,c} := P_{(r,2 \cdot c)} - P_{r,2 \cdot c+1} \quad 20$$

【 0 0 8 1 】

数式 3. 2 / 1 0 変換における予測子 p H の誘導

【 0 0 8 2 】

【数 6】

$$pH_{(r,c)} := \text{floor} \left[\frac{3 \cdot L_{(r,c-2)} - 22 \cdot L_{(r,c-1)} + 22 \cdot L_{(r,c+1)} - 3 \cdot L_{(r,c+2)} + 32}{64} \right] \quad 30$$

【 0 0 8 3 】

数式 4. 2 / 1 0 変換における H の誘導

【 0 0 8 4 】

【数 7】

$$H_{r,c} := P_{(r,2 \cdot c)} - P_{r,(2c)+1} + pH_{(r,c)} \quad 40$$

【 0 0 8 5 】

数式 5. ハール変換および 2 / 1 0 変換の両方のための L L の誘導

【 0 0 8 6 】

【数 8】

$$LL_{(r,c)} := \text{floor} \left[\frac{L_{(2 \cdot r, c)} + L_{(2 \cdot r + 1, c)}}{2} \right]$$

【0087】

数式 6. ハール変換における LH の誘導

【0088】

【数 9】

10

$$LH_{r,c} := \left[L_{(2 \cdot r, c)} - L_{[(2 \cdot r) + 1, c]} \right]$$

【0089】

数式 7. $2/10$ 変換における予測子 p LH の誘導

【0090】

【数 10】

20

$$pLH_{(r,c)} := \text{floor} \left[\frac{3 \cdot LL_{(r-2,c)} - 22 \cdot LL_{(r-1,c)} + 22 \cdot LL_{(r+1,c)} - 3 \cdot LL_{(r+2,c)} + 32}{64} \right]$$

【0091】

数式 8. $2/10$ 変換における LH の誘導

【0092】

【数 11】

30

$$LH_{r,c} := L_{(2 \cdot r, c)} - L_{[(2 \cdot r) + 1, c]} + pLH_{(r,c)}$$

【0093】

数式 9. ハール変換および $2/10$ 変換の両方のための HL の誘導

【0094】

【数 12】

$$HL_{(r,c)} := \text{floor} \left[\frac{H_{(2 \cdot r, c)} + H_{(2 \cdot r + 1, c)}}{2} \right]$$

40

【0095】

数式 10. ハール変換における HH の誘導

【0096】

【数 1 3】

$$HH_{r,c} := [H_{(2 \cdot r, c)} - H_{[(2 \cdot r) + 1, c]}]$$

【0 0 9 7】

数式 1 1. 2 / 1 0 変換における予測子 p H H の誘導

【0 0 9 8】

【数 1 4】

10

$$pHH_{(r,c)} := \text{floor} \left[\frac{3 \cdot HL_{(r-2,c)} - 22 \cdot HL_{(r-1,c)} + 22 \cdot HL_{(r+1,c)} - 3 \cdot HL_{(r+2,c)} + 32}{64} \right]$$

【0 0 9 9】

数式 1 2. 2 / 1 0 変換における H H の誘導

【0 1 0 0】

【数 1 5】

20

$$HH_{r,c} := H_{(2 \cdot r, c)} - H_{[(2 \cdot r) + 1, c]} + pHH_{(r,c)}$$

【0 1 0 1】

〔稼働中の変換数式の例〕

上述の数式の動作は例によってもっともよく理解される。図 8 は、P についていくつかの任意の値を持つ 1 0 × 1 0 画素の画素アレーを示している。レイアウトは純粋に 2 / 1 0 変換およびハール変換のはたらきに注意を引くための一例である。1 0 × 1 0 アレーは 2 / 1 0 を例示するために使用できる最小限度のものでありその他の意味合いはない。

30

【0 1 0 2】

数式 1 および 2 を図 8 のアレーに適用した場合、結果は図 9 に示すとおりである。ここで、留意すべき点は、

(a) 変換プロセスによりコラム数が半減する(数式は $r = 0 \dots 9$ および $c = 0 \dots 4$ について解かれる)。

(b) とはいえ、画像データ全体の量は同じままである(これは 2 セットの係数 L および H を持つためである)。

【0 1 0 3】

図 9 の結果は第 1 パスの「1 次元」水平変換を表している。図 9 の結果を数式 5、6、9、および 1 0 に適用すると、第 2 の「2 次元」垂直変換が完了する。結果全体は完全なハール変換であり、図 1 0 のような形態である。ここで、どのようにして縦および横データが半分になっているか、しかしふたたびここでもデータ全体の量は同じままということに留意するべきである。

40

【0 1 0 4】

ハール変換がアレー全体に適用されることがわかるが、2 / 1 0 変換を使う状況は全く違ったものである。ハール変換は 2 × 2 画素ブロックで作用するが、2 / 1 0 はより数多くのデータ画素を必要とする。そして、実際のところ、例の場合、アレーの 4 つの中央の画素についてしか有効な結果を提供できない(すなわち、図 8 の値が 8 0、4 8、4 5、1 1 0 の画素)。

【0 1 0 5】

50

$c = 4 \dots 5$ および $r = 2$ の範囲の数式 1、3、および 4 を適用すると、L および H について $2/10$ 値が得られる。その後、数式 5、7、8、9、11、および 12 を解くと、LL、LH、HL および HH について $2/10$ 値が導き出される。

【0106】

図 11 は、実施例の場合のこれらの解答を示している。左側には L および H についての解答を示し、右側には LL、LH、HL および HH についての解答を示している。図 11 の右側の情報が、元の画素データを確実に回復可能とするのに転送されねばならない最小限のものである点に留意すべきである。

【0107】

[逆方向変換]

ハールおよび $2/10$ 変換は両方とも可逆であり、損失なしおよび損失ありの圧縮両方に適している。しかしながら、説明した形態の上述の数式を使う場合、「詳細」出力にはビット成長がある（LL または「平滑」出力にはビット成長はない）。かかる理由により、好ましいシステムでは、出力変換データは上ですでに参照した「精度保存の特性」の原則を使って演算され、これにより損失なし性能を維持しつつもビット成長はなくなる。（このようにして適用される精度保存の特性（PPP）は Hongyang, Fisher および Zeyi によるものである）

【0108】

変換数式に関して理解すべき重要な点は、これらがすべて整数ドメインで演算されるがそれでも損失なしの結果を生み出す点である。ここでの洞察は Pearlman によるものであり、また Ricoh の Gormish らにもよるものである。

【0109】

ウェーブレット変換を実行するために設定される数式は上ですでに提供済みである。ここではプロセスを逆転させ画素データを回復するための対応する数式がその後に引き続く。

【0110】

図 10 および図 11 に示す変換結果が以下に続く数式（適切な範囲にわたって演算される）に供給されると、現れる画素データはまさに図 8 に示すとおりのものになるであろう。

【0111】

数式セット 13. L および H を回復するための垂直逆転ハール変換

【0112】

10

20

30

【数 1 6】

$$L_{r,2,c} := LL_{r,c} + \text{floor}\left(\frac{LH_{r,c} + 1}{2}\right)$$

$$L_{r,2+1,c} := LL_{r,c} - \text{floor}\left(\frac{LH_{r,c}}{2}\right)$$

10

$$H_{r,2,c} := HL_{r,c} + \text{floor}\left(\frac{HH_{r,c} + 1}{2}\right)$$

$$H_{r,2+1,c} := HL_{r,c} - \text{floor}\left(\frac{HH_{r,c}}{2}\right)$$

【0 1 1 3】

20

数式セット 1 4 . 画素を回復するための水平逆転ハール変換

【0 1 1 4】

【数 1 7】

$$P_{r,2,c,2} := L_{r,2,c} + \text{floor}\left(\frac{H_{r,2,c} + 1}{2}\right)$$

$$P_{r,2,c,2+1} := L_{r,2,c} - \text{floor}\left(\frac{H_{r,2,c}}{2}\right)$$

30

$$P_{r,2+1,c,2} := L_{r,2+1,c} + \text{floor}\left(\frac{H_{r,2+1,c} + 1}{2}\right)$$

$$P_{r,2+1,c,2+1} := L_{r,2+1,c} - \text{floor}\left(\frac{H_{r,2+1,c}}{2}\right)$$

40

【0 1 1 5】

数式セット 1 5 . L および H を回復するための垂直逆転 2 / 1 0 変換

【0 1 1 6】

【数 1 8】

$$L_{r,c} := LL_{r,c} + \text{floor}\left(\frac{LH_{r,c} - pLH_{r,c} + 1}{2}\right)$$

$$L_{r+1,c} := LL_{r,c} - \text{floor}\left(\frac{LH_{r,c} - pLH_{r,c}}{2}\right)$$

10

$$H_{r,c} := HL_{r,c} + \text{floor}\left(\frac{HH_{r,c} - pHH_{r,c} + 1}{2}\right)$$

$$H_{r+1,c} := HL_{r,c} - \text{floor}\left(\frac{HH_{r,c} - pHH_{r,c}}{2}\right)$$

【0 1 1 7】

20

数式セット 1 6 . 画素を回復するための水平逆転 2 / 1 0 変換

【0 1 1 8】

【数 1 9】

$$P_{r,c} := L_{r,c} + \text{floor}\left(\frac{H_{r,c} - pH_{2,r,c} + 1}{2}\right)$$

$$P_{r,c+1} := L_{r,c} - \text{floor}\left(\frac{H_{r,c} - pH_{2,r,c}}{2}\right)$$

30

$$P_{r+1,c} := L_{r+1,c} + \text{floor}\left(\frac{H_{r+1,c} - pH_{2,r+1,c} + 1}{2}\right)$$

$$P_{r+1,c+1} := L_{r+1,c} - \text{floor}\left(\frac{H_{r+1,c} - pH_{2,r+1,c}}{2}\right)$$

40

【0 1 1 9】

[変換エンジンの動作]

変換エンジンを実際に実現するにあたっての最も基本的かつ重要な点は、タスクを、それぞれ 1 つずつが完全に決定的な方法で動作する多数の簡素な工程にまで分解することである。解決すべきいくつかの問題点は次のとおりである。

(a) 2 / 1 0 変換が必要とする「ブロック外」画素データを取り扱うこと (ハール変換に関しては、3 2 × 3 2 ブロックはそれ自体で処理できるが、2 / 1 0 は、3 2 × 3 2 変換を完成させるのにブロック全体および部分的なブロックからの画素からのデータを必要とする。)

50

(b) 変換エンジン成分には垂直データを取り扱っているのか水平データを取り扱っているのかが「分からなくても」よいようにタスクを簡素化すること。それぞれの素子はただ単に簡素な算術的タスクを実行するだけにとどまるべきである。

(c) 処理時間を短縮するための方策を見出すこと。5レベル2次元変換処理には、続けざまにオペレーションを実行する必要性が内在しているため、単一フレーム分に相当する画素データを処理するのにかかる時間が複数となる。明らかに、1つのフレームを変換し符号化するプロセス全体がすべてもとのフレーム時間より短い時間内で実行できることが確実である必要がある。

【0120】

図12は、符号化システムアーキテクチャ全体のブロック図であるが、ただしこのステージではアイテム1から7だけを説明することにする。YUV変換処理はすでに上で説明済みである。符号器およびパケタイザを以下に説明する。適正な変換エンジン、これは実際には2つの変換エンジンおよび大型メモリからなるものであるが、における主なプロセスをここで説明する。

【0121】

1. 画像データは変換エンジンの2つの画素に同時に入り、最初のタスクはレベル1水平変換である。これにより数式1、および数式2または数式4のいずれかに従ってLおよびHデータが生成される。H数式は、予測子を除いて同じであるため、ハール変換の場合であればpをゼロに設定した状態で単一セットの数式を使うことが可能なことがわかる。図13は、どのようにして典型的な画素nについてのデータが導き出されるかを示している。s(n)およびd(n)の値を導き出すためのフィルタ数式を以下に示す。図13は、どのようにして2/10変換の場合の予測子pが導き出されるかを示している。変換エンジンそれ自体は座標には関与しないため数式はsまたは「平滑な」成分、および/または「詳細」成分を示す簡素化された形態で表される。このステージでは、これらはLおよびHに対応する。

【0122】

【数20】

$$s(n) = \lfloor (x(2n) + x(2n+1))/2 \rfloor$$

$$d(n) = x(2n) - x(2n+1) + p(n)$$

【0123】

2. 図12は8ビット色を想定しているため、変換エンジンへの入力48ビット幅で示されている（一度に2つの画素が読み取られる）。これが現れるときは、符号の追加により54ビット幅になっている。「18」から「36」ボックスが4つの変換係数のデータを組み合わせることによってデータを108ビット幅に変換する。これはメモリをローディングするのにかかる時間を短縮するための策略であるため、後続の2次元変換に必要とされるメモリに対する複数のパリアクセスを行うための時間がかせげる。

【0124】

3. 2つの変換エンジン1、5が大型DDR（二重データ速度）メモリ3によって供給される。メモリ3の入力および出力には多重スイッチ（MUX）が備わっている。入力側のスイッチは2つの変換エンジンの出力からのデータのいずれかを選択し、出力側のスイッチは第2の変換エンジンまたはコーダのいずれかにデータを送信する。メモリ3はデータの2つの画像フレームの等価物を含むのに十分な大きさである。連続的な奇数番号のフレームからの変換データはメモリ3の第1のセクションに保存され、偶数番号のフレームからの変換データは第2のセクションに保存される。

【0125】

4. 第1の変換の出力からのデータは 32×32 フォーマットでメモリ3から読み出される。後続のレベルの変換を実行するにはデータが第2の変換エンジンを通る複数のパスを経ることが必要である。エンジン自体が「ダム」となることができそれが横列のデータを処理しているのか縦列のデータを処理しているのかには無関係であるために、変換エンジンの外部には横列および縦列の制御が設けられている。変換エンジンに到着する前に、データは54ビット幅に並び戻される。

【0126】

5. 外部の横列および縦列制御を使うという概念は、第2の変換エンジン(5)を第1の変換エンジンと同一にするというものである。これ自身は単次元でのみ機能するが、横列および縦列データを連続して取り扱うことで2次元の変換を生み出す。5レベル変換を生み出すには、YUVブロックデータは変換エンジンを通る複数のパスを持っていないなければならない。これがフレーム時間内で可能な理由は、レベル1変換が時間の大部分を占める(約75%)ためである。後続のレベルは、複数のパスを必要とするものの、実際にはあまり時間をとることはない、なぜなら係数の数がずっと少ないからである(図6を参照のこと)。2/10変換を実行するには、再循環されたデータは「ブロック外」係数を含んでいなければならない点に留意する。

【0127】

6. 第2変換エンジンの出力はメモリに戻される前に108ビット幅に並び戻される。図14は、メモリ内における連続的な書き換えの概念を示している。左側はレベル1変換の結果であり、レベル2変換が完了すると、レベル1データのLL部分だけがレベル2データで上書きされる。この図面から、なぜ再循環されたデータの量がそれぞれのレベルの変換が完了するごとに少なくなるのなのかが明らかである。いったんレベル1の2次元変換が完了すると、係数は図面の左側に従って保存される。その後、LL係数がレベル2についての新しいセットの係数によって上書きされる。これらは右側に描かれるようにメモリ内でまさに同じ空間を占める。このプロセスがレベル5まで繰り返される。

【0128】

7. 完成されたYUVブロック変換データはMUXによって符号器セクションに解放される。

【0129】

次の点に留意することが重要である。まず第1には、元のYUVデータがRGBオリジナルに関して本質的に損失のないものであることを確認し、このデータすべてが変換プロセスへと進むことを確認する点。このことは専門的なビデオ用語ですべての処理が「4:4:4」である、と言うのに等しく、「縁部」での色漏れが絶対にならないようにするものである。そして、第2には、変換ステージではブロック間で2/10係数を保存する概念によりフレームベースの変換の数多くの等価物が実現される点。これにより、最終的に得られる結果はブロックのない画像忠実度である。しかしながら、すべての変換管理およびすべてのその後のコード化はブロックドメイン内で行われる。このことは効率的かつ決定的な動作を得るにあたって鍵である。

【0130】

[得られたデータの符号化]

上で述べたように、変換の当初の効果はデータの量を少なくすることではなく、圧縮をより効率的にするような形態でデータを提示することにすぎない。

【0131】

データ圧縮は標準的な数学的方法(用途によって全く左右される)を使って行えるが、下層データの性質を利用するとよりよい結果を得ることができる。ウェーブレット変換データは、データが「ツリー」構造に組織されている場合、効率的な損失のない圧縮および損失のある圧縮に非常に向いている。

【0132】

「ツリー」の使用の背後にある基本的な概念は、画像内の隣接し合う画素が似る傾向に

10

20

30

40

50

あることである。変換ドメイン内では、このことは別のやり方で表わされる。分解のより高いサブ帯域におけるウェーブレット係数の大きさが特定のしきい値に対してあまり重要でない場合、同じ空間的位置を持つがより低いサブ帯域に関連しているウェーブレット係数もまたあまり重要ではない見込みがある。さらに、ウェーブレット「ピラミッド」のもっとも高いレベルからもっとも低いレベルへと進む場合、ウェーブレット係数の変動は小さくなる。このことにより、数多くの重要でないウェーブレット係数のコード化を非常に効率的に行えるという概念に結び付く。

【0133】

公知の方法としては、空間オリエンテーションツリーまたはSOT (Shapiro)、およびヒエラルキーツリーにおけるパーティショニングセットSPHT (Pearlman) が挙げられる。これら方法の両方が抱える問題点は、1回より多くデータを訪問する必要があることである。好ましい方法はまた、「4分ツリー」形態のツリー原則も使用するが、データを一回だけ訪問するだけでよいやり方でこれを使用する。これにより、正確に定義されたサイクルでタスクを実行するリアルタイムの単一パス圧縮エンジンの生成が可能となる。

10

【0134】

システムの目的は、2つの異なるタイプの情報をコード化することである。1つの情報は「制御」またはコード化情報であり、他方は「データ」である。コード化情報はデータに先駆けて送信されるため、復号化システムはその後に続くデータをどのように取り扱うかを前もって知る。基本的な符号化システムは損失なしであるが、正確に定義されたレベルの損失あり圧縮に非常に向いている。

20

【0135】

[LKCS符号化]

個々の画像に関するデータは 32×32 ウェーブレット係数のブロックに分割される。そしてこのブロックデータは9つの面、すなわち8つのデータ面と1つの符号面に分離される。そしてそれぞれの面は、図19に示し以下により詳細に説明するように16ビットの64列に並べ替えられる。

【0136】

図15は、どのようにしてこのような1つの列が「CKLツリー」に符号化されるかを示している。話を簡素化するために16個の係数のデータビットを一行に示しているが、これらは実際には2次元のアレーについて言及していることを心に留めておかねばならない。これら16個の係数は4つのセットに分割され、それぞれのセットが「K型ツリー」に接続される。セット内のすべての係数がゼロであれば、対応するK型もまたゼロであり、K型を保持するだけでよい。もしセットがゼロでなければ、元の係数およびK型を保持する必要がある。(プール項では、Kツリーは4つの入力を持つORゲートである。出力が0であれば、 $K=0$ の情報だけが保持される。出力が1であれば、 $K=1$ の情報および4つの個々のデータビットを保持しなければならない。)

30

【0137】

4つのK型もまた1つのセットを形成しており、同じ確率法則に従うため、ツリー概念を繰り返すことが可能である。K型セットはツリーをL型に形成する。従って、K型セットがゼロであれば、L型だけを保持する必要がある。

40

【0138】

次のステップは、ここのビット面内でL型ツリーを符号化することである。それぞれのL型は64横列ブロック内の横列を表しており、これは64L型のLツリー構造に完全に合う。図16は、これがどのようにして起こるかを示しており、またL型がどのように元に変換データ(HL、LHおよびHH)に関連するかも示している。図面は、レベル1および2における20個のL型、およびレベル3、4、および5における4つの最終的なL型を示している。これらはまた図面に示すようにレベル1および2におけるLHおよびHHそれぞれについての20個のL型でもある。

【0139】

50

Lツリーはやはり類似の可能性を利用する。符号化はツリーの底部（レベル1）からレベル4／5へと行われる。ヒエラルキー用語で、レベル4／5は「親」とみなすことができ、レベル3，2および1はそれに対して「子」の関係を持つ。符号化手順は上と同じである。

【0140】

符号化「ノード」の正確な動作を図16に示す。プロセスはL2__0と印が付けられているノードを考慮することによって図示できる。ここで、ブール演算は入力5つのORゲートのものであり、入力のうち4つがL1__0からL1__3データであり、5番目の入力がL2__0データである。上で述べたように、ゲートの出力が1であれば、L型および前のデータを保持しなければならないが、もし0であれば、L型だけを保持する。そしてレベル3ノードからレベル4／5までプロセスを繰り返す。

10

【0141】

ゼロ係数がある広い領域があると、非常に大型の符号化ゲインが実現されることがわかる。そして極端な場合、1つのビット面にあるすべての係数がゼロであれば、レベル4／5だけが保持される。

【0142】

ここで、コード化プロセスによりデータを大幅に縮小できるが、LおよびK値を保持しなければならない付帯部分があることがわかる。LおよびKビット自体は元のデータに対する付け加えである、すなわち、ツリープロセスが元のデータを縮小する一方で、制御データの追加もできる。コード化ゲインのいくらかは失われるため、この損失を最小限にすることが望ましい。このことは、8つのデータ面のすべての概要を把握することによって行われる。図17は面の概念を示しており、面7が最上位であり面0が最下位である。ウェーブレット変換のおかげで、面7はゼロのほとんどを含んでいるため、この面では係数のほとんどがゼロであろうにもかかわらずKおよびL構造がコード化ゲインの意味で最も効率的となっている。

20

【0143】

KおよびL型を見て調べるやり方の1つは、これらが面における係数の有意性の記録を提供することである。この記録は1つの面から他の面へと手渡され、対応するKまたはL型がいつ有意になったか（すなわち1に等しくなったか）を判断するのに使用できる。いったんこれが検出されると、もはや後続の面のために型を保存する必要はない（なぜならデータはいずれにしろ保持されているからである）。この手順は冗長なL型およびK型を排除するものである。

30

【0144】

連続的な面を走査するプロセスも使って符号面（面8）をコード化する。変換プロセスでは8ビット画素データが9ビットとなるが、これは±255の範囲を示している。符号面では、1024個の係数が0によって正に指定されており、1によって負に指定されている。符号データは有意性に基づいてコード化されているため、（冗長なKおよびL型を排除するために）有意性の走査が行われると、有意な係数だけが符号化された符号データを持つ（なぜなら、明らかに、すでに廃棄されているゼロ係数に符号データをコード化することは冗長と思われるからである）。

40

【0145】

今や全体の符号化プロセスはLKCSデータの生成として要約でき、この場合、それぞれの面は一連の4つのセクションでコード化される。ここで、

L = L型ツリー

K = K型ツリー

C = 係数データ

S = 符号

である。

【0146】

損失のない符号化を行うには、「最悪の場合」、すなわち、元の画像が非常に複雑であ

50

るため実際にはコード化ゲインがない場合について符号化データを計画する必要がある。従って、プロセスは次のとおりである。

(a) Lツリーは64個までのビットデータによってコード化され、L型に対応する。これら自体はK型の知識から導き出されるが、このセクションがビットストリームの最初でなければならない。これは、復号器はどの横列が送信されておりどの横列が送信されていないか(有意でないか)を前もって知る必要があるからである。L型ビットは、圧縮プロファイル(以下を参照のこと)とともに、復号器にL型ツリーを再構築させる。

(b) 次に、K型が256個のK型に対応する256個までのビットデータによってコード化される。復号器は再構築されたLツリーを使ってK型のマップを復号化する。

(c) 次に、もとの係数データCが1024個までのビットによってコード化される。復号器は再構築されたLおよびK型を使ってCデータのマップを復号化する。 10

(d) 最後に、符号データSが1024個までのビットデータによってコード化される。復号器を使ってCデータを再構築してSデータのマップを復号化する。

【0147】

8つの画像面のそれぞれについてLKCSPROCESS全体が繰り返される。

【0148】

[空間的圧縮のための符号化]

明らかに、上述のプロセスがいったん完了すると、実際の符号化されたデータの長さが可変であるという状況が生じる。符号化損失があるという状況さえもあることは統計学的にはあり得ない(不可能でさえある)ものの、損失なしのコード化により可変の結果がもたらされ、かかる結果は、意図するリアルタイム用途で管理するのは困難と思われる場合がそうである。 20

【0149】

ビット速度の意味で、より予測可能な結果を実現した高いコード化ゲインを持つ損失のない圧縮を導くために、LKCSPROCESSデータは圧縮プロファイルを受ける。原則として、これは解像度およびビット面数に基づくデータの除去にすぎない。このプロファイルはヘッダとしてビットストリームに送付され、その結果、復号器はどれが削除されたかを前もって知る。

【0150】

連続的なプロファイルの傾向は、ほとんどの積極的な削除を面0およびレベル1に適用し、レベルおよび面の上昇につれて削除を累進的に少なくするというものである。実際には、圧縮プロファイルはCCKおよびLツリーのコード化のときに適用されるもので、このことは望ましくない係数データおよび対応するKおよびL型のいずれも削除することを意味する。このことは重要である、なぜならこれにより元のデータおよび圧縮されつつある制御情報の両方が得られるからである、さもなければ高い圧縮レベルでは制御情報が優勢になるという状況が生じたであろう。 30

【0151】

[圧縮プロファイル]

圧縮プロファイルは、人間の目の視覚的知覚特徴を活用する重み付け方法を使用する。原則として、人間の目は高周波数の損失にはあまり敏感ではないため、どのような圧縮体系でも、高周波数成分を排除することによってまた量子化ノイズの効果も確実にやはり排除することによって始まる。 40

【0152】

重み付け体系はウェーブレット変換のサブ帯域レベルに合わされており、その概念を図18に示す(図6と比較すべきものである)。これを言い換えると、圧縮は元のデータではなくLツリーに適用される。

【0153】

図18で、簡単な(そして典型的な)例として「a」を $a=2$ とする。すると、いずれのレベルのHHも対応するLHおよびHLの圧縮の2倍を持つことがわかる。さらに、累進的により小さい圧縮を、より重要な情報が存在しているより高いレベルに適用する。 50

【0154】

好ましい重み付け方法は、「a」の値を変動させて広い範囲の圧縮比を得ることである。図18は、適用される相対的な重みを運搬する点で概念的であるが、実際には、圧縮の範囲は符号器で適用される1セットの個々のプロファイルの形態である。ユーザーは所定のプロファイルのうちの1つを選択できるかまたはカスタムプロファイルを定義さえもできる。

【0155】

プロファイルを定義するにあたりその目的は、再構築された画像が最小限の間違いしか持たず、同時に最も高い知覚品質を持つ（すなわち、画像が見る者によって「自然だ」と知覚される）ことを確実にすることである。このことにより必然的にいくつかの妥協が生じ、実際には、知覚的品質はより低いビット速度でより重要である。

10

【0156】

重み付け体系は簡素で、効率的で、実際の画像には左右されない。この体系は、Lツリーの1つのビット面に直接関連する16ビットプロファイルパラメータによって成し遂げられる。このことを表1に示す。もしある1つのビットが0であれば、データは取り除かれ、そのビットが1に等しければ、データは保持される。プロセスは「Lツリーの伐採」と記述でき、L型の単数または複数のビットを使う論理AND演算によって実行される。

【0157】

例えば、ビット10=0であれば、L__LH2の4つすべてのビットはゼロとされるであろうが、もしビット10=1であれば、L__LH2のうち値が1のビットだけが保持されるであろう。

20

【0158】

「スペアビット」の存在を説明する必要がある。元の構造では、L__4/5の個々の成分のために空間が見込まれていた。実際には、これはすべての通常の画像にとって冗長であるが、万が一、後の開発（おそらく非常に大型の画像をとまなう）でこれら余分のビットが必要とされる場合に備えて機能が保持されている。チップ設計はこれらを使用するための機能を保持しているが、体系が冗長なデータを送信することはない。

【0159】

L型ツリーの制御により非常に効率的な圧縮を行うことができる、なぜならより多くのゼロが形成される場合、データおよび制御の両方ともプロファイルパラメータによって取り除かれるからである。

30

【0160】

[プロファイルの例]

表2および表3は、圧縮プロファイルの例を示している。表2は、視覚的に損失のない画像を提供する圧縮プロファイルの実例である。1または0に設定されるビットを検討することにより、プロファイルの重み付けの概念を得ることが可能であり、有意な面およびレベルには圧縮がかけられていないことが直ちに明らかである。かかるプロファイルは20:1から30:1の範囲の空間的圧縮を提供できる。

【0161】

表3は、範囲が50:1から100:1の激しい圧縮の一例であり、より多くのデータが廃棄されていることが明白である。すべての面についてL4/5データが保持されており、これは、かかるデータのどのような損失も画像品質に重大な影響を及ぼすと思われるからであるが、かたやビット速度には限界的な削減しか行われていない点に留意する必要がある。

40

【0162】

プロファイルを定義する場合、特定の解像度レベルに関するすべてのデータが確実にビット面内で除去されるようなやり方でデータの除去を行うことが重要である。なぜなら、さもないと得られる画像は不均一的な空間品質を持つことになると思われるからである。目はかかる潜在的な人工物に敏感である、例えば、人間の顔面を見ている場合、目は均

50

一な品質を期待しているが、もし顔面の異なる部分が異なる品質を持っていればとまどう。

【0163】

〔符号化エンジン〕

これまで行ってきた符号化プロセスの説明では数多くの離散プロセスを説明してきた。図12および図13には、ウェーブレット変換を実行するための「変換エンジン」の概念をいくつか詳細に示した。しかしながら、図12では、コード化プロセスがただ単にブロック図内の機能として示されていたにすぎず、符号器がどのように作動するかについての詳細はまったくなかった。

【0164】

〔変換データの並べ替え〕

変換プロセスの結果は 32×32 ブロックの画像係数であり（3セットのかかるブロック、すなわちY、U、およびVがある）、ブロック内で、レベル5データから始めに係数データが整列され、レベル1データで終わる。

【0165】

それぞれのビット面について、データが、最初にそれぞれが16ビットの64横列並び替えられる、なぜならこうするとLツリーを簡単に導き出すことができるからである。係数データに対する組織およびその関係は以下の図面ではっきりとわかる。

【0166】

〔L符号器〕

上で述べたように、Lツリーがまず最初に導き出される、なぜならLツリーはまず最初に復号化プロセスに必要でありまた最も多い量の係数データの廃棄ももたらすからである。「Lツリーコード化エンジン」のタスクは非常に複雑である、なぜならこれは3次元で作動しなければならないからである。

(a) 論理AND演算をデータに行って所望の圧縮プロファイルをかけるようにしなければならない。

(b) Lタイプの引き出し自体は非常に簡素である、なぜならこれはデータの単一の横列に対する論理OR演算であり、横列63から横列0まで作用するものである。

(c) もし係数データが有意であることがすでにわかっているのであれば、L型を指定することは冗長であるため、プロセスは最上位の面の下流で作用しなければならない。

(d) 所望の最終結果はすべての有意でない係数データの廃棄、および残りの係数データの保持、およびすべてのL型の位置のコンパクトな記述である。

(e) エンジンは、データを「再訪問」してはならないという点で、単一のパスをベースとして機能しなければならない。

【0167】

図20は、プロセスをブロック図の形態で示している。LFORMATは1つの面の 64×16 係数ビットから64個のLを構築している。

【0168】

LTREEは8つの面を生成する。すなわち、最上位の面7から最下位の面0まで機能するL__CUR[63..0]、L__MSK[63..0]およびL__SIG[3..0]である。圧縮ファイルがこのステージでどのように適用されるかに留意すること。これらのアイテムはL符号器の「出力」を表している。ここで、

L__CUR[63..0]は現在の面のLツリー状態である。

【0169】

L__MSK[63..0]は、どのL__CURビットを送信すべきでないかを決定するマスクである。

【0170】

L__SIG[3..0]はL__SIGNIFICANCEであり、K、CおよびSパスによって使用されるもので、どの横列を送信すべきでないかを示している。

【0171】

10

20

30

40

50

L_A C CはL__A C C [6 3 .. 0] を生成する。これはすべての以前の面の現在のO R化された状態の記録である。

【 0 1 7 2 】

L 符号器で使用される数式は次のとおりである。

【 0 1 7 3 】

[L__C U R および L__S D I G の定義]

[圧縮プロファイル]

【 0 1 7 4 】

【 数 2 1 】

L__cur[0]	= L4/5	C_prof[15..0]	10
L__cur[1]	= L3_LH	LL5 = [0]	
L__cur[2]	= L3_HL	LH5 = [1]	
L__cur[3]	= L3_HH	HL5 = [2]	
		HH5 = [3]	
L__cur[7..4]	= L2_LH[3..0]	LH4 = [4]	
L__cur[11..8]	= L2_HL[3..0]	HL5 = [5]	
L__cur[15..12]	= L2_HH[3..0]	HH5 = [6]	
L__cur[31..16]	= L1_LH[15..0]	LH3 = [7]	
L__cur[47..32]	= L1_HL[15..0]	HL3 = [8]	
L__cur[63..48]	= L1_HH[15..0]	HH3 = [9]	20
		LH2 = [10]	
		HL2 = [11]	
		HH2 = [12]	
L_sig[3..0],		LH1 = [13]	
is the L significance for one 64-way row (total = 64 * 16)		HL1 = [14]	
		HH1 = [15]	

【 0 1 7 5 】

[L__C U R [6 3 .. 0] U P L ツリーを算出するための論理数式]

数式

コメント

【 0 1 7 6 】

30

【数 2 2】

$L_cur[0] = (L[0] \# L_cur[3] \# L_cur[2] \# L_cur[1]) \& C_prof[0];$ (L4/5 L[0] or child)

$L_cur[1] = (L[1] \# L_cur[7] \# L_cur[6] \# L_cur[5] \# L_cur[4]) \& C_prof[7];$ (L3_LH)

$L_cur[2] = (L[2] \# L_cur[11] \# L_cur[10] \# L_cur[9] \# L_cur[8]) \& C_prof[8];$ (L3_HL)

$L_cur[3] = (L[3] \# L_cur[15] \# L_cur[14] \# L_cur[13] \# L_cur[12]) \& C_prof[9];$ (L3_HH)

$L_cur[4] = (L[4] \# L_cur[19] \# L_cur[18] \# L_cur[17] \# L_cur[16]) \& C_prof[10];$ (L2_LH[0])

$L_cur[5] = (L[5] \# L_cur[23] \# L_cur[22] \# L_cur[21] \# L_cur[20]) \& C_prof[10];$ (L2_LH[1])

$L_cur[6] = (L[6] \# L_cur[27] \# L_cur[26] \# L_cur[25] \# L_cur[24]) \& C_prof[10];$ (L2_LH[2])

$L_cur[7] = (L[7] \# L_cur[31] \# L_cur[30] \# L_cur[29] \# L_cur[28]) \& C_prof[10];$ (L2_LH[3])

10

$L_cur[8] = (L[8] \# L_cur[35] \# L_cur[34] \# L_cur[33] \# L_cur[32]) \& C_prof[11];$ (L2_HL[0])

$L_cur[9] = (L[9] \# L_cur[39] \# L_cur[38] \# L_cur[37] \# L_cur[36]) \& C_prof[11];$ (L2_HL[1])

$L_cur[10] = (L[10] \# L_cur[43] \# L_cur[42] \# L_cur[41] \# L_cur[40]) \& C_prof[11];$ (L2_HL[2])

$L_cur[11] = (L[11] \# L_cur[47] \# L_cur[46] \# L_cur[45] \# L_cur[44]) \& C_prof[11];$ (L2_HL[3])

$L_cur[12] = (L[12] \# L_cur[51] \# L_cur[50] \# L_cur[49] \# L_cur[48]) \& C_prof[12];$ (L2_HH[0])

$L_cur[13] = (L[13] \# L_cur[55] \# L_cur[54] \# L_cur[53] \# L_cur[52]) \& C_prof[12];$ (L2_HH[1])

$L_cur[14] = (L[14] \# L_cur[59] \# L_cur[58] \# L_cur[57] \# L_cur[56]) \& C_prof[12];$ (L2_HH[2])

$L_cur[15] = (L[15] \# L_cur[63] \# L_cur[62] \# L_cur[61] \# L_cur[60]) \& C_prof[12];$ (L2_HH[3])

$L_cur[19..16] = L[19..16] \& C_prof[13];$ (L1_LH[3..0])

$L_cur[23..20] = L[23..20] \& C_prof[13];$ (L1_LH[7..4])

$L_cur[27..24] = L[27..24] \& C_prof[13];$ (L1_LH[11..8])

$L_cur[31..28] = L[31..28] \& C_prof[13];$ (L1_LH[15..12])

20

$L_cur[35..32] = L[35..32] \& C_prof[14];$ (L1_HL[3..0])

$L_cur[39..36] = L[39..36] \& C_prof[14];$ (L1_HL[7..4])

$L_cur[43..40] = L[43..40] \& C_prof[14];$ (L1_HL[11..8])

$L_cur[47..44] = L[47..44] \& C_prof[14];$ (L1_HL[15..12])

$L_cur[51..48] = L[51..48] \& C_prof[15];$ (L1_HH[3..0])

$L_cur[55..52] = L[55..52] \& C_prof[15];$ (L1_HH[7..4])

$L_cur[59..56] = L[59..56] \& C_prof[15];$ (L1_HH[11..8])

$L_cur[63..60] = L[63..60] \& C_prof[15];$ (L1_HH[15..12])

30

$L_cur[63..0] = L_cur_L[63..0] \& !L_acc[63..0];$

$L_cur[n]$ can only be significant for 1 plane (transition to significance); note that that L coding stops beyond the point of becoming significant.

【0 1 7 7】

$L_cur[n]$ は1つの面についてのみ有意である（有意への遷移）。有意になる点を超えるとLコード化は停止する点に留意すること。

【0 1 7 8】

[L_SIG を算出するための論理数式]

40

L_SIG はK、CおよびSパスによって使用されるもので、どの横列を送信しないかを示している。 $L_n_XX_sig = 0$ の場合、横列は送信されない。 $L_sig[3..0]$ は4つの横列をマップする、すなわち、16サイクルK、CおよびSパスを処理するための16セットのシーケンスがあり、ここで $sel[0]$ から $sel[15]$ がそのシーケンスを選択する。

【0 1 7 9】

【数 2 3】

```

L4_sig          = (L_cur[0] # L_acc[0]) & C_prof[0];

L3_LH_sig       = (L_cur[1] # L_acc[1]) & C_prof[7];
L3_HL_sig       = (L_cur[2] # L_acc[2]) & C_prof[8];
L3_HH_sig       = (L_cur[3] # L_acc[3]) & C_prof[9];

L2_LH_sig[3..0] = (L_cur[7..4] # L_acc[7..4]) & C_prof[10];
L2_HL_sig[3..0] = (L_cur[11..8] # L_acc[11..8]) & C_prof[11];
L2_HH_sig[3..0] = (L_cur[15..12] # L_acc[15..12]) & C_prof[12];

L1_LH_sig[15..0] = (L_cur[31..16] # L_acc[31..16]) & C_prof[13];
L1_HL_sig[15..0] = (L_cur[47..32] # L_acc[47..32]) & C_prof[14];
L1_HH_sig[15..0] = (L_cur[63..48] # L_acc[63..48]) & C_prof[15];

```

10

【0 1 8 0】

L__sig [3..0] は4つの有意値をそれぞれの(4*16) K、C、S言語に手渡す。それぞれのタイプ(K、C、S)について16の3つのパスがある。

【0 1 8 1】

【数 2 4】

```

L_sig[3..0] = ((L3_HH_sig, L3_HL_sig, L3_LH_sig, L4_sig) & sel[0]) #
               (L2_LH_sig[3..0] & sel[1]) #
               (L2_HL_sig[3..0] & sel[2]) #
               (L2_HH_sig[3..0] & sel[3]) #
               (L1_LH_sig[3..0] & sel[4]) #
               (L1_LH_sig[7..4] & sel[5]) #
               (L1_LH_sig[11..8] & sel[6]) #
               (L1_LH_sig[15..12] & sel[7]) #
               (L1_HL_sig[3..0] & sel[8]) #
               (L1_HL_sig[7..4] & sel[9]) #
               (L1_HL_sig[11..8] & sel[10]) #
               (L1_HL_sig[15..12] & sel[11]) #
               (L1_HH_sig[3..0] & sel[12]) #
               (L1_HH_sig[7..4] & sel[13]) #
               (L1_HH_sig[11..8] & sel[14]) #
               (L1_HH_sig[15..12] & sel[15]).

```

20

30

【0 1 8 2】

[L__MSKを算出するための論理数式]

これを使ってL__cur [63..0]のどのLビットを送信しないかを決定する。Lビットは、その親が0である場合またはそのC__prof[]が0の場合は送信されない。これは下降(上から下向き)ツリーを組み込んでいる。それぞれのパスは平面であり、最上位から最下位へと実行される(面7から面0)。

40

【0 1 8 3】

コメント

【0 1 8 4】

【数 2 5】

L_msk[0] = !L_acc[0] & C_prof[0];	(L4/5)	
L_msk[1] = !L_acc[1] & C_prof[7] & (L_cur[0] # L_acc[0]);	(L3_LH () = parent)	
L_msk[2] = !L_acc[2] & C_prof[8] & (L_cur[0] # L_acc[0]);	(L3_HL)	
L_msk[3] = !L_acc[3] & C_prof[9] & (L_cur[0] # L_acc[0]);	(L3_HH)	
L_msk[7..4] = !L_acc[7..4] & C_prof[10] & (L_cur[1] # L_acc[1]);	(L2_LH)	
L_msk[11..8] = !L_acc[11..8] & C_prof[11] & (L_cur[2] # L_acc[2]);	(L2_HL)	
L_msk[15..12] = !L_acc[15..12] & C_prof[12] & (L_cur[3] # L_acc[3]);	(L2_HH)	
L_msk[19..16] = !L_acc[19..16] & C_prof[13] & (L_cur[4] # L_acc[4]);	(L1_LH[3..0])	10
L_msk[23..20] = !L_acc[23..20] & C_prof[13] & (L_cur[5] # L_acc[5]);	(L1_LH[7..4])	
L_msk[27..24] = !L_acc[27..24] & C_prof[13] & (L_cur[6] # L_acc[6]);	(L1_LH[11..8])	
L_msk[31..28] = !L_acc[31..28] & C_prof[13] & (L_cur[7] # L_acc[7]);	(L1_LH[15..12])	
L_msk[35..32] = !L_acc[35..32] & C_prof[14] & (L_cur[8] # L_acc[8]);	(L1_HL[3..0])	
L_msk[39..36] = !L_acc[39..36] & C_prof[14] & (L_cur[9] # L_acc[9]);	(L1_HL[7..4])	
L_msk[43..40] = !L_acc[43..40] & C_prof[14] & (L_cur[10] # L_acc[10]);	(L1_HL[11..8])	
L_msk[47..44] = !L_acc[47..44] & C_prof[14] & (L_cur[11] # L_acc[11]);	(L1_HL[15..12])	
L_msk[51..48] = !L_acc[51..48] & C_prof[15] & (L_cur[12] # L_acc[12]);	(L1_HH[3..0])	
L_msk[55..52] = !L_acc[55..52] & C_prof[15] & (L_cur[13] # L_acc[13]);	(L1_HH[7..4])	
L_msk[59..56] = !L_acc[59..56] & C_prof[15] & (L_cur[14] # L_acc[14]);	(L1_HH[11..8])	
L_msk[63..60] = !L_acc[63..60] & C_prof[15] & (L_cur[15] # L_acc[15]);	(L1_HH[15..12])	20

【0 1 8 5】

[CS 符号器]

図 2 1 は CS 符号器を示している。この内部では、次のことが行われる。

【0 1 8 6】

CS FORMAT が元の 16 ビット横列フォーマット [15..0] を a × 4 横列フォーマット、すなわち [63..0] に変換する。これを行うことでデータを 64 ビットに合致させ、その結果、符号化エンジンの最終的な部分が 64 ビットベースでのみ機能できるようになる。

30

【0 1 8 7】

すべての係数面について符号データが平行して複写される。これは、すべての C 面について符号が利用可能である必要がある次のステージにとって必要である。

【0 1 8 8】

C ACC はそれぞれの係数が有意となった点を記録するもので、符号をいつ符号化すべきかを決定するのに次のステージで使用される。

【0 1 8 9】

[LKCS パス]

図 2 2 は符号化エンジン全体を示している。ここで、L ENCODE および CS ENCODE はすでに上で説明したプロセスである。

40

【0 1 9 0】

MX LDPS は符号化エンジンである。その所望の出力は、MX__CUR [63..0] および MX__MSK [63..0] からなる。図 2 2 に示されるその他の出力は中間データで、出力を算出するのに使用されるものであり以下に示す数式に現れる。

【0 1 9 1】

リアルタイム符号化エンジンは 64 サイクルベースで機能するため、最大値にある L、K、C および S のそれぞれの理論上の最悪のケースが実際に「合う」ことが重要なことである。このことは、次のことがらを理解することで試験される。

【0 1 9 2】

L__PASS = 面あたり 1 × L__CUR [63..0]

50

K__P A S S = 面あたり 1 6 × K__C U R [1 5 .. 0]

C__P A S S = 面あたり 1 6 × C__C U R [6 3 .. 0]

S__P A S S = 面あたり 1 6 × S__C U R [6 3 .. 0]

従って、MX__C U R および MX__M S K を生成するには、L、K、C および S パスのすべてのシーケンスが必要である、すなわち、

1 + 1 6 + 1 6 + 1 6 = 面あたり 4 9 サイクル

これは十分に 6 4 サイクルの許容範囲内である。

【 0 1 9 3 】

出力 MX__M S K [6 3 .. 0] は L、K、C および S__C U R [] のそれぞれのどのビットを符号化すべきかを選択するためのマスクである。

10

【 0 1 9 4 】

L K C S パスで使用される数式はここで次のとおりである。

【 0 1 9 5 】

[C 累算からの K 累算の導出]

【 0 1 9 6 】

【 数 2 6 】

```
K_acc[0] = C_acc[0] # C_acc[1] # C_acc[2] # C_acc[3];
K_acc[1] = C_acc[4] # C_acc[5] # C_acc[6] # C_acc[7];
K_acc[2] = C_acc[8] # C_acc[9] # C_acc[10] # C_acc[11];
K_acc[3] = C_acc[12] # C_acc[13] # C_acc[14] # C_acc[15];
```

20

```
K_acc[4] = Cacc[16] # Cacc[17] # C_acc[18] # C_acc[19];
K_acc[5] = Cacc[20] # Cacc[21] # Cacc[22] # C_acc[23];
K_acc[6] = Cacc[24] # Cacc[25] # Cacc[26] # C_acc[27];
K_acc[7] = Cacc[28] # Cacc[29] # Cacc[30] # C_acc[31];
```

```
K_acc[8] = C_acc[32] # C_acc[33] # C_acc[34] # C_acc[35];
K_acc[9] = C_acc[36] # C_acc[37] # C_acc[38] # C_acc[39];
K_acc[10] = C_acc[40] # C_acc[41] # C_acc[42] # C_acc[43];
K_acc[11] = C_acc[44] # C_acc[45] # C_acc[46] # C_acc[47];
```

```
K_acc[12] = C_acc[48] # C_acc[49] # C_acc[50] # C_acc[51];
K_acc[13] = C_acc[52] # C_acc[53] # C_acc[54] # C_acc[55];
K_acc[14] = C_acc[56] # C_acc[57] # C_acc[58] # C_acc[59];
K_acc[15] = C_acc[60] # C_acc[61] # C_acc[62] # C_acc[63];
```

30

【 0 1 9 7 】

[C__c u r からの K 型の累算]

【 0 1 9 8 】

【数 2 7】

```

K[0] = C_cur[0] # C_cur[1] # C_cur[2] # C_cur[3];
K[1] = C_cur[4] # C_cur[5] # C_cur[6] # C_cur[7];
K[2] = C_cur[8] # C_cur[9] # C_cur[10] # C_cur[11];
K[3] = C_cur[12] # C_cur[13] # C_cur[14] # C_cur[15];

```

```

K[4] = C_cur[16] # C_cur[17] # C_cur[18] # C_cur[19];
K[5] = C_cur[20] # C_cur[21] # C_cur[22] # C_cur[23];
K[6] = C_cur[24] # C_cur[25] # C_cur[26] # C_cur[27];
K[7] = C_cur[28] # C_cur[29] # C_cur[30] # C_cur[31];

```

10

```

K[8] = C_cur[32] # C_cur[33] # C_cur[34] # C_cur[35];
K[9] = C_cur[36] # C_cur[37] # C_cur[38] # C_cur[39];
K[10] = C_cur[40] # C_cur[41] # C_cur[42] # C_cur[43];
K[11] = C_cur[44] # C_cur[45] # C_cur[46] # C_cur[47];

```

```

K[12] = C_cur[48] # C_cur[49] # C_cur[50] # C_cur[51];
K[13] = C_cur[52] # C_cur[53] # C_cur[54] # C_cur[55];
K[14] = C_cur[56] # C_cur[57] # C_cur[58] # C_cur[59];
K[15] = C_cur[60] # C_cur[61] # C_cur[62] # C_cur[63];

```

【0 1 9 9】

20

[K パス]

K__cur の導出 (K__パスおよび C__パスの両方に必要である)

【0 2 0 0】

【数 2 8】

```

K_cur[15..0] = K[15..0] & (!K_acc[15..12] & L_sig[3],
                !K_acc[11..8] & L_sig[2],
                !K_acc[7..4] & L_sig[1],
                !K_acc[3..0] & L_sig[0]) & (K_pass_ # C_pass_);

```

```

K_cur[15..0] = K_cur[15..0] & K_pass_;

```

30

【0 2 0 1】

[K__パスマスクの作成]

【0 2 0 2】

【数 2 9】

```

K_msk[15..0] = (!K_acc[15..12] & L_sig[3],
                !K_acc[11..8] & L_sig[2],
                !K_acc[7..4] & L_sig[1],
                !K_acc[3..0] & L_sig[0]) & K_pass_;

```

40

【0 2 0 3】

[C パス]

C__cur の導出

【0 2 0 4】

【数 3 0】

```

C_cur[63..0] = (C_cur[63..48] & L_sig[3],
                C_cur[47..32] & L_sig[2],
                C_cur[31..16] & L_sig[1],
                C_cur[15..0] & L_sig[0]) & P_pass_;

```

【 0 2 0 5】

[C __ パスマスクの準備および作成]

【 0 2 0 6】

10

【数 3 1】

```

a[15..0]      = ( K_cur[15..0] #(K_acc[15..12] & L_sig[3],
                        K_acc[11..8] & L_sig[2],
                        K_acc[7..4] & L_sig[1],
                        K_acc[3..0] & L_sig[0]) ) & C_pass_;

```

```

C_msk[63..0] = (a[15],a[15],a[15],a[15],
                a[14],a[14],a[14],a[14],
                a[13],a[13],a[13],a[13],
                a[12],a[12],a[12],a[12],
                a[11],a[11],a[11],a[11],
                a[10],a[10],a[10],a[10],
                a[9],a[9],a[9],a[9],
                a[8],a[8],a[8],a[8],
                a[7],a[7],a[7],a[7],
                a[6],a[6],a[6],a[6],
                a[5],a[5],a[5],a[5],
                a[4],a[4],a[4],a[4],
                a[3],a[3],a[3],a[3],
                a[2],a[2],a[2],a[2],
                a[1],a[1],a[1],a[1],
                a[0],a[0],a[0],a[0]).

```

20

30

【 0 2 0 7】

[S パス]

【 0 2 0 8】

【数 3 2】

```

S_msk[63..0] = C_cur[63..0] & (!C_acc[63..48] & L_sig[3],
                        !C_acc[47..32] & L_sig[2],
                        !C_acc[31..16] & L_sig[1],
                        !C_acc[15..0] & L_sig[0]) & S_pass_;

```

```

S_cur[63..0] = S[63..0] & S_msk[63..0].

```

40

【 0 2 0 9】

[M X L K C S]

【 0 2 1 0】

【数 3 3】

```

MX_cur[63..0] = (L_cur[63..0] & L_pass) #
                ((z[63..16], K_cur[15..0]) & K_pass) #
                (C_cur[63..0] & C_pass) #
                (S_cur[63..0] & S_pass);

```

```

MX_msk[63..0] = (L_msk[63..0] & L_pass) #
                ((z[63..16], K_msk[15..0]) & K_pass) #
                (C_msk[63..0] & C_pass) #
                (S_msk[63..0] & S_pass).

```

10

【0 2 1 1】

[復号化エンジン]

復号化エンジンは、符号化フォーマットを鏡写しにすることによって「1つのパス」符号化解答を提供する1セットの決定的原則に基づいている。このフォーマットは、先駆けて知っておくべき後続のデータのための1セットのポインタを見込んだ累進的な計算を規定している。従属的なフィードバック素子を含むパイプライン化された論理構造では、将来のデータの位置を前もって知ることは必須要件である、さもなければ遅延（パイプライン）によりリアルタイムでない復号器となる。

【0 2 1 2】

符号器と同様に、復号器は面あたり64ビット×64サイクルベースで機能する。復号器は埋め込まれた制御およびデータをそれが符号化された順序と同じ順序、すなわちLKCで累進的に復号する。

20

【0 2 1 3】

[L復号器]

L制御ビット[63..0]の復号は2つのパスで行われる。

【0 2 1 4】

Lパス1 = レベル4、3、2 = L[15..0]

Lパス2 = レベル1 = L[63..16]

Lパス1は、どの面についてもシリアルデータd[15..0]の最初の16ビットで機能する。次の入力、

L_acc[15..0]

C_prof[15..0]

と一緒に、次の8つの面を生成する、すなわち、

L_cur[15..0]

L4_sig

L3_LH_sig

L3_HL_sig

L3_HH_sig

L2_LH_sig[3..0]

L2_HL_sig[3..0]

L2_HH_sig[3..0]

これらパラメータの定義は符号化エンジンにおいて定義される。

30

40

【0 2 1 5】

[L__パス1数式]

Lデータのためのポインタ

【0 2 1 6】

【 数 3 4 】

```

pointer for L3_LH
s1_[1..0]) = 2 x (!L_acc[0] & C_prof[0]);

pointer for L3_HL
s2_[2..0]) = s1_[1..0] x 2 x (!L_acc[1] & C_prof[7]);

pointer for L3_HH
s3_[3..0]) = s2_[2..0] x 2 x (!L_acc[2] & C_prof[8]);

pointer for start of L2 data
s4_[4..0]) = s3_[3..0] x 2 x (!L_acc[3] & C_prof[9]);

pointer for L2_LH[3]
not required (base = 0)

pointer for L2_LH[2]
s5_[1..0] = 2 x (!L_acc[5] & L3_LH_sig_ & C_prof[10]);

pointer for L2_LH[1]
s6_[2..0]) = s5_[1..0] x 2 x (!L_acc[6] & L3_LH_sig_ & C_prof[10]);

pointer for L2_LH[0]
s7_[3..0]) = s6_[2..0] x 2 x (!L_acc[7] & L3_LH_sig_ & C_prof[10]);

pointer for L2_HL[3]
s8_[4..0]) = s7_[3..0] x 2 x (!L_acc[8] & L3_HL_sig_ & C_prof[11]);

pointer for L2_HL[2]
s9_[5..0]) = s8_[4..0] x 2 x (!L_acc[9] & L3_HL_sig_ & C_prof[11]);

pointer for L2_HL[1]
s10_[6..0]) = s9_[5..0] x 2 x (!L_acc[10] & L3_HL_sig_ & C_prof[11]);

pointer for L2_HL[0]
s11_[7..0]) = s10_[6..0] x 2 x (!L_acc[11] & L3_HL_sig_ & C_prof[11]);

pointer for L2_HH[3]
s12_[8..0]) = s11_[7..0] x 2 x (!L_acc[12] & L3_HH_sig_ & C_prof[12]);

pointer for L2_HH[2]
s13_[9..0]) = s12_[8..0] x 2 x (!L_acc[13] & L3_HH_sig_ & C_prof[12]);

pointer for L2_HH[1]
s14_[10..0]) = s13_[9..0] x 2 x (!L_acc[14] & L3_HH_sig_ & C_prof[12]);

pointer for L2_HH[0]
s15_[11..0]) = s14_[10..0] x 2 x (!L_acc[15] & L3_HH_sig_ & C_prof[12]);

```

10

20

30

40

【 0 2 1 7 】

[L データ]

レベル 4 および 3 のための L __ c u r [3 .. 0]

【 0 2 1 8 】

【 数 3 5 】

```

L_cur[0] = d[0] & (!L_acc[0] & C_prof[0]);

L_cur[1] = ( (d[1] & s1_[1]) #
             (d[0] & s1_[0]) ) &
             ((!L_acc[1] & C_prof[7]) & L4_sig_);

L_cur[2] = ( (d[2] & s2_[2]) #
             (d[1] & s2_[1]) #
             (d[0] & s2_[0]) ) &
             ((!L_acc[2] & C_prof[8]) & L4_sig_);

L_cur[3] = ( (d[3] & s3_[3]) #
             (d[2] & s3_[2]) #
             (d[1] & s3_[1]) #
             (d[0] & s3_[0]) ) &
             ((!L_acc[3] & C_prof[9]) & L4_sig_);

```

10

【 0 2 1 9 】

レベル 2 L データの位置範囲 ([1 5 .. 4])

【 0 2 2 0 】

20

【 数 3 6 】

```

L2_[4] = s4_[4..0] & d[4..0];
L2_[5] = s4_[4..0] & d[5..1];
L2_[6] = s4_[4..0] & d[6..2];
L2_[7] = s4_[4..0] & d[7..3];
L2_[8] = s4_[4..0] & d[8..4];
L2_[9] = s4_[4..0] & d[9..5];
L2_[10] = s4_[4..0] & d[10..6];
L2_[11] = s4_[4..0] & d[11..7];
L2_[12] = s4_[4..0] & d[12..8];
L2_[13] = s4_[4..0] & d[13..9];
L2_[14] = s4_[4..0] & d[14..10];
L2_[15] = s4_[4..0] & d[15..11];

```

30

【 0 2 2 1 】

レベル 2 のための L__c u r [1 5 .. 4]

【 0 2 2 2 】

【 数 3 7 】

L2_LH_cur

```
L_cur[4] = L2_[4] & (!L_acc[4] & L3_LH_sig_ & C_prof[10]);
```

```
L_cur[5] = ( (L2_[5] & s5_[1]) #
  (L2_[4] & s5_[0]) ) &
  (!L_acc[5] & L3_LH_sig_ & C_prof[10]);
```

```
L_cur[6] = ( (L2_[6] & s6_[2]) #
  (L2_[5] & s6_[1]) #
  (L2_[4] & s6_[0]) )&
  (!L_acc[6] & L3_LH_sig_ & C_prof[10]);
```

10

```
L_cur[7] = ( (L2_[7] & s7_[3]) #
  (L2_[6] & s7_[2]) #
  (L2_[5] & s7_[1]) #
  (L2_[4] & s7_[0]) )&
  (!L_acc[7] & L3_LH_sig_ & C_prof[10]);
```

【 0 2 2 3 】

20

【 数 3 8 】

L2_HL_cur

```

L_cur[8] = ( (L2_[8] & s8_[4]) #
             (L2_[7] & s8_[3]) #
             (L2_[6] & s8_[2]) #
             (L2_[5] & s8_[1]) #
             (L2_[4] & s8_[0]) ) &
             (!L_acc[8] & L3_HL_sig_ & C_prof[11]);

```

10

```

L_cur[9] = ( (L2_[9] & s9_[5]) #
             (L2_[8] & s9_[4]) #
             (L2_[7] & s9_[3]) #
             (L2_[6] & s9_[2]) #
             (L2_[5] & s9_[1]) #
             (L2_[4] & s9_[0]) ) &
             (!L_acc[9] & L3_HL_sig_ & C_prof[11]);

```

```

L_cur[10] = ((L2_[10] & s10_[6]) #
             (L2_[9] & s10_[5]) #
             (L2_[8] & s10_[4]) #
             (L2_[7] & s10_[3]) #
             (L2_[6] & s10_[2]) #
             (L2_[5] & s10_[1]) #
             (L2_[4] & s10_[0]) ) &
             (!L_acc[10] & L3_HL_sig_ & C_prof[11]);

```

20

```

L_cur[11] = ((L2_[11] & s11_[7]) #
             (L2_[10] & s11_[6]) #
             (L2_[9] & s11_[5]) #
             (L2_[8] & s11_[4]) #
             (L2_[7] & s11_[3]) #
             (L2_[6] & s11_[2]) #
             (L2_[5] & s11_[1]) #
             (L2_[4] & s11_[0]) ) &
             (!L_acc[11] & L3_HL_sig_ & C_prof[11]);

```

30

【 0 2 2 4 】

【 数 3 9 】

L2_HH_cur

```

L_cur[12] = ((L2_[12]&s12_[8])#
  (L2_[11]&s12_[7])#
  (L2_[10]&s12_[6])#
  (L2_[9]&s12_[5])#
  (L2_[8]&s12_[4])#
  (L2_[7]&s12_[3])#
  (L2_[6]&s12_[2])#
  (L2_[5]&s12_[1])#
  (L2_[4]&s12_[0]))&
  (!L_acc[12]&L3_HH_sig_&C_prof[12]);

```

10

```

L_cur[13] = ((L2_[13]&s13_[9])#
  (L2_[12]&s13_[8])#
  (L2_[11]&s13_[7])#
  (L2_[10]&s13_[6])#
  (L2_[9]&s13_[5])#
  (L2_[8]&s13_[4])#
  (L2_[7]&s13_[3])#
  (L2_[6]&s13_[2])#
  (L2_[5]&s13_[1])#
  (L2_[4]&s13_[0]))&
  (!L_acc[13]&L3_HH_sig_&C_prof[12]);

```

20

```

L_cur[14] = ((L2_[14]&s14_[10])#
  (L2_[13]&s14_[9])#
  (L2_[12]&s14_[8])#
  (L2_[11]&s14_[7])#
  (L2_[10]&s14_[6])#
  (L2_[9]&s14_[5])#
  (L2_[8]&s14_[4])#
  (L2_[7]&s14_[3])#
  (L2_[6]&s14_[2])#
  (L2_[5]&s14_[1])#
  (L2_[4]&s14_[0]))&
  (!L_acc[14]&L3_HH_sig_&C_prof[12]);

```

30

```

L_cur[15] = ((L2_[15]&s15_[11])#
  (L2_[14]&s15_[10])#
  (L2_[13]&s15_[9])#
  (L2_[12]&s15_[8])#
  (L2_[11]&s15_[7])#
  (L2_[10]&s15_[6])#
  (L2_[9]&s15_[5])#
  (L2_[8]&s15_[4])#
  (L2_[7]&s15_[3])#
  (L2_[6]&s15_[2])#
  (L2_[5]&s15_[1])#
  (L2_[4]&s15_[0]))&
  (!L_acc[15]&L3_HH_sig_&C_prof[12]);

```

40

【 0 2 2 5 】

【数 4 0】

L significance

```

L4_sig_ = L_cur[0] # (L_acc[0] & C_prof[0]);
L3_LH_sig_ = L_cur[1] # (L_acc[1] & C_prof[7]);
L3_HL_sig_ = L_cur[2] # (L_acc[2] & C_prof[8]);
L3_HH_sig_ = L_cur[3] # (L_acc[3] & C_prof[9]);

L2_LH_sig[3..0] =(L_cur[7..4] # L_acc[7..4]) & C_prof[10];
L2_HL_sig[3..0] =(L_cur[11..8] # L_acc[11..8]) & C_prof[11]; ;
L2_HH_sig[3..0] =(L_cur[15..12] # L_acc[15..12]) & C_prof[12]; ;

```

10

【0 2 2 6】

L__パス 2 は、L__パスのデータの最後からあらかじめ指定されているデータ d [6 3 .. 1 6] の範囲で機能する。次の入力、

```

L_acc[63..16]
L2_LH_sig[3..0]
L2_HL_sig[3..0]
L2_HH_sig[3..0]
C_prof[15..0]

```

20

とともに、次の 8 つの面を生成する、すなわち、

```

L_cur[63..16]
L_acc[63..16]
L1_LH_sig[15..0]
L1_HL_sig[15..0]
L1_HH_sig[15..0]

```

これらのパラメータは符号化エンジンにおいて定義される。

【0 2 2 7】

【数 4 1】

```

a[63..16] = (d[63..16] & !L_acc[63..16]);

L_cur[19..16] = a[19..16] & L2_LH_sig[0] & C_prof[13];
L_cur[23..20] = a[23..20] & L2_LH_sig[1] & C_prof[13];
L_cur[27..24] = a[27..24] & L2_LH_sig[2] & C_prof[13];
L_cur[31..28] = a[31..28] & L2_LH_sig[3] & C_prof[13];

L_cur[35..32] = a[35..32] & L2_HL_sig[0] & C_prof[14];
L_cur[39..36] = a[39..36] & L2_HL_sig[1] & C_prof[14];
L_cur[43..40] = a[43..40] & L2_HL_sig[2] & C_prof[14];
L_cur[47..44] = a[47..44] & L2_HL_sig[3] & C_prof[14];

L_cur[51..48] = a[51..48] & L2_HH_sig[0] & C_prof[15];
L_cur[55..52] = a[55..52] & L2_HH_sig[1] & C_prof[15];
L_cur[59..56] = a[59..56] & L2_HH_sig[2] & C_prof[15];
L_cur[63..60] = a[63..60] & L2_HH_sig[3] & C_prof[15];

b[63..16] = (d[63..16] & !L_acc[63..16]) # L_acc[63..16];

L1_LH_sig[3..0] = b[19..16] & L2_LH_sig[0] & C_prof[13];
L1_LH_sig[7..4] = b[23..20] & L2_LH_sig[1] & C_prof[13];
L1_LH_sig[11..8] = b[27..24] & L2_LH_sig[2] & C_prof[13];
L1_LH_sig[15..12] = b[31..28] & L2_LH_sig[3] & C_prof[13];

L1_HL_sig[3..0] = b[35..32] & L2_HL_sig[0] & C_prof[14];
L1_HL_sig[7..4] = b[39..36] & L2_HL_sig[1] & C_prof[14];
L1_HL_sig[11..8] = b[43..40] & L2_HL_sig[2] & C_prof[14];
L1_HL_sig[15..12] = b[47..44] & L2_HL_sig[3] & C_prof[14];

L1_HH_sig[3..0] = b[51..48] & L2_HH_sig[0] & C_prof[15];
L1_HH_sig[7..4] = b[55..52] & L2_HH_sig[1] & C_prof[15];
L1_HH_sig[11..8] = b[59..56] & L2_HH_sig[2] & C_prof[15];
L1_HH_sig[15..12] = b[63..60] & L2_HH_sig[3] & C_prof[15].

```

10

20

30

【0 2 2 8】

[K 復号器]

K__パスは、L__パス 2 のデータの最後からあらかじめ指定されているデータの範囲で機能する。次の入力、

```

16 x d[15..0]
16 x C_acc[63..0]
16 x L_sig[3..0]

```

但し、L__s i g [3 . 0] は、（パラメータ）から（パラメータ）への順次的な四重マッピングである。

40

とともに、

```

面あたり 16 x K__c u r [ 1 5 .. 0 ]
面あたり 16 x K__m s k [ 1 5 .. 0 ]

```

を生成する。

【0 2 2 9】

[C 累算からの K 累算]

【0 2 3 0】

【数 4 2】

```

K_acc[0] = C_acc[0] # C_acc[1] # C_acc[2] # C_acc[3];
K_acc[1] = C_acc[4] # C_acc[5] # C_acc[6] # C_acc[7];
K_acc[2] = C_acc[8] # C_acc[9] # C_acc[10] # C_acc[11];
K_acc[3] = C_acc[12] # C_acc[13] # C_acc[14] # C_acc[15];

```

```

K_acc[4] = C_acc[16] # C_acc[17] # C_acc[18] # C_acc[19];
K_acc[5] = C_acc[20] # C_acc[21] # C_acc[22] # C_acc[23];
K_acc[6] = C_acc[24] # C_acc[25] # C_acc[26] # C_acc[27];
K_acc[7] = C_acc[28] # C_acc[29] # C_acc[30] # C_acc[31];

```

```

K_acc[8] = C_acc[32] # C_acc[33] # C_acc[34] # C_acc[35];
K_acc[9] = C_acc[36] # C_acc[37] # C_acc[38] # C_acc[39];
K_acc[10] = C_acc[40] # C_acc[41] # C_acc[42] # C_acc[43];
K_acc[11] = C_acc[44] # C_acc[45] # C_acc[46] # C_acc[47];

```

```

K_acc[12] = C_acc[48] # C_acc[49] # C_acc[50] # C_acc[51];
K_acc[13] = C_acc[52] # C_acc[53] # C_acc[54] # C_acc[55];
K_acc[14] = C_acc[56] # C_acc[57] # C_acc[58] # C_acc[59];
K_acc[15] = C_acc[60] # C_acc[61] # C_acc[62] # C_acc[63];

```

```

K_cur[15..0] = d[15..0] & (!K_acc[15..12] & L_sig[3],
                    !K_acc[11..8] & L_sig[2],
                    !K_acc[7..4] & L_sig[1],
                    !K_acc[3..0] & L_sig[0]) & K_pass_en;

```

```

K_msk[15..0] = ( K_cur[15..0] # (K_acc[15..12] & L_sig[3],
                    K_acc[11..8] & L_sig[2],
                    K_acc[7..4] & L_sig[1],
                    K_acc[3..0] & L_sig[0]) ) & K_pass_en;

```

【 0 2 3 1】

[C 復号器]

C__パスは、L__パスのデータの最後からあらかじめ指定されているデータの範囲で機能する。次の入力、

```

16 x d[63..0]
16 x C_acc[63..0]
16 x K_msk[15..0]
16 x L_sig[3..0]

```

とともに、

面あたり 16 × C__c u r [6 3 .. 0]

面あたり 16 × S__m s k [6 3 .. 0]

を生成する。

【 0 2 3 2】

10

20

30

40

【数 4 3】

```

C_cur[63..0] = d[63..0] &
    (K_msk[15], K_msk[15], K_msk[15], K_msk[15],
     K_msk[14], K_msk[14], K_msk[14], K_msk[14],
     K_msk[13], K_msk[13], K_msk[13], K_msk[13],
     K_msk[12], K_msk[12], K_msk[12], K_msk[12],
     K_msk[11], K_msk[11], K_msk[11], K_msk[11],
     K_msk[10], K_msk[10], K_msk[10], K_msk[10],
     K_msk[9], K_msk[9], K_msk[9], K_msk[9],
     K_msk[8], K_msk[8], K_msk[8], K_msk[8],
     K_msk[7], K_msk[7], K_msk[7], K_msk[7],
     K_msk[6], K_msk[6], K_msk[6], K_msk[6],
     K_msk[5], K_msk[5], K_msk[5], K_msk[5],
     K_msk[4], K_msk[4], K_msk[4], K_msk[4],
     K_msk[3], K_msk[3], K_msk[3], K_msk[3],
     K_msk[2], K_msk[2], K_msk[2], K_msk[2],
     K_msk[1], K_msk[1], K_msk[1], K_msk[1],
     K_msk[0], K_msk[0], K_msk[0], K_msk[0]) &
    C_pass_en;
10

S_msk[63..0] = C_cur_[63..0] & (!C_acc[63..48] & L_sig[3],
    !C_acc[47..32] & L_sig[2],
    !C_acc[31..16] & L_sig[1],
    !C_acc[15..0] & L_sig[0]) & C_pass_en;
20

```

【0 2 3 3】

[S 復号器]

S __パスは、C __パスのデータの最後からあらかじめ指定されているデータの範囲で機能する。次の入力、

16 x d[63..0]

16 x S_msk[63..0]

とともに、

16 x S_cur[63..0] per plane

S_cur[63..0] = (d[63..0] & S_msk[63..0]) & S_pass_en ;

を生成する。

【0 2 3 4】

[一時的な圧縮のための符号化]

一時的な圧縮は、高い圧縮比を実現するための鍵である。しかしながら、方法によっては計算上集約的なものがあり、処理時間が画像内容に大きく左右される。好ましい体系では、次の2つの優先度に対応している。

(a) どのような方法を使おうとも、変換の決定性およびコード化エンジンを保持しなければならない。このようにすると、プロセス全体が簡素化され、内容を符号化するのにかかる時間が正確に定義される。

(b) ストリーム化すべきデータは「絶対的」でなければならない。言い換えると、画像は受け取ったデータだけを使って再構築され、画像の履歴または将来の予測には依存しない。絶対的なデータの概念によりネットワークエラーに対する高い免疫が提供され、特に、画像待ち時間を延長することがない。(延長された画像待ち時間、すなわち、符号化および復号化の間における複数のフレーム遅延は画像のグループ全体にわたって複雑な計算を必要とするシステムではどのようなものにおいても不可避である。)

【0 2 3 5】

好ましい一時的圧縮体系の基礎は、変更のあった映像情報だけをコード化することである。この体系は、映像内容の領域は、その体系が検出しコード化しないいくつかのフレームにわたって静止したままにできるという事実を活用する。このようにすると、大きなコ

10

20

30

40

50

ード化ゲインが実現される。この体系が実行可能なものとなるには、変更を正確かつ確実に検出することは最も重要な点である、なぜなら変更をどのように誤って検出しても明らかなエラーにつながり、これは復号化された画像の「凍結した」領域としてはっきりと示されるものである。

【0236】

動きを確実に検出することは体系の核心である。しかしながら、絶対的データの送信の基づいた体系を考案することは、変化の間の違いを送信することだけに頼る体系（例えば、MP EGを使って行われているように）を使うことに比べるとはるかに困難である。画像にノイズがあるため困難が生じるため、その結果、真の映像内容とノイズとを区別する問題がある。ノイズは2つの主な理由により生じる。すなわち、カメラセンサノイズ（特に、証明レベルの低い場面）およびアナログからデジタル信号変換から生じる量子化ノイズである。

10

【0237】

ノイズと画像内容とを区別する好ましい方法の基礎は、ウェーブレットドメインにおいて、すなわち、コード化の前に、変換出力において動き検出を処理することである。ウェーブレットドメインにおける「脱ノイズ」は、ウェーブレット変換は信号ドメインにおけるノイズを変換におけるノイズにマップするということに気づいた人物であるDonohoによって最初に提案された概念に基づいている。

【0238】

どのような所定の画像でも、信号エネルギーは変換ドメインの係数にはほとんど集中しないが、ノイズエネルギーはそうではない。ノイズから信号の分離を可能とするのはこの重要な原則であり、ウェーブレット係数を「しきい値処理」することによって実現される。ノイズは有意の係数よりもレベルが非常に低いため、インテリジェントな低レベルしきい値処理を適用することで、ノイズと思われる低レベルの係数だけを取り除くことができる。しきい値処理は、最適なノイズ抑制を実現するために変換レベル全体にわたって動的である。好ましい体系は新規である、なぜなら信号は非線形手段によってノイズから分離されるからである。そしていくつかのやり方においては、プロセスは、上で説明した圧縮プロファイルに使用され適用される方法に類似である。

20

【0239】

好ましい一時的な圧縮体系では、最上位の係数の貧弱なセットだけをノイズ除去のための基礎として使用する。かかる積極的なやり方は、動きを検出するための非常にきれいな「ウェーブレット署名」を得るように設計されている。この「署名」は認識可能な映像に似る必要はなく、ただ単に有効な変更検出の手段であればよい。

30

【0240】

〔一時的な圧縮の定義〕

一時的な圧縮アルゴリズムの目的は、必要な計算を最小限とすることである。好ましいシステムでは、当初の色空間変換の性質を利用する。

【0241】

動きを定義するための境界は、 32×32 係数のYUV変換されたブロックである。それぞれのブロックは画像フレーム内で自身の位置を定義するようなやり方で番号付けされている。フレーム間に対応し合うブロックを比較し、これらが異なる場合だけ、コード化し送信される。Y自体はUおよびVから導き出されると考えることができるため、動きを査定するにはY係数だけを使えば十分である。このことは動きの計算およびフレーム保存の要件を、もし計算がYUV（またはRGB）画像全部について行われた場合に比べたった3分の1にまで少なくする効果がある。

40

【0242】

一時的な符号化のプロセスを図19に図示しており、これはプロセスにおける次のステップを示している。

【0243】

1. Y変換情報を 32×32 ブロックで抽出する。それぞれのブロックについて位置情

50

報を割り当てる。

【0244】

2. ノイズしきい値をデータに適用する。これによりプログラムされた値未満のすべての係数を排除する。この「しきい値」は非常に低く、ノイズレベルにある有意でない係数を排除することだけを目的としている。

【0245】

3. 最上位の係数の大きさおよび位置を検出する。このプロセスでは、5レベル変換を形成している16個のサブ帯域をそれぞれろ過して最上位の係数およびその関連する位置を選択する。

【0246】

4. 16個の得られた係数から、最上位なものを選択する。選択される個数はプログラム可能で、実際には最大8つで十分であることがわかっている。この体系の背後にある概念は、信頼性のある動き検出を保証するのに十分な情報を得るが、それと同時に、最上位の係数のグループの大きさに上限を設けることにより最大のノイズ免疫を実現するというものである。有意な係数および対応する位置データを要約するこの情報は「ウェーブレット署名」と呼ばれる。

【0247】

5. 得られた「署名」を以前の画像フレーム内の対応するブロックのものと比較する。このステージでは、別のプログラム可能なしきい値が適用される。この「異なるしきい値」によりある一定の比較が可能となり得るが、この比較が正しいと思われるかどうかはまだ定かでない。このしきい値は、係数間の小さなピーク変調差を見込んでおり、大きさ情報だけに適用され、位置情報には適用されない。

【0248】

6. 比較の結果、もし署名が同じであれば送信はない。もし署名が異なれば、コード化のためのデータ送信がある点に留意すること。コード化のために前進するデータは下のYUV変換データである。これは重要な原則である、なぜならこれにより（圧縮プロファイルの制約内で）最も高いと思われる画像品質が保持されること、およびコード化／復号化プロセスが静止している画像データと移動している画像データとを区別しなくてよいことが保証されるからである。

【0249】

[参照フレームデータ]

一時的な圧縮体系はまた、（単数または複数の）復号器を符合器の現状に同期させるための参照フレームデータを出力するためにも組織される。このプロセスは「背景」画像リフレッシュ機能を提供するものと考えられる。

【0250】

この機能は、ステップ6（上記を参照のこと）で取られた決定に関わりなく、確実にブロックの全YUV変換データが間隔をあけて送信されるようにするものである。この間隔はプログラム可能で、確実に参照フレームデータの送信が出力ネットワークにおいてデータフローに対し最小限の影響を持つようになっている。パラメータは「x個の通常のブロックごとに送信される1つの参照ブロック」によって定義される。xは典型的には100以上である。

【0251】

このリフレッシュ機構は画像解像度とは無関係であり、画像における一時的な変更とは非対称であるため、画像内容とは無関係である。この機構はただ単に、1つのブロックについて最新の更新を送信するに過ぎず、このブロックは現在のフレーム内のブロックの指標によって支配されている。高い解像度の画像全体をリフレッシュするのにこのやり方では数秒間かかる場合があることがわかる。しかし、かかるシステムは静止画像に悪影響を及ぼすエラー（例えばネットワークの問題から生じるエラー）に効果的に対処し、例えば、新しいユーザーがログオンする場合に複数のユーザーをサポートする、さもないとこれら複数のユーザーは元の画像が変更されるまで静止画像データを受信することはないで

10

20

30

40

50

あろう。

【0252】

〔ネットワーク接続〕

ここでのタスクは、圧縮データをイーサネットネットワーク全体にわたって渡すことのできる形態へと変換することである。画像データはコード化された「画像ブロック」の形態であり、これらはそれぞれが画素の 32×32 アレーを記述している。かかるブロックは必ずしもイーサネットのペイロード仕様に一致するわけではない。加えて、デジタルオーディオデータを最終的なデータストリームに多重化するための規定がなくてはならない。

【0253】

それぞれのYUVブロックは最上位のデータを先頭とし最下位のデータを追跡することをベースに符号化される。ブロックフォーマットを表4に示す。

【0254】

【表1】

フィールド	ビットサイズ	説明
ID	32	同期ワードー16ビット 指標番号ー11ビット（フレーム内におけるブロックの位置を定義） 余白ー5ビット
ブロックデータ	可変	YUV圧縮化データ Yビット面7つのL型 Yビット面7つのK型 Yビット面7つのZC型 Yビット面7つのS型 Yビット面6つのL型 など、 Yビット面S型なし、までこの順序。その後、UおよびVについて繰り返し

表4：画像ブロックフォーマット

【0255】

〔ユーザーデータグラムプロトコル（UDP）の選択〕

本発明の背後にある開発プログラムの初期には、デジタルリンク全体にわたって画像データを多重化し送信する数多くの異なる方法が考慮された。しかし、その後、インターネットプロトコルを使う汎用のイーサネットネットワークに乗じることにするという決定が下された。そして、システムが確実に「現実世界」で機能した実行する上でどのような

困難も絶対に引き起こさないことが重要であった。

【0256】

その結果、ネットワーク全体にわたってデータを送信する方法を定義する際の手引きとなる原則は次のとおりである。

(a) 目的は、同期画像送達の要件と食い違ってネットワークは事実上、非同期であるという事実にもかかわらず、リアルタイム画像の信頼性があり効率的なトランスポートである。

(b) システムは既存のネットワークトランスポート基準およびプロトコルに基づいていなければならない。

(c) ネットワーク全体にわたってシステムの複雑性が低くなければならない。

(d) システムはマルチノードシステムとして機能しなければならない（すなわち、典型的には1つの画像供給源が複数の「ユーザー」または「視聴者」に分配されている）。

(e) 上述したことの当然の帰結として、（単数または複数の）ディスプレイノードをいかなるやり方でも管理するのに（単数または複数の）捕獲ノードは不必要でなければならない。これによりノードの計算上の複雑性が最小となり、（この実行において）スケーラビリティが提供される。

【0257】

技術上の要件は、図24に示すように、データを、IEEE 802.3メディアアクセスコントロール(MAC)フレームに一致するフォーマットに取り入れることである。上述の最後の要件は、「マルチキャスト」法を指しており、これを実現するのに利用可能な方法は、MAC「ペイロード」を、ユーザーデータグラムプロトコル(UDP)に準拠する「データグラム」の形態にあるインターネットプロトコル(IP)に従わせることである。マルチキャストメッセージは特別なマルチキャスト「目的地アドレス」を持っている。最大データパケットサイズは1500バイトであり、これはなんらかのプロトコルオーバーヘッドを備えていなければならない点に留意すること。より大きなパケットはギガバイトイーサネットが可能である。

【0258】

UDPは、簡素さおよび最小のデータオーバーヘッドという最大の長所をもっている。しかしながら、IPと同じく、これはコネクションレス型プロトコルでありそれ自体は信頼性のある通信をなんら保証もせず、どのような形態のエラー訂正または回復、またはどのような種類のメッセージフロー制御も行わない。通信は「最大努力」であり、欠点を克服する何らかの手段がアプリケーションに存在しなければならない。

【0259】

双方向通信（信頼性のあるポイント間通信を提供する、接続重視のプロトコルトランスポートコントロールプロトコル(TCP)で使用されているような）の必要を完全に省くために、好ましいシステムはパケット損失に対して頑丈なように設計されている。表4は、特有のやり方でコード化された同期ワードによってそれぞれのデータブロックが分離されていることを示している。万が一、1つのブロックまたは一連のブロックからのデータが破損された場合に備えて、同期ワードがエラーの伝播を捕獲するように設計されている。同期ワードを使ってエラーを「くるみ」、これにより復号化されたビットストリームが不要情報を表示するのを防止する。（単数または複数の）エラーが生じた場合は、最新の手付かずの1つのブロックまたは複数のブロックが表示され続ける。

【0260】

[IPパケットへの変換]

画像データのIP/UDPへの準拠は2ステージプロセスである。最初のステージは元のコード化YUVブロックデータを一連の標準化データパケットに変換することである。あらゆる付随のオーディオが画像データで多重化されるのはこの時点である。オーディオはAES/SPDIF標準に従い非圧縮デジタルオーディオとして搬送される（ドルビーAC3もまた搬送できる）。表5はパケットフォーマットを示している。

【0261】

【表 2】

フィールド	ビットサイズ	Description
ID	3 2	「VID0」 -0x56494430、「AUD0」 -0x41554430
データ開始	1	データの1セット中の最初の packets
ハール	1	変換、ハール/2 - 1 0
プロファイル Y	4	圧縮プロファイル
プロファイル U	4	
プロファイル V	4	
未使用	2	
パケットサイズ	1 6	バイトで表したデータパケットの大きさ
PTS	3 2	1 0 μ s 単位で表したデータパケット
データ	{パケットサイズ} * 8	のタイムスタンプ 未処理データ

表 5：多重化オーディオ／ビデオストリームのためのパケットフォーマット。バイト順序は「リトル・エンディアン」（インテル）フォーマットである。

【0 2 6 2】

得られたビットストリームをバッファメモリに入れられ、そして必要なヘッダをつけた状態でイーサネットペイロードに変換される。プロセス全体を図 2 5 に示す。

【0 2 6 3】

図 2 5 にはっきりと示されていないアイテムは、さらに別のリアルタイムプロトコル（RTP）ヘッダで、これはトランスポート層ヘッダとトランスポート層ペイロードとの間で搬送されるものである。

【0 2 6 4】

「ビデオのためのデータ開始」はフレーム開始を示し、1つのフレームに必要な可変の数のパケットにおける最初の packets を示している。（1つのフレームに必要な packets の数は、画像の性質、圧縮プロファイルおよび一時的な圧縮の影響により変わる。）もとの画像ブロックと packets 境界との間では位置合わせがなされていないため、最後の packets は部分的にしか満たされていない場合がある。このような場合、packets にはゼロをつめこんで packets サイズを作り上げる。

【0 2 6 5】

[ネットワークローディング]

上述のことから、圧縮画像を保持しているネットワークに提示されている「ローディング」は画像の性質によって可変であることが明らかである。事実上、どのような所定の画像解像度および圧縮プロファイルの場合でも、平均ビット速度はまったく一定のままであ

ることがわかっている。これは、どのような現実のアプリケーションにおいても画像のために十分なネットワーク容量が確実にあるようにすることは簡単であることを意味しており、このことは特に複数の画像が保持されている場合に言えることである、なぜなら統計学的に全体のビット速度は非常に狭い制限内で一定のままとなるであろうからである。

【0266】

全体のシステムの「プログラム可能な」側面は個々のフレームベースで適用できる点に留意すべきである。これは、圧縮パフォーマンスが変更できるため平均ビット速度も「進行中に」変更できることを意味する。その結果、システムが一定のビット速度を提供することはないが、予測可能なパフォーマンス、および必要が生じると直ちにビット速度を変更する能力は提供する。

【0267】

[復号化オプション]

好ましいシステムの目的とする原則は、符号化および復号化プロセスが対称的であるということである。従って、復号化プロセスを普通に実行すると図12に示すものの逆になることになり、類似のハードウェア構造に基づかせることができる。要するに、

(a) 入ってくるデータの流れは「脱パケット化」、すなわち、UDPフォーマットとエラー訂正とに関連しているすべてのオーバーヘッドデータが取り除かれ、コード化されたYUVブロックデータが回復される。

(b) 圧縮プロファイル情報をまず最初に受信することにより、その後この情報を圧縮化ブロックデータに適用することが可能であり、そしてこれにより完全なLKCS情報を回復することが可能となる。多くの場合、完全な「ツリー」はコード化された状態の単一のビットだけによって表される場合があるが、復号化を行う場合、すべての「かくれた」値が復元される点に留意すること。

(c) LKCS情報を使って完全なセットのウェーブレット係数を作成する。

(d) このデータは逆変換され、数式13および15を使ってLおよびH値を回復する。符号化プロセスと同じく、これを行う場合には、レベル1の「垂直」に到達するまで何度も逆変換エンジンを経る必要がある。符号化プロセスの場合のように、「横列および縦列」制御を使えば簡素な1次元逆変換エンジンを使える。

(e) 回復されたデータはその後、水平なレベル1次元でのみ作用する第2の逆変換エンジンにかけられ、個々の画素データを回復する。

(f) 画素データ（個々では2つの画素ごとに16ビットに戻っている）がYUVから8ビットRGBに変換し戻される。

【0268】

[パケット損失の対処]

現実のネットワークでは、データパケットが無くなる重大な機会がある。例えば、ITUが推奨するY.1541（IPベースのサービスのためのネットワークパフォーマンス目標）はIPネットワークにおける 1×10^{-3} のIPLR（IPパケット損失比）を想定している。明らかに、これは受信した画像に対して壊滅的な影響を持ち得る場合がある。しかしながら、複雑な前向きエラー訂正（帯域および待ち時間の両方を増加させると思われる）により生じるであろうさらなるオーバーヘッドを回避するために、好ましいシステムはそれ自身の画像ブロックフォーマット（表4）を使うことで、パケット損失の結果生じる汚染されたデータを破棄する方法を提供する。

【0269】

データストリームは連続的であるが、同期ワードは一連の16ビット値の1として簡単に区別できる。図26は、YUVデータの2つの連続的なブロックを示しており、すべて良好であれば、それぞれのブロックはそれ自身の同期ワードを持つが、ブロックの末端には次のブロックの同期ワードがあることがわかる。

【0270】

IPネットワークでは、それぞれのIPパケットはCRCチェックサムを使って検証される。IPパケットは、無効であれば廃棄される。図面に示される画像ビットストリーム

の影響は、ある1つのセクションが切り離され、典型的には、無関係のブロックデータの2つのロットが1つにされる可能性があるというものである。

【0271】

ブロックの長さは可変であるが、画像データの「ツリー」の性質は、復号器には画像再構築を完成させるのに十分なデータをいつ受信したか「わかって」おり、従って、いつ次の同期ワードに出会うかを予測するというものである。かかる機能を利用してブロックデータを、ディスプレイメモリに送られる前に検証する。

【0272】

そのメカニズムは、Y U B B L O C K mの形態のブロックデータはそれ自身のS Y N C W O R D mおよびそのすぐ隣のS Y N C W O R D n'によって検証されるというものである。復号器内の「Y U Vブロック送付モジュール」は、復号化されているY U Vブロックを保存しており、もし復号化されているY U Vブロックの末端に後続のS Y N Cワードが存在していれば、モジュールはY U Vブロックをディスプレイメモリへと通過させる。もし存在していなければ、復号化されているY U Vブロックは破棄される。

【0273】

画像フレーム内の最後のブロックの特別なケースは、通常は次の同期ワードに出会うことはないのであるが、フレームの最後にさらに別の同期ワードを挿入することによって対処される。これにより、有効なY U Vブロックだけをディスプレイへと確実に通過させることができる。この方法が機能するのは、Y U Vブロックが絶対的画像データを含んでおり、履歴または順方向データのいずれにも依存しないからである。万が一、ある1つのブロックが破棄される場合は、ディスプレイシステムは以前の「良好な」Y U V画像ブロックを示し続ける。典型的なディスプレイフレーム速度(24-60 Hz)で作動するシステムでは、パケットの損失により生じるランダムエラーは、実際には注目に値するものではない。

【0274】

[ソフトウェア復号]

好ましいシステムの目的とするアプリケーションは、ほとんどの場合、ハードウェア復号を使って決定的なパフォーマンスを確実なものにするというものであろう。しかしながら、これまでの説明から、符号化プロセスの「出力」は1つの画像またはセットになった画像のビットストリームであることは明らかである。従って、理論上は、ビットストリーム構文を知っていれば誰でもソフトウェア方法だけを使ってそれを復号化する手段を考案できるであろうと思われる。

【0275】

特定の市場ニーズを満たすように「ソフトウェア復号」生成物を開発できると思われる。かかるニーズは(例えば、低い解像度で画像を受信するまたは部分的な画像を検討するために)より低レベルのパフォーマンスが受け入れられやすい場合に見込まれるものである。

【0276】

[圧縮された画像フォーマットの利点]

これまでの説明ではc o d e c(コード器-復号器)を説明してきた。これは、符号化および復号化プロセスで対称的にディスプレイを行う、すなわち決定的であり、(ただし一時的な圧縮プロセスにおけるビット速度は除く)また最小限の待ち時間しか招かないというものであった。明らかに、特に復号ステージにおいてさらに別の画像処理機能を導入する可能性もある。コード化されたデータは基礎をなす画像の非常に経済的な「短縮化された」記述を表し、これは画素ドメイン内で計算上集約的であると思われる動作は、コード化されたブロックドメイン内の最小の資源で実行できることを意味する。このことは特に、単一のc o d e cユニットを使って複数の画像を同時に処理する場合にいえることである(例えば、業界で標準的なF P G Aを使って実現されたc o d e cは同時に8つの標準的なビデオ画像を処理できる)。

【0277】

いくつかの可能性として次のものが挙げられる。

(a) 異なる画像ストリームから必要なブロックだけを選択することによって複数の画像表示を組み立て、並べ替えて必要なディスプレイフォーマットを生成する。

(b) (変換のすべてのレベルを復号するわけではないと選択することで) 異なる画像忠実度を選択する。

(c) ディスプレイノードの能力に合致するように画像ブロックを選択する (例えば、画像ストリームは 1600×1200 の等価物を持ち得るが、ディスプレイは 800×600 だけしか表示できない)。このことはサイズ変更を意味するのではない点に注意すること。それはまた別の主題である。

【0278】

重要な理論上のポイントは、コード化ブロックレベルで行われるどのような処理または並べ替えもリアルタイムよりも高速で行われるものとして考えることができるということである。例えば、もし画像が $20:1$ で圧縮されていれば、どのような処理でも、対応する処理が画素レベルで生じた場合の 20 分の 1 で起こることになる。

【0279】

[要約]

ウェーブレット変換に基づく画像圧縮の好ましい実施の有利な特徴のいくつかの要約は以下のとおりである。

(a) RGBからYUVへの変換とウェーブレット変換との組み合わせにより得られる結果に精度維持の特性を使うことで、ビット成長のない全体的に可逆な損失なしの変換が提供される。

(b) 平行パイプラインアーキテクチャに基づく高速スケーラブル変換エンジン。変換の選択がプログラム可能なことからグラフィックおよび移動画像用途のどちらについても最適な結果が得られる。変換処理は決定的である、すなわち正確なサイクル時間で実行され画像内容から全く独立している。待ち時間を最小限にできる。

(c) 全フレーム変換の結果を実現して画像を移動させつつ実際にはブロックドメイン内ですべての処理を実行する方法 (「ブロック外」変換データの使用)。

(d) 効率的な圧縮を導くことができるウェーブレット変換の特徴を利用できるように設計された新規の「LKCS」コード化アレンジメント。

(e) 損失のない、視覚的に損失のない、また、高い空間的圧縮効率で高い圧縮を提供できるプログラム可能な圧縮プロファイル。たとえば最も遅いビットストリーム速度に対する最も速いビットストリーム速度の割合は $1000:1$ である。

(f) ウェーブレット署名のアプリケーションに基づく新規のプログラム可能な一時的圧縮体系。絶対的なコード化で、画像履歴または順方向予測を必要としない。画像待ち時間の延長がない。参照フレームを使って送信エラーの影響を排除する。

(g) ビットストリームを自己記述することで、コード化画像ブロックデータを保持する。

(h) システム出力はコネクションレスネットワークオペレーションのために構成されており、スケーラブルな複数の画像ネットワークの構成を規定している。ネットワーク送信エラーに対する高い免疫性。

(i) 符号化ビットストリームの性質を利用した、IPパケット損失を検出する新規の方法。

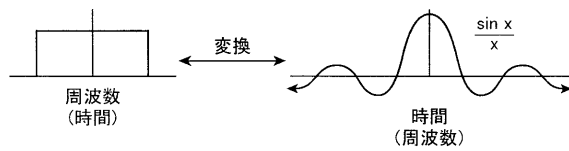
10

20

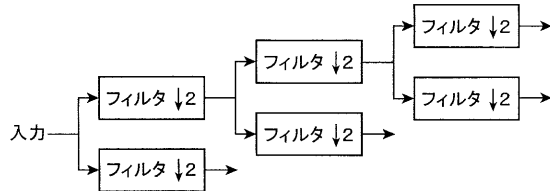
30

40

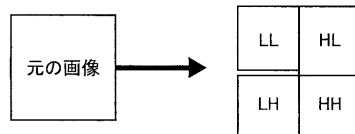
【図 1】



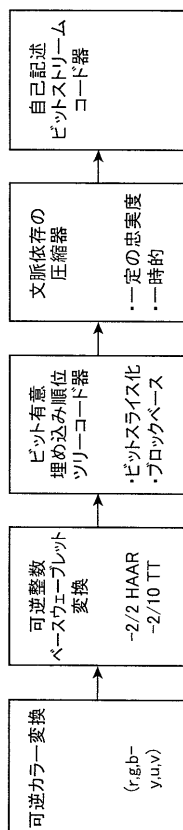
【図 2】



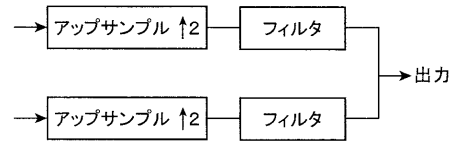
【図 3】



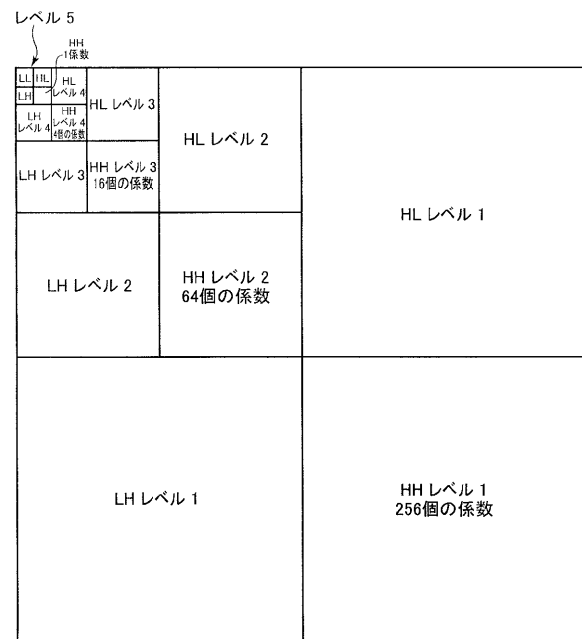
【図 5】



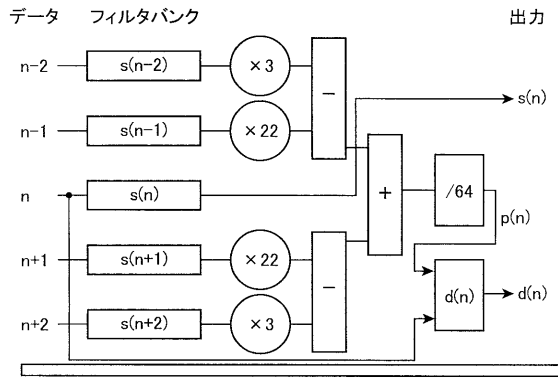
【図 4】



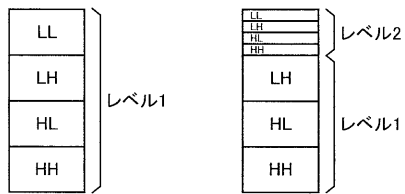
【図 6】



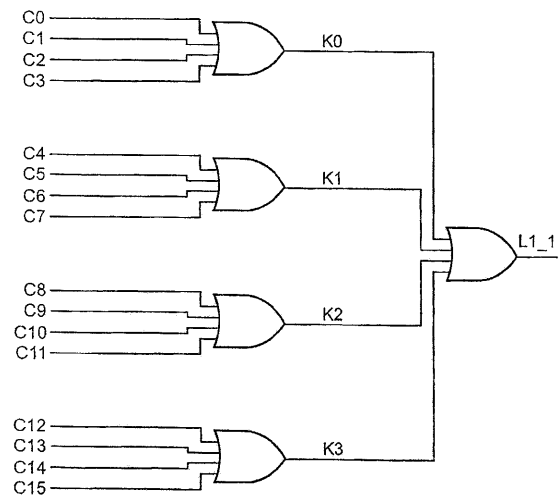
【図 1 3】



【図 1 4】

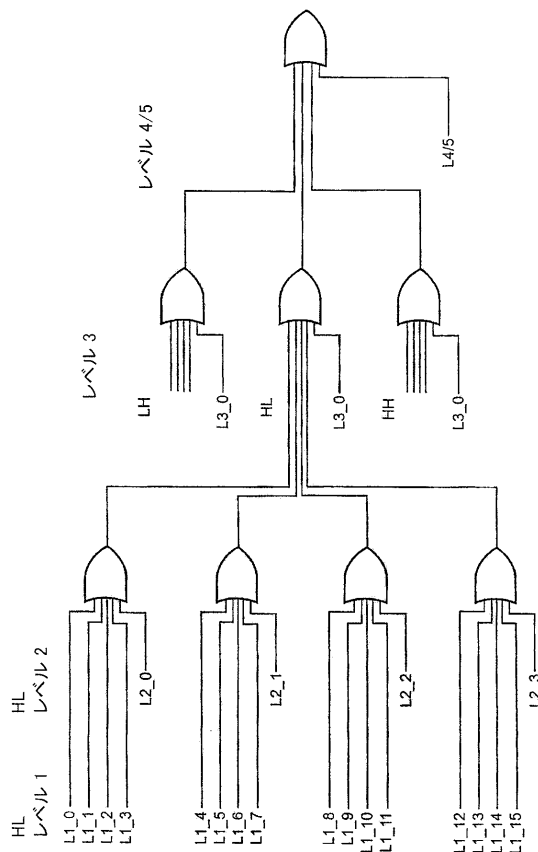


【図 1 5】

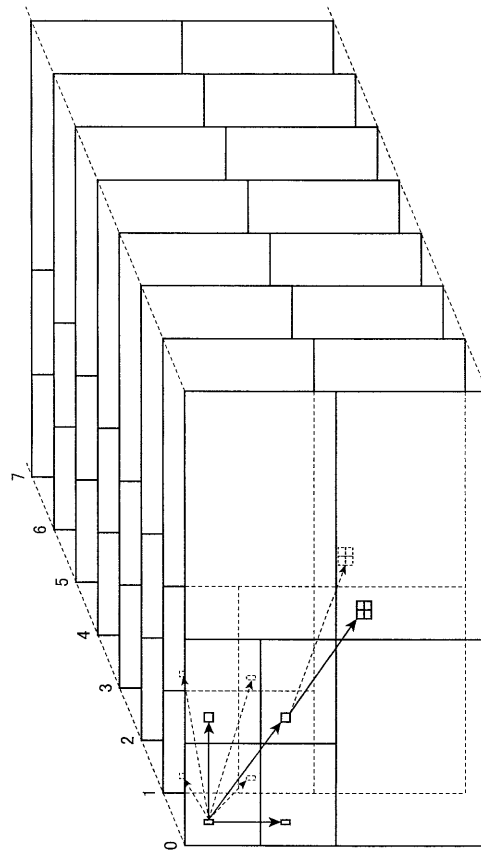


C0 TO C15 = 係数
K0 TO K3 = Kタイプ
L1_0 = Lタイプ

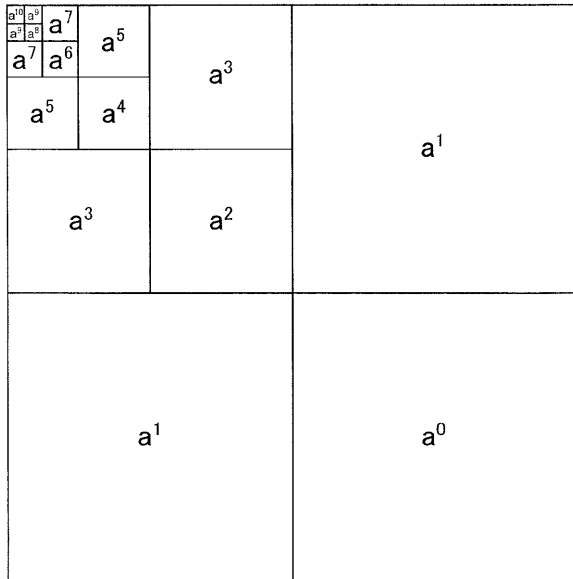
【図 1 6】



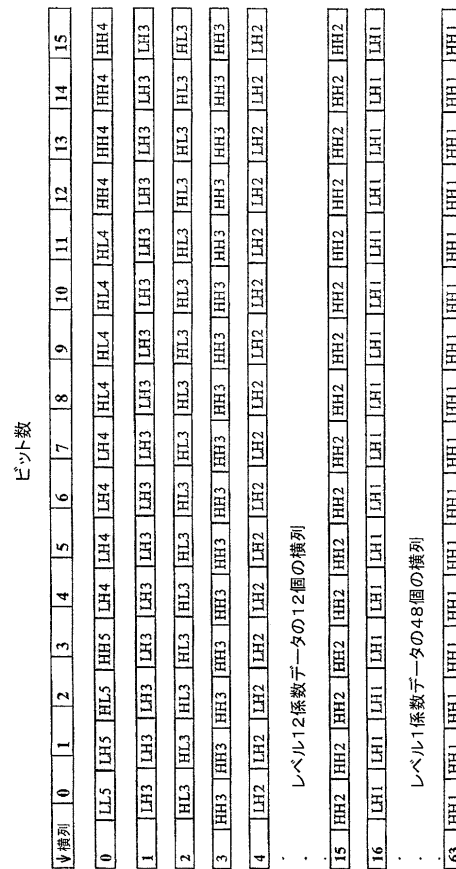
【図 1 7】



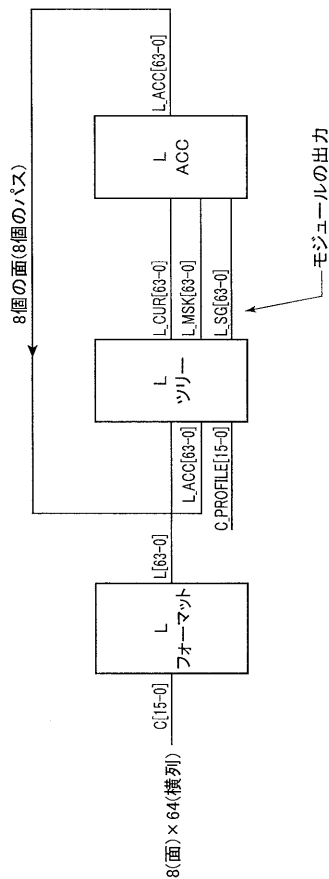
【図 18】



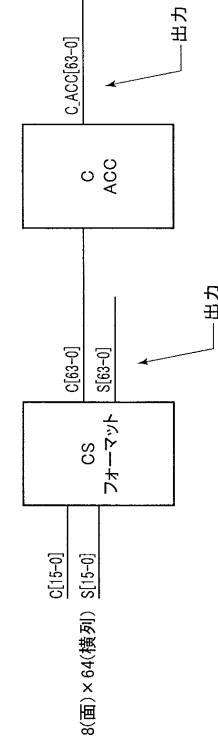
【図 19】



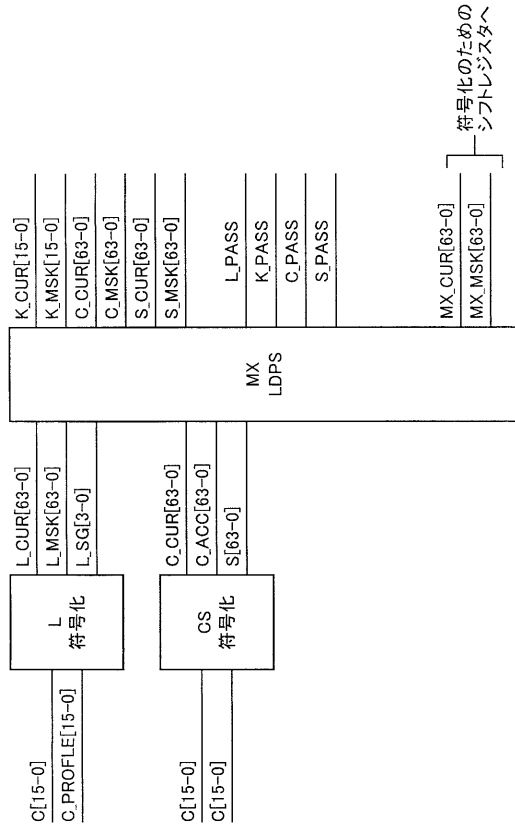
【図 20】



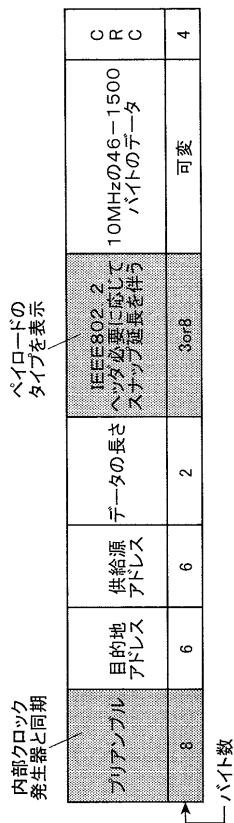
【図 21】



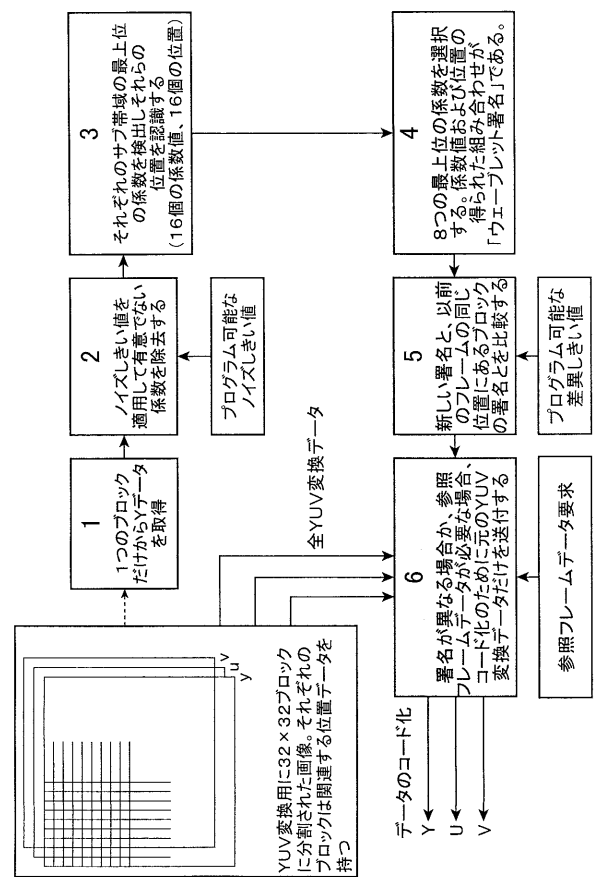
【図 2 2】



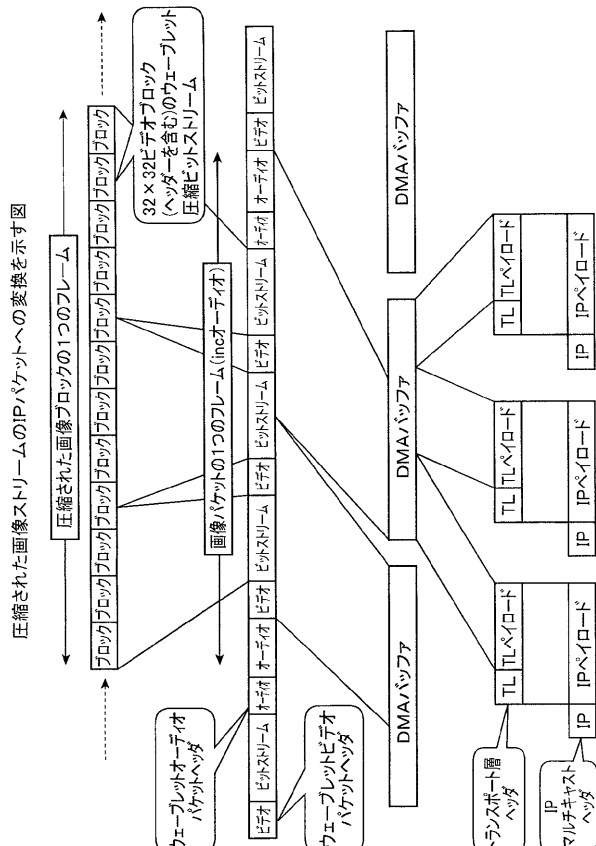
【図 2 4】



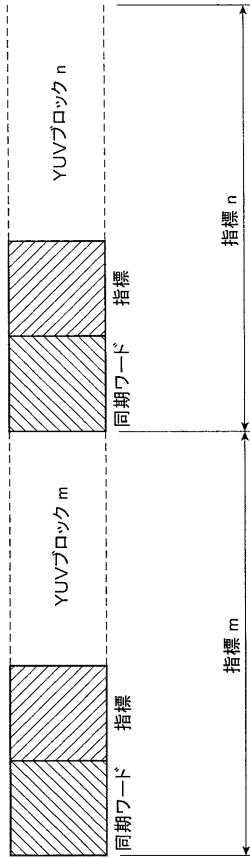
【図 2 3】



【図 2 5】



【図 26】



フロントページの続き

Fターム(参考) 5C178 AC10 AC16 BC52 BC63 CC60 EC62