(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5420254号
(P5420254)
(45)発行日　平成26年2月19日(2014.2.19)　(24)登録日　平成25年11月29日(2013.11.29)

(51)Int.Cl.　　F I
H04N 19/50 (2014.01)　　H04N 7/137 Z
H04N 19/60 (2014.01)　　H04N 7/133 Z

請求項の数 17　(全 50 頁)

(21)出願番号　特願2008-556422 (P2008-556422)
(86)(22)出願日　平成19年2月21日(2007.2.21)
(65)公表番号　特表2009-527991 (P2009-527991A)
(43)公表日　平成21年7月30日(2009.7.30)
(86)国際出願番号　PCT/US2007/004638
(87)国際公開番号　WO2007/100616
(87)国際公開日　平成19年9月7日(2007.9.7)
審査請求日　平成22年2月17日(2010.2.17)
(31)優先権主張番号　11/276,336
(32)優先日　平成18年2月24日(2006.2.24)
(33)優先権主張国　米国(US)
(31)優先権主張番号　11/673,423
(32)優先日　平成19年2月9日(2007.2.9)
(33)優先権主張国　米国(US)

(73)特許権者　500046438
マイクロソフト　コーポレーション
アメリカ合衆国　ワシントン州　９８０５２－６３９９　レッドモンド　ワン　マイクロソフト　ウェイ
(74)代理人　100140109
弁理士　小野　新次郎
(74)代理人　100089705
弁理士　社本　一夫
(74)代理人　100075270
弁理士　小林　泰
(74)代理人　100080137
弁理士　千葉　昭男
(74)代理人　100096013
弁理士　富田　博行

最終頁に続く

(54)【発明の名称】加速ビデオ符号化

(57)【特許請求の範囲】
【請求項１】
少なくとも部分的にコンピューティングデバイスの１つ又は複数のプロセッサによって実施される方法であって、
前記方法は、
前記コンピューティングデバイスの前記１つ又は複数のプロセッサによって実現されたビデオ符号化加速モジュールによって、ビデオエンコーダから１つ又は複数のクエリを受け取って加速ハードウェアの実施仕様を特定するステップ、
前記１つ又は複数のクエリの受け取りに応答して、前記ビデオ符号化加速モジュールが、
前記加速ハードウェアとインターフェースして前記実施仕様を取得するステップ、
前記実施仕様の受け取りに応答して、前記ビデオエンコーダへ前記実施仕様を通信するステップ
を含み、
前記実施仕様は、前記ビデオエンコーダが、ランタイム中に、
（ａ）前記ビデオエンコーダの符号化アーキテクチャと前記実施仕様とに基づいて、前記ビデオエンコーダに関連付けられるソフトウェア符号化オペレーションの速度及び品質のうちの１つ又は複数を、１つ又は複数のサポートされる符号化パイプラインの構成及び機能から成る特定の符号化パイプラインの実施によって増加させることができるか否かを判断すること、及び

（ｂ）前記ビデオ符号化加速モジュールとインターフェースすることによって前記特定の符号化パイプラインを実施すること、
を可能にし、
前記方法は、さらに、
前記ビデオエンコーダによって、前記特定の符号化パイプラインを実施するエンコーダオブジェクトを作成するための構成パラメータを特定するステップ、及び
前記ビデオエンコーダによって、前記構成パラメータに基づき前記エンコーダオブジェクトを作成するステップ
を含み、
復号されたソースビデオデータを符号化するための前記エンコーダオブジェクトが、前記特定の符号化パイプラインを使用する、
方法。

【請求項２】
請求項１に記載の方法において、前記ソフトウェア符号化オペレーションは、動き推定オペレーション、残差計算オペレーション、動き補償オペレーション、及び変換オペレーションのうちの１つ又は複数を含む、方法。

【請求項３】
請求項１に記載の方法において、前記ソフトウェア符号化オペレーションは、ノイズ低減オペレーション、画像安定化オペレーション、エッジ検出オペレーション、鮮鋭化オペレーション、及びフレームレート変換オペレーションのうちの１つ又は複数を含む、方法。

【請求項４】
請求項１に記載の方法において、前記１つ又は複数のクエリは、機能取得クエリを含み、前記受け取られた実施仕様は、前記１つ又は複数のサポートされる符号化パイプライン構成に関連付けられる情報を含む、方法。

【請求項５】
請求項１に記載の方法において、前記１つ又は複数のクエリは、距離メトリック取得クエリを含み、前記受け取られた実施仕様は、動き推定オペレーションに関して前記ビデオ符号化加速ハードウェアによってサポートされる１つ又は複数の検索メトリックの記述を含む、方法。

【請求項６】
請求項１に記載の方法において、前記１つ又は複数のクエリは、検索プロファイル取得クエリを含み、前記受け取られた実施仕様は、前記ビデオ符号化加速ハードウェアによってサポートされる１つ又は複数の検索プロファイルの記述を含み、該１つ又は複数の検索プロファイルは、前記ビデオエンコーダが、ビデオ符号化処理時間とビデオ符号化品質メトリックとの間の実施仕様トレードオフを、異なる検索プロファイルに関する相対的な動きベクトルの品質及び相対的な処理時間に基づいて、評価することを可能にする、方法。

【請求項７】
請求項１に記載の方法において、前記１つ又は複数のクエリは、動き推定機能取得クエリを含み、前記受け取られた実施仕様は、サポートされる最大画像サイズ、サポートされる最大検索ウィンドウサイズ、及び加速ハードウェアが可変マクロブロックサイズをサポートするか否かの表示のうちの１つ又は複数を示すデータを含む、方法。

【請求項８】
請求項１に記載の方法において、前記構成パラメータは、前記特定の符号化パイプライン、符号化されたビデオの出力フォーマット、前記特定の符号化パイプラインとの関連付けのためのＩ／Ｏデータストリームの個数、前記Ｉ／Ｏデータストリームのデータバッファの提案される個数、並びに利用可能な資源に基づくデバイスドライバ指定のキューサイズのうちの１つ又は複数を指定する、方法。

【請求項９】
プロセッサによって実行可能な以下のステップを実行するためのコンピュータプログラ

ム命令を記憶したコンピュータ可読ストレージ媒体であって、該ステップは、
ビデオエンコーダプログラムモジュールによって、加速ハードウェアによってサポートされるビデオ符号化パイプラインの構成及び機能のうちの１つ又は複数の機能を特定するための１つ又は複数の要求をビデオ符号化加速モジュールへ通信するステップ、
前記ビデオ符号化加速モジュールから前記機能を受け取ったことに応答して、前記ビデオエンコーダが、
前記ビデオエンコーダプログラムモジュールの符号化アーキテクチャ、及び前記機能に基づいて、前記加速ハードウェアによって実施される場合に速度及び品質のうちの１つ又は複数で利益を受ける前記ビデオエンコーダに関連付けられる１つ又は複数のビデオ符号化オペレーションを特定するステップ、
前記ビデオエンコーダによって、前記１つ又は複数のビデオ符号化オペレーションを前記加速ハードウェアを介して実施し、あらゆる残りのビデオ符号化オペレーションがソフトウェアで実施されるようにするためのカスタマイズされたビデオ符号化パイプラインを作成するように前記ビデオ符号化加速モジュールに要求するステップ、
前記ビデオエンコーダプログラムモジュールによって、前記カスタマイズされたビデオ符号化パイプラインを実施するエンコーダオブジェクトを作成するための構成パラメータを特定するステップ、
前記ビデオエンコーダプログラムモジュールによって、前記構成パラメータに基づき前記エンコーダオブジェクトを作成するステップ、
前記エンコーダオブジェクトが、前記カスタマイズされたビデオ符号化パイプラインを使用するステップ、
からなる、コンピュータ可読ストレージ媒体。
【請求項１０】
請求項９に記載のコンピュータ可読ストレージ媒体において、前記１つ又は複数のビデオ符号化オペレーションは、動き推定オペレーション、残差計算オペレーション、動き補償オペレーション、及び変換オペレーションのうちの１つ又は複数を含む、コンピュータ可読ストレージ媒体。
【請求項１１】
請求項９に記載のコンピュータ可読ストレージ媒体において、前記１つ又は複数のビデオ符号化オペレーションは、ノイズ低減オペレーション、画像安定化オペレーション、エッジ検出オペレーション、鮮鋭化オペレーション、及びフレームレート変換オペレーションのうちの１つ又は複数を含む、コンピュータ可読ストレージ媒体。
【請求項１２】
請求項９に記載のコンピュータ可読ストレージ媒体において、前記要求するステップのためのコンピュータプログラム命令は、システムメモリと前記加速ハードウェアのメモリとの間のデータフローが最小にされるように、前記ビデオ符号化加速モジュールに、前記カスタマイズされたビデオ符号化パイプラインを作成するように指示するための命令をさらに含み、前記プロセッサが処理を実行しない場合に前記加速ハードウェアのメモリから前記システムメモリへのデータのコピーを行わないことによって、前記データフローが最小にされる、コンピュータ可読ストレージ媒体。
【請求項１３】
請求項９に記載のコンピュータ可読ストレージ媒体であって、プロセッサによって実行可能な以下のステップを実行するためのコンピュータプログラム命令を更に記憶し、該ステップは、
前記ビデオエンコーダによって、符号化された又は復号されたソースビデオデータを受け取るステップ、
前記受け取られたソースビデオデータが符号化されている場合に、前記ビデオエンコーダによって、該ソースビデオデータを少なくとも部分的に復号し、前記ビデオ符号化加速モジュールによって作成される符号化オブジェクトによって符号化するための復号されたソースビデオデータを生成するステップであって、該符号化オブジェクトは、前記カスタ

マイズされたビデオ符号化パイプラインを実施する、ステップ
からなるコンピュータ可読ストレージ媒体。
【請求項１４】
請求項９に記載のコンピュータ可読ストレージ媒体において、前記コンピュータプログラム命令は、前記カスタマイズされたビデオ符号化パイプラインを使用して、復号されたソースビデオデータを符号化するための命令をさらに含む、請求項１１に記載のコンピュータ可読ストレージ媒体。
【請求項１５】
ビデオ符号化加速モジュールを備えるコンピューティングデバイスであって、該ビデオ符号化加速モジュールは、
加速ハードウェアの実施仕様を特定するように前記ビデオ符号化加速モジュールに要求する１つ又は複数のクエリをビデオエンコーダから受け取り、前記実施仕様は、前記ビデオエンコーダが、（ａ）前記ビデオエンコーダの符号化アーキテクチャと前記実施仕様とに基づいて、該ビデオエンコーダに関連付けられるソフトウェア符号化オペレーションの速度及び品質のうちの１つ又は複数を、１つ又は複数のサポートされる符号化パイプラインの構成及び機能から成る特定の符号化パイプラインの実施によって増加させることができるか否かを判断すること、並びに、（ｂ）前記ビデオ符号化加速モジュールを介して前記特定の符号化パイプラインを実施して復号されたソースビデオデータを符号化することを可能にするためのものであり、
前記加速ハードウェアに問い合わせて前記実施仕様を取得し、
前記加速ハードウェアから受け取った前記実施仕様を前記ビデオエンコーダへ通信し、
前記特定の符号化パイプラインを実施するエンコーダオブジェクトを作成するためのエンコーダオブジェクト作成要求を前記ビデオエンコーダから受け取り、
前記特定の符号化パイプラインに関連付けられるオペレーションを前記加速ハードウェアで実施するための１つ又は複数の実行要求を前記ビデオエンコーダから受け取り、
前記１つ又は複数の実行要求に関連付けられる情報を前記加速ハードウェアへ転送して前記復号されたソースビデオデータを符号化する、
コンピューティングデバイス。
【請求項１６】
請求項１５に記載のコンピューティングデバイスにおいて、前記ソフトウェア符号化オペレーションは、動き推定オペレーション、残差計算オペレーション、動き補償オペレーション、及び変換オペレーションのうちの１つ又は複数を含む、コンピューティングデバイス。
【請求項１７】
請求項１５に記載のコンピューティングデバイスにおいて、前記ソフトウェア符号化オペレーションは、ノイズ低減オペレーション、画像安定化オペレーション、エッジ検出オペレーション、鮮鋭化オペレーション、及びフレームレート変換オペレーションのうちの１つ又は複数を含む、コンピューティングデバイス。
【発明の詳細な説明】
【技術分野】
【０００１】
関連出願
本願は、２００６年２月２４日に出願された「Accelerated Video Encoding」という発明の名称の同時係属中の米国特許出願第１１／２７６，３３６号の一部継続出願である。本記載によって、この米国特許出願は、参照により援用される。
【背景技術】
【０００２】
マルチメディアコンテンツの生成オペレーション及び配布オペレーションは、通常、ビデオ符号化を含む。ビデオ符号化プロセスは、通常、データ及び計算量が非常に多い。そ

の結果、ビデオ符号化プロセスは、非常に多くの時間を要する可能性がある。たとえば、ソフトウェアエンコーダが高品質高品位映画を符号化するには、数十時間を要する場合がある。ビデオ符号化プロセスの品質及び速度は、マルチメディアコンテンツの生成及び配布のパイプラインが成功するための重要なファクタであるので、高品質ビデオコンテンツを符号化することができる速度を増加させるためのシステム及び技法が有用となる。

【特許文献１】米国特許第６１０１２７６号公報
【特許文献２】米国特許第６２５２９０５号公報
【特許文献３】特表第２００４－５０４７８０号公報（国際公開第ＷＯ０２／０７４４６号）
【特許文献４】米国特許第５９９０９５８号公報

【発明の開示】

【０００３】

この概要は、詳細な説明でさらに後述する概念のうちの選択したものを簡略化した形で紹介するために設けられている。この概要は、特許を請求する主題の重要な特徴又は本質的な特徴を特定することを目的としておらず、特許を請求する主題の範囲を決定することを助けるものとして使用されることも目的としていない。

【０００４】

上記に鑑み、ビデオ符号化の速度及び品質のうちの１つ又は複数を増加させるためのビデオ符号化加速サービスが記載される。当該サービスは、任意のビデオエンコーダコンピュータプログラムアプリケーションと任意のビデオ加速ハードウェアとの間の仲介手段として動作する。当該サービスは、ビデオ加速ハードウェアの実施仕様を特定するために１つ又は複数のクエリをビデオエンコーダから受け取る。当該サービスは、ビデオ加速ハードウェアとインターフェースして、実施仕様を取得する。当該サービスは、ビデオエンコーダへ実施仕様を通信する。実施仕様によって、ビデオエンコーダは、（ａ）ビデオエンコーダに関連付けられるソフトウェア符号化オペレーションの速度及び品質のうちの１つ又は複数を、１つ又は複数のサポートされる符号化パイプラインの構成及び機能から成る１つのパイプラインの実施によって増加させることができるか否かを判断すること、並びに、（ｂ）当該サービスとインターフェースすることによってそのパイプラインを実施することが可能になる。

【発明を実施するための最良の形態】

【０００５】

概観

加速ビデオ符号化のシステム及び方法は、ビデオ符号化加速サービスを提供する。このサービスによって、任意のビデオ符号化アプリケーションが、デバイスに依存しない方法で、任意のビデオ加速ハードウェアとインターフェースして、実質的に最適なビデオ符号化パイプラインを定義して実施することが可能になる。これを実現するために、このサービスは、ビデオ加速（ＶＡ）アプリケーションプログラムインターフェース（ＡＰＩ）を公開する。これらのＡＰＩは、ビデオ符号化プロセスのモデルをカプセル化する。符号化パイプラインを定義するために、ビデオエンコーダアプリケーションは、ＶＡ　ＡＰＩを使用して、利用可能なビデオ（グラフィックス）加速ハードウェアの実施仕様（たとえば、機能等）を問い合わせる。ビデオエンコーダは、ハードウェアで加速されることで利益（たとえば、速度の利益及び／又は品質の利益）を得ることができるあらゆる符号化オペレーションを特定するために、そのアプリケーションの特定のビデオ符号化アーキテクチャ（ソフトウェア実装されたもの）を考慮してこれらの仕様を評価する。このようなオペレーションには、たとえば、動き推定オペレーション、変換オペレーション、量子化オペレーション、及び、動き補償、逆変換、逆量子化等の逆のオペレーションが含まれる。また、ＡＰＩによって、ビデオエンコーダは、ホストコンピューティングデバイス及び加速ハードウェアに関連付けられているバス及びプロセッサにわたるデータフロー移行（data flow transition）を実質的に最小にする符号化パイプラインを設計することも可能になり、それによって、符号化速度をさらに増加させることが可能になる。また、ＡＰＩによ

って、加速ハードウェアは、ローカルキャッシュを改善するようにデータのロケーションに影響を与えることも可能になる（たとえば、ビデオ加速ハードウェアは、ビデオハードウェアにローカルなメモリでより効率的に機能することができる）。

【０００６】

これらの評価に基づいて、ビデオエンコーダは、或る個数の符号化オペレーションをソフトウェアで実行し、或る個数の符号化オペレーション（すなわち、ハードウェアで加速されることで利益を得ることができるオペレーションの少なくともサブセット）を加速ハードウェアを使用して実行する、カスタマイズされたビデオ符号化パイプラインを設計する。エンコーダアプリケーションは、次に、ＡＰＩを使用してパイプラインを作成し、ビデオコンテンツを符号化する。このカスタマイズされたパイプラインは、一定の符号化オペレーションが加速され、ホストと加速ハードウェアとの間のデータ移行が最小にされるので、完全にソフトウェア実装されたパイプラインと比較して実質的に最適化されている。加えて、符号化プロセスの特定の態様を加速化すると共にデータ移行を最小にすることによって解放された処理時間によって、ホストプロセッサ（複数可）は、より高品質な符号化オペレーションを解放された処理サイクルで実行することが可能になる。また、ＡＰＩは、計算資源の使用量を最大にすることができるように、コンポーネントが並列に動作することを可能にするようにも設計される。

【０００７】

次に、加速ビデオ符号化のシステム及び方法のこれらの態様及び他の態様をより詳細に説明する。

【０００８】

１つの例示的なシステム

必須ではないが、加速ビデオ符号化のシステム及び方法は、コンピュータ実行可能命令（プログラムモジュール）が、パーソナルコンピュータやグラフィックス（ビデオ）符号化加速ハードウェア等のコンピューティングデバイスによって実行される一般的な文脈で説明される。プログラムモジュールは、一般に、特定のタスクを実行するか又は特定の抽象データタイプを実施するルーチン、プログラム、オブジェクト、コンポーネント、データ構造等を含む。

【０００９】

図１は、一実施形態による加速ビデオ符号化の１つの例示的なシステム１００を示している。システム１００は、ホストコンピューティングデバイス１０２を含む。ホストコンピューティングデバイス１０２は、パーソナルコンピュータ、ラップトップ、サーバ、ハンドヘルドコンピューティングデバイス、モバイルコンピューティングデバイス等の任意のタイプのコンピューティングデバイスを表す。ホストコンピューティングデバイス１０２は、バス１０３を横断してシステムメモリ１０６に結合される１つ又は複数の処理ユニット１０４を含む。システムメモリ１０６は、コンピュータプログラムモジュール（「プログラムモジュール」）１０８及びプログラムデータ１１０を含む。プロセッサ１０４は、プログラムモジュール１０８のそれぞれからコンピュータプログラム命令をフェッチして実行する。プログラムモジュール１０８は、ビデオコンテンツを処理するビデオ処理モジュール１１２、及び、オペレーティングシステム、（たとえば、ビデオ符号化加速ハードウェアにインターフェースするため等の）デバイスドライバ等の他のプログラムモジュール１１４を含む。ビデオ処理モジュール１１２は、たとえば、ビデオエンコーダ１１６、ビデオ符号化加速サービス１１８、及び他の処理モジュール１２０を含む。他の処理モジュール１２０は、たとえば、ビデオデコーダ、ビデオフィルタ（複数可）、ビデオレンダラ等である。

【００１０】

この実施態様では、ビデオエンコーダ１１６は、任意のビデオエンコーダである。これは、ビデオエンコーダ１１６によって実施され且つ／又は利用される特定のアーキテクチャ、オペレーション、データフォーマット等が任意であることを意味する。たとえば、ビデオエンコーダ１１６は、ＯＥＭ等のサードパーティによって配布される場合がある。加

えて、図１は、ビデオ符号化加速サービス１１８が「他のプログラムモジュール」１１４のオペレーティングシステム部分から独立していることを示しているが、一実施態様では、ビデオ符号化加速サービス１１８は、オペレーティングシステムの一部である。

【００１１】

ビデオ処理モジュール１１２は、圧縮された又は未圧縮の入力ビデオデータ１２２を受け取る。入力ビデオデータ１２２が圧縮（既に符号化）されているとき、ビデオ処理モジュール１１２は、入力ビデオデータ１２２を復号して、復号されたソースビデオデータを生成する。このような復号オペレーションは、デコーダモジュールによって実行される。別の実施態様では、符号化プロセスをさらに援助するために、部分的に復号されたデータを保持することもできる。例示的な説明のために、このようなデコーダモジュールは、「他のビデオ処理モジュール」１２０のそれぞれの部分として示される。したがって、復号されたソースビデオデータは、復号された状態で受け取られた入力ビデオデータ１２２によって表されるか、又は、符号化された状態で受け取られた入力ビデオデータ１２２の復号の結果で表される。復号されたソースビデオデータは、「他のプログラムデータ」１２４のそれぞれの部分として示される。

【００１２】

復号されたソースビデオデータを符号化して符号化されたビデオデータ１２６にするのに使用することができる、カスタマイズされたビデオ符号化パイプラインを設計して実施するために、ビデオエンコーダ１１６は、ビデオ加速（ＶＡ）ＡＰＩ１２８を介してビデオ符号化加速サービス１１８とインターフェースする。ＶＡ　ＡＰＩ１２８の複数の可能な実施態様のうちの例示的な一実施態様は、付録に記載されている。符号化パイプラインを定義するために、ビデオエンコーダアプリケーションは、ＶＰ　ＡＰＩ１２８（付録の§３．４のIVideoEncoderServiceを参照されたい）のそれぞれを使用して、利用可能な加速ハードウェア１３０の実施仕様を取得する。このような実施仕様は、たとえば、以下のものを含む。

・加速ハードウェア１３０のサポートされるビデオ符号化パイプライン構成を特定する列挙アレイ（enumerated array）（たとえば、付録の§３．４．１に記載されたGetCapabilitiesインターフェースを介して取得される）、

・サポートされるビデオフォーマットの表示（たとえば、ＭＰＥＧ、ＷＭＶ等、付録の§３．４．２のGetSupportedFormatsを参照されたい）、

・動き推定（ＭＥ）オペレーション用のサポートされる検索メトリック（付録の§３．４．３のGetDistanceMetricsを参照されたい）、

・処理時間対品質のトレードオフを確定するためのサポートされる検索プロファイル（付録の§３．４．４のGetSearchProfilesを参照されたい）、及び／又は

・たとえば、画像サイズ情報、最大検索ウィンドウサイズ、可変マクロブロックサポート表示等のサポートされるＭＥ機能（付録の§３．４．５のGetMECapabilitiesを参照されたい）。

【００１３】

ビデオ符号化加速サービス１１８は、このような要求をビデオエンコーダ１１６から受け取ったことに応答して、要求された実施仕様をビデオ加速ハードウェア１３０に問い合わせ、加速ハードウェア１３０からビデオエンコーダ１１６へ、対応する応答に関連付けられる情報を返す。ビデオ符号化加速サービス１１８は、対応するデバイスドライバを使用してビデオ加速ハードウェア１３０とインターフェースする。このようなデバイスドライバは、「他のプログラムモジュール」１１４のそれぞれの部分として示される。

【００１４】

ビデオエンコーダ１１６は、ハードウェアで加速されることで利益（たとえば、速度の利益及び／又は品質の利益）を得ることができるあらゆる符号化オペレーションを特定し、ビデオ符号化品質と速度との間のトレードオフをカプセル化する検索プロファイルを選択すること、バスにわたるデータ移行及びプロセッサ間のデータ移行を最小にすること等を行うために、アプリケーションの特定のビデオ符号化アーキテクチャ（ソフトウェア実

装されたもの）を考慮して、加速ハードウェア１３０によってサポートされる実施仕様を評価する。ハードウェア加速で利益を得ることができる例示的なオペレーションには、たとえば、動き推定、変換、及び量子化が含まれる。たとえば、ハードウェアで量子化を行う１つの理由は、パイプラインステージ間のデータフローを最小にすることである。

【００１５】

図２は、ビデオ符号化パイプライン構成の１つの例示的な実施形態を示している。符号化プロセスのうちのいくつかは、ハードウェアで加速される。説明及び例示的な説明のために、図２に関連付けられるオペレーション及びデータフローは、図１のコンポーネントの特定のものに関して説明されている。この説明では、参照符号の左端の数字は、コンポーネント／データパス／参照アイテムが最初に紹介された特定の図を示している。たとえば、パイプライン２００の左端の数字は、たとえば「２」であり、パイプライン２００が図２で最初に紹介されていることを示している。この例では、ホスト１０２によって実施される処理オペレーションのそれぞれがハードウェア１３０で加速されるように、符号化パイプライン２００は、ビデオエンコーダ１１６（図１）がビデオ符号化サービス１１８とインターフェースすることによって構成／カスタマイズされている。説明のために、図２の太破線の右側に示す処理オペレーションは、ハードウェア（たとえば、図１の加速ハードウェア１３０）によって加速され、図の左側に示す処理オペレーションは、ホストコンピューティングデバイス１０２（図１）によって実行される。符号化パイプライン２００では、オプションの構成されたデータアクセスパスウェイが、太線でない破線で示される。楕円２０４及び２１２は、それぞれ、元の映像のメモリストア及び符号化された映像のメモリストアを表す。

【００１６】

この例の実施態様では、ビデオエンコーダ１１６（図１）は、或る形態の圧縮された又は未圧縮のビデオデータ２０２を入力として取り込む（図１の入力ビデオデータ１２２も参照されたい）。ソース２０２が、ホスト１０２を起源とするものではなく、且つ、ホスト意志決定エンジン（たとえば、ビデオエンコーダ１１６）がソースビデオを使用しない場合には、図２の例示的なパイプライン構成は、入力ソースビデオ２０２（「生のビデオソース」）をホストコンピューティングデバイス１０２にコピーしないことに留意されたい。たとえば、量子化決定が、ビデオデータに作用するようにホストに要求しない場合には、そのデータは転送されない。この例では、パイプライン２００は、ブロック２０６、２０８、及び２１４～２１８のそれぞれのオペレーションを使用して、入力データ２０２を別の圧縮された形態に変換するように構成されている。

【００１７】

このようなオペレーションは、未圧縮（ＹＵＶ）のビデオデータを圧縮されたＭＰＥＧ－２に変換することを含む場合もあるし、ビデオデータをＭＰＥＧ－２データフォーマットからＷＭＶデータフォーマットにトランスコードすることを含む場合もある。例示的な説明のために、トランスコードオペレーションは、解凍ステージを全体的又は部分的に含み、その後に符号化ステージが続くものと仮定する（解凍を迂回し、純粋に変換（ＤＣＴ）空間で機能する、より効率的なモデルが存在する）。複数のビデオ圧縮フォーマットが、動き推定、変換、及び量子化を活用して、圧縮を達成する。圧縮ステージの場合、動き推定は、通常、最も遅いステップであり、エンコーダ（たとえば、ビデオエンコーダ１１６）が、所与の画像のマクロブロックに関して、最も近似する基準マクロブロックを見つけようと試みる大規模な検索オペレーションを含む。

【００１８】

マクロブロックのそれぞれに関して最適な動きベクトルが（たとえば、ブロック２０６を介して）求められると、エンコーダ１１６は、事前に符号化された画像及び最適な動きベクトルに基づき（たとえば、ブロック２０８を介して）差分残差（differential residue）を計算する。動きベクトルは、この差分残差と共に、現画像のコンパクトな表現である。動きベクトルデータは、さらに、差分でも表現される。ホストエンコーダは、オプションとして、動きベクトル及び／又は残差のより小さな組み合わせを有するマクロブロッ

クを見つけるために、ビデオ加速ハードウェアによる動きベクトルの再評価を要求することができる。結果の差分動きベクトルデータ及び残差データは、たとえば、ランレングス符号化（ＲＬＥ）及び差分符号化（たとえば、ハフマン符号化及び算術符号化）のような技法を使用して（たとえば、ブロック２１８を介して）コンパクト化され、ユーザに提示するために宛先（ブロック２１８）へ通信する、最終的な符号化されたビットストリーム（符号化されたビデオデータ１２６）が生成される。この例では、ブロック２０６、２０８、及び２１４～２１８のオペレーション（たとえば、動き推定オペレーション（２０６）、モード決定オペレーション、動きベクトル（ＭＶ）選択オペレーション、及びレート制御オペレーション（２０８）、予測形成（prediction formation）オペレーション（２１０）、変換オペレーション及び量子化オペレーション（２１４）、逆量子化器（quantizer inversion）及び変換及びバージョン（２１６）、並びにエントロピー符号化（２１８））は、当該技術分野で既知であり、本明細書ではこれ以上説明しない。

【００１９】

図１を再び参照して、一実施態様では、ビデオエンコーダ１１６は、加速ハードウェア１３０の完全利用を可能にするマルチスレッド化されたアプリケーションである。この実施態様では、いずれのビデオ符号化オペレーションをハードウェアで加速するのかを決定するときに、ビデオエンコーダ１１６は、プロセッサ１０４及び加速ハードウェア１３０の双方が完全利用されるように特定のパイプライン構成を構築することができる。たとえば、ビデオ符号化パイプライン動き推定オペレーションが、ビデオデータの特定のフレームに関してハードウェアによって実行されているときに、パイプラインは、ビデオデータの異なるフレームに対してホストによってソフトウェアでエントロピー（又は算術若しくはハフマン）符号化オペレーションを実行するように構成することができる。選択／構築された特定のパイプライン構成を表す１つの例示的な単一の動きベクトルパイプラインが、付録のセクション５．１．１で後述される。ビデオエンコーダ１１６が複数の動きベクトルを加速ハードウェア１３０に要求し、さまざまなパラメータに基づいて１つの動きベクトルパイプラインを選択する例示的な複数の動きベクトル（比較的複雑な）パイプラインは、付録のセクション５．１．２で後述される。

【００２０】

検索プロファイルの選択に関して、動きベクトルの品質は、動きベクトルの使用によって生成されるストリームのビットレートを指す。高品質の動きベクトルは、低ビットレートストリームに関連付けられる。品質は、ブロック検索の完全性、アルゴリズムの質、使用される距離メトリック等によって決まる。高品質の動きベクトルは、高品質ビデオ符号化オペレーションを実行するのに使用されるべきである。これを扱うために、ビデオ符号化加速サービス１１８は、検索プロファイルと呼ばれる一般的な構成体を提供して、品質と時間との間のトレードオフをカプセル化する。検索プロファイルは、加速ハードウェア１３０等によって使用される検索アルゴリズムを特定するためのメタデータも含む。ビデオエンコーダ１１６は、エンコーダの実施態様の特定の要件に基づいて特定の検索プロファイルを選ぶ。

【００２１】

バスにわたるデータ移行及びプロセッサ間のデータ移行を最小にすることに関して、ビデオ符号化パイプライン構成によって実施される符号化プロセスは、通常、いくつかの処理ステージを含む。これらの処理ステージのそれぞれは、加速ハードウェア１３０を介して加速される場合もあるし、されない場合もある。ビデオエンコーダ１１６が、符号化パイプラインの連続したステージでハードウェア加速を利用することを決定した場合に、加速ハードウェア１３０ベースのメモリ１３２から、ホストコンピューティングデバイス１０２に関連付けられているシステムメモリ１０６へデータを移動させ、その後、次のステージのために、加速ハードウェアベースのメモリ１３２へデータを戻す等を行うことが必要でない場合がある。

【００２２】

より詳細には、さまざまなタイプのビデオ及び動きベクトルデータへのポインタは、ホ

ストコンピューティングデバイス１０２と加速ハードウェア１３０との間を往復して転送される場合があるが、一実施態様では、データポインタ（D3D9 Surface（サーフェス）ポインタ）が、たとえばIDirect3DSurface9::LockRectを使用して明示的にロックされているときにのみ、実際のデータは、システムメモリ１０６にコピーされる。サーフェスをロックするための例示的なインターフェースは既知である（たとえば、既知のIDirect3DSurface9::LockRect.interface）。したがって、２つの符号化パイプラインステージが互いの後に続き、且つ、ホストコンピューティングデバイス１０２が、中間処理を何ら実行する必要がない場合に、ホストコンピューティングデバイス１０２は、処理ステージ間に割り当てられたバッファを「Lock（ロック）」しないように決定することができる。これによって、データの冗長なメモリコピーが防止され、それによって、不要なデータ移動／転送が回避される。このようにして、ビデオエンコーダ１１６は、バスにわたるデータ転送及びプロセッサ間のデータ転送を実質的に最小にし、それによって、ビデオ符号化速度をさらに増加させるビデオ符号化パイプラインを設計する。

【００２３】

この時点で、ビデオエンコーダ１１６は、ハードウェアで加速されることで利益を得ることができるあらゆる符号化オペレーションを特定すること、検索プロファイルを選択すること、バスにわたるデータ移行及びプロセッサ間のデータ移行を最小にすること等を行うために、アプリケーションの特定のビデオ符号化アーキテクチャ（ソフトウェア実装されたもの）を考慮して、加速ハードウェア１３０によってサポートされる実施仕様を既に評価している。これらの決定に基づいて、ビデオエンコーダ１１６は、復号されたソースビデオデータを符号化し、それによって、符号化されたビデオデータ１２６を生成するように特定のパイプライン構成を選択する。次に、ビデオエンコーダ１１６は、ビデオ符号化加速サービス１１８とインターフェースして、エンコーダオブジェクトを作成し、選択されたパイプラインを実施する（付録の§３．４．６のCreateVideoEncoderを参照されたい）。この実施態様では、選択されたパイプライン構成及び以下のもののうちの１つ又は複数を特定することによって、エンコーダオブジェクト（たとえば、通常のＣＯＭオブジェクト）が作成される。以下のものとは、すなわち、符号化された出力ビットストリームのフォーマット、そのパイプライン構成に関連付けられる入力データストリーム及び出力データストリームの個数、静的な構成プロパティ、選択されたパイプライン構成に基づく異なるＩ／Ｏストリームとの関連付けのためのバッファ（サーフェス）の提案された個数、及び、グラフィックスデバイスドライバが収集することができる資源に基づくドライバ指定のアロケータキューサイズ、並びに他のパラメータである。（キューサイズ及びデータバッファの個数は、本質的には同じものを指している。一方は「提案される」ものであり、他方は「実際の」ものである。）

【００２４】

次に、ビデオエンコーダ１１６は、作成されたエンコーダオブジェクトを使用して、ビデオ符号化加速サービス１１８とインターフェースし、復号されたソースビデオデータを符号化する。このために、エンコーダオブジェクトは、実行要求を加速ハードウェア１３０にサブミットする（付録の§３．２．３のIVideoEncode:Execute APIを参照されたい）。

【００２５】

上記を考慮すると、システム１００によって、ビデオエンコーダアプリケーション１１６の任意の実施態様が、利用可能なビデオ符号化加速ハードウェアをフル活用して符号化速度及び符号化品質を増加させるために、ランタイム中にビデオ符号化パイプライン構成を定義して作成することが可能になる。これらのランタイム構成オペレーションの一部として、ビデオエンコーダ１１６は、ＶＡ　ＡＰＩ１２８を使用して、符号化パイプラインが反復有向検索（iterative directed searching）（精緻さを増加させる複数の検索経路）を実施すること、一般に選択可能な検索ストラテジー（たとえば、使用されている実際のアルゴリズムに関しての詳細のあらゆる知識とは無関係の品質メトリックに基づいて検索アルゴリズムを選択すること）を定義して使用すること、フォーマットから独立した方

法論を利用して（たとえば、ビデオエンコーダ１１６が、入力ビデオデータ１２２の特定の画像フォーマットを知らず、加速ハードウェア１３０が、符号化されたビデオデータ１２６の圧縮された出力フォーマットを知らない場合）検索を制御し、データサイズを適合させること（たとえば、ビデオエンコーダ１１６が、検索アルゴリズムに基づいてマクロブロックサイズを選択する場合）等を指定することができる。

【００２６】

１つの例示的なプロシージャ

図３は、一実施形態による加速ビデオ符号化の１つの例示的なプロシージャ３００を示している。例示的な説明のために、このプロシージャのオペレーションは、図１のシステム１００のコンポーネントに関して説明される。コンポーネントの参照符号の左端の数字は、そのコンポーネントが最初に説明された特定の図を示している。

【００２７】

ブロック３０２において、ビデオエンコーダ１１６（図１）は、入力ビデオデータ１２２を受け取る。入力ビデオデータ１２２が圧縮されていない場合には、入力ビデオデータは、復号されたソースビデオデータを表す。ブロック３０４において、入力ビデオデータ１２２が圧縮されている場合には、ビデオエンコーダ１１６は、入力ビデオデータを解凍して、復号されたソースビデオデータを生成する。ブロック３０６において、ビデオエンコーダ１１６は、ＶＡ　ＡＰＩ１２８とインターフェースして、機能及びビデオ符号化パイプライン構成の実施仕様に関して加速ハードウェア１３０に問い合わせる。ブロック３０８において、ビデオエンコーダ１１６は、ハードウェア加速で利益を得ることができるビデオエンコーダ１１６の特定の実施態様に関連付けられるビデオ符号化オペレーションを特定すること、符号化速度及び／又は符号化品質の決定を行うこと、バスにわたるデータ移行及びプロセッサ間のデータ移行を最小にすること等を行うために、ビデオエンコーダ１１６の実施態様の文脈内で、サポートされる機能及び実施仕様を評価する。

【００２８】

ブロック３１０において、ビデオエンコーダ１１６は、加速ハードウェア１３０によるハードウェア加速から利益を得ることができる特定されたビデオ符号化オペレーションを実行し、速度／品質トレードオフを（たとえば、選択された検索プロファイルを介して）実施すると共に、データフロー移行を最小にするように構成された符号化パイプラインを実施する符号化オブジェクトを作成する。ブロック３１２において、ビデオエンコーダは、作成されたエンコーダオブジェクトを使用して、ブロック３１０で生成された、カスタマイズされたビデオ符号化パイプラインによって描写されるオペレーションのシーケンス及び符号化アーキテクチャに従って、復号されたソースビデオデータを符号化する。ブロック３１２のこれらの符号化オペレーションは、符号化されたビデオデータ１２６（図１）を生成する。

【００２９】

結論

加速ビデオ符号化のシステム及び方法を、構造的な特徴及び／又は方法論的なオペレーション若しくは動作に固有の文言で説明してきたが、添付の特許請求の範囲に規定された実施態様は、説明した特定の特徴又は動作に必ずしも限定されるとは限らないことが理解される。

【００３０】

たとえば、図１のＡＰＩ１２８は、符号化ビデオデータの文脈内で説明されているが、ＡＰＩ１２８は、エッジ検出、動きベクトルベースのノイズ低減、画像安定化、鮮鋭化、フレームレート変換、コンピュータビジョンアプリケーション用の速度計算等、他の機能のハードウェア加速に関して、符号化の文脈外で使用することができる。ノイズ低減に関する例として、一実施態様では、ビデオエンコーダ１１６（図１）は、復号されたソース画像データのすべてのマクロブロックの動きベクトルを計算する。次に、ビデオエンコーダ１１６は、動きの大きさ、方向、及び周囲のマクロブロックの動きベクトルに対する相関を利用して、ローカルなオブジェクトの動きが入力画像にあるか否かを判断する。この

実施態様では、ビデオエンコーダ１１６は、次に、ベクトルの大きさを利用して、静的にランダムなノイズを低減するように、オブジェクトのトラッキング／フィルタリングアグレッシブネス（tracking / filtering aggressiveness）を管理するか又は特定のオブジェクトのシフトを平均化する。

【００３１】

画像安定化に関する別の例では、一実施態様において、ビデオエンコーダ１１６はすべてのマクロブロック及び復号されたソースデータの動きベクトルを計算する。ビデオエンコーダ１１６は、次に、大域的な動きが画像にあるか否かを判断する。これは、すべての動きベクトル値を相関させ、相関値が類似しているか否かを判断することによって実現される。相関値が類似している場合には、ビデオエンコーダ１１６は大域的な動きがあると結論付ける。或いは、ビデオエンコーダ１１６は、大きなマクロブロックサイズを利用し、その大きなマクロブロックの全体的な動きがあるか否かを判断する。グローバルな動きが存在するか否かを判断した後、ビデオエンコーダ１１６は、グローバルな動きベクトルがフレームにわたって動きが不自然になる傾向があることも発見した場合には、カメラのジャーキネスがあると結論付け、ノイズフィルタリングオペレーション及び符号化オペレーションを開始する前にこれを補償する。

【００３２】

したがって、システム１００の特定の特徴及びオペレーションは、特許を請求する主題を実施する例示的な形態として開示されている。

【００３３】

付録Ａ

例示的なビデオ符号化加速ＡＰＩ

ビデオ符号化

この付録は、ＶＡ符号化とも呼ばれる加速ビデオ符号化用のビデオ符号化加速ＡＰＩ１２８（図１）の１つの例示的な実施態様の複数の態様を記載している。この実施態様では、ＡＰＩ１２８は、符号化アプリケーション及びビデオ処理アプリケーション（たとえば、ビデオエンコーダモジュール１１６）が、Motion Estimation（動き推定）、Residue Computation（残差計算）、Motion Compensation（動き補償）、及びTransform（変換）を加速するための加速ハードウェア１３０（たとえば、ＧＰＵ）のサポートを利用することを可能にするように設計される。

１　目次

【００３４】

２　例示的な設計

２．１　エンコーダレイアウト

図５は、一実施形態による、ビデオ符号化加速ＡＰＩを利用することができる方法を示すための１つの例示的なビデオエンコーダアプリケーションを示している。この例では、ビデオエンコーダ１１６は、ＤＭＯ又はＭＦＴの形態で実施される。図５は、入力データ（「Receive」（受け取り）に対応する）及びいくつかの処理ステージ後の出力データを示している。ボックスはデータを表す一方、円は、エンコーダアプリケーションによって起動されるＡＰＩ機能を表す。したがって、円は、エンコーダアプリケーションによって見られるようなＡＰＩのコアを表す。

【００３５】

２．２　パイプライン構成又はモード構成

加速ハードウェアはパイプラインとみなされ、パイプラインＧＵＩＤは、パイプラインの最も基本的な構成要素を記述するのに使用される。符号化の高速化の目的は、パイプライン効率の目的に密接に結びついていると考えることができる。

【００３６】

この設計によって、最終的な出力が取得される前に、データがホストＰＣとハードウェアとの間で往復する分割（又はマルチステージ）パイプラインが可能になる。マルチステージパイプライン構成はまだ設計されておらず、以下の構成は、非分割のシングルステージパイプラインを説明している。

２．２．１　VA2_EncodePipe_Full

【００３７】

【数１】

```
// {BFC87EA2-63B6-4378-A619-5B451EDCB94D}
cpp_quote( "DEFINE_GUID(VA2_EncodePipe_Full, 0xbfc87ea2, 0x63b6, 0x4378, 0xa6, 0x19, 0x5b,
0x45, 0x1e, 0xdc, 0xb9, 0x40);" )
```

【００３８】

図６は、一実施形態による、加速ハードウェアが、Motion Estimation、Transform、Quantization（量子化）、及びそれらの逆のプロセスを加速して、復号された画像を生成する１つの例示的なビデオ符号化パイプライン構成を示している。ハードウェアは、出力としてMotion Vectors（動きベクトル）、Residues（ルマ（luma）及びクロマ（chroma））、並びにDecoded Image（復号された画像）を生成する。Decoded Imageの唯一の目的は次のフレームのMotion Vectorsを計算することであるので、復号された画像をシステムメモリに転送する必要はない。

【００３９】

後のドキュメンテーションは、NumStreamsと呼ばれるパラメータに言及する。このパイプライン構成では、NumStreamsは５である。実際のStreamIdは、図において括弧内に示されている。

これはシングルステージの非分割パイプラインであり、したがって、Execute（実行）

のStage（ステージ）パラメータは適用されない。

【００４０】

ストリーム記述

StreamType_*は、VA2_Encode_StremaType_*の省略形であることに留意されたい。

Input Image（入力画像）

ストリームＩＤは１であり、ストリームタイプはStreamType_Videoである。このストリームは、動きデータが求められる画像を表す。このストリームのアロケータは取り決めることができ、すなわち現在のインターフェースがアロケータを供給することもできるし、外部アロケータを使用することもできる。アロケータの選択は作成時に行われ、外部アロケータが選ばれた場合には、１のストリームＩＤは、GetBuffer（バッファ取得）の違法な入力値とみなされる。

Reference Images（基準画像）

ストリームＩＤは２であり、ストリームタイプはStreamType_Videoである。このストリームは動きベクトルを計算するのに使用される基準画像のリストを表す。現在のインターフェースは、このストリームに関して別個のアロケータを供給しない。入力サーフェスは、復号された画像ストリーム（ＩＤ＝５）から再循環されるか、又は他の場所から取得される。

Motion Vectors

ストリームＩＤは３であり、ストリームタイプはStreamType_MVである。このストリームは動きベクトルデータを含む出力パラメータを表す。このストリームのバッファはGetBufferのみを介して取得される。

Residues

ストリームＩＤは４であり、ストリームタイプはStreamType_Residuesである。このストリームは３つのすべての平面の残差値を含む出力パラメータを表す。このストリームのバッファはGetBufferのみを介して取得される。

Decoded Image

ストリームＩＤは５であり、ストリームタイプはStreamType_Videoである。このストリームは、量子化された残差及び動きベクトル値から取得された、復号された画像を含む出力パラメータを表す。このストリームのバッファは、GetBufferのみを介して取得される。

【００４１】

２．２．２　VA2_EncodePipe_MotionEstimation

【００４２】

【数２】

```
// {F18B3D19-CA3E-4a6b-AC10-53F86D509E04}
cpp_quote( "DEFINE_GUID(VA2_EncodePipe_MotionEstimation, 0xf18b3d19, 0xca3e, 0x4a6b, 0xac,
0x10, 0x53, 0xf8, 0x6d, 0x50, 0x9e, 0x4);" )
```

【００４３】

図７は、一実施形態による、ハードウェアがMotion Estimationのみを加速する１つの例示的なビデオ符号化パイプライン構成を示している。このパイプライン構成は、入力として基準画像のセットを取り込み、出力としてMotion Vectorsをダンプする。この場合のDecoded Imageは、ホストソフトウェアが生成して供給しなければならない。

【００４４】

このパイプライン構成のNumStreamsは３である。さまざまなストリームのStreamIdは、図において括弧内に示される。

これはシングルステージの非分割パイプラインであり、ExecuteのStageパラメータは適用されない。

【００４５】

３　例示的なＡＰＩ

３．１　インターフェース定義

3．1．1 IVideoEncoder

【００４６】

【数３】

```
interface IVideoEncoder : IUnknown
{
    HRESULT GetBuffer(
        [in] UINT8 StreamId,
        [in] UINT32 StreamType,
        [in] BOOL Blocking,
        [out] PVOID pBuffer

       );

    HRESULT ReleaseBuffer(
        [in] UINT8 StreamId,
        [in] UINT32 StreamType,
        [in] PVOID pBuffer
        );

    HRESULT Execute(
        [in] UINT8 Stage,
        [in] UINT32 NumInputDataParameters,
        [in, size_is(NumInputDataParameters)]
             VA2_Encode_ExecuteDataParameter** pInputData,
        [in] UINT32 NumOutputDataParameters,
        [out, size_is(NumOutputDataParameters)]
             VA2_Encode_ExecuteDataParameter** pOutputData,
        [in] UINT32 NumConfigurationParameters,
        [in, size_is(NumConfigurationParameters)]
             VA2_Encode_ExecuteConfigurationParameter** pConfiguration,
        [in] HANDLE hEvent,
        [out] HRESULT* pStatus
        );
}
```

【００４７】

3．1．2 IVideoEncoderService

【００４８】

【数４】

```
interface IVideoEncoderService : IVideoAccelerationService
{
    HRESULT GetPipelineConfigurations(
        [out] UINT32* pCount,
        [out, unique, size_is(*pCount)] GUID** pGuids
        );

    HRESULT GetFormats(
        [out] UINT32* pCount,
        [out, unique, size_is(*pCount)] GUID** pGuids
        );

    HRESULT GetDistanceMetrics(
        [out] UINT32* pCount,
        [out, unique, size_is(*pCount)] GUID** pGuids
        );

    HRESULT GetSearchProfiles(
        [out] UINT32* pCount,
        [out, unique, size_is(*pCount)] VA2_Encode_SearchProfile** pSearchProfiles
        );

    HRESULT GetMECapabilities(
        [out] VA2_Encode_MECaps* pMECaps
        );

    HRESULT CreateVideoEncoder(
        [in] REFGUID PipelineGuid,
        [in] REFGUID FormatGuid,
        [in] UINT32 NumStreams,
        [in] VA2_Encode_StaticConfiguration* pConfiguration,
        [in, size_is(NumStreams)] VA2_Encode_DataDescription* pDataDescription,
        [in, size_is(NumStreams)] VA2_Encode_Allocator* SuggestedAllocatorProperties,
        [out, size_is(NumStreams)] VA2_Encode_Allocator* pActualAllocatorProperties,
        [out] IVideoEncoder** ppEncode
        );
};
```

【００４９】
３．２　メソッド：IVideoEncoder
３．２．１　GetBuffer
この関数はExecuteコールで使用するためのバッファ（符号化されたサーフェス）を返す。バッファは、パイプラインのストールを回避するために、ReleaseBuffer（バッファ解放）をコールすることによって使用後に即座に解放される。
【００５０】
【数５】

```
HRESULT GetBuffer(
    [in] UINT8 StreamId,
    [in] UINT32 StreamType,
    [in] BOOL Blocking,
    [out] PVOID pBuffer
    );

#define E_NOTAVAILABLE          HRESULT_FROM_WIN32(ERROR_INSUFFICIENT_BUFFER)
#define E_INVALIDPARAMETER      HRESULT_FROM_WIN32(ERROR_INVALID_PARAMETER)
```

【００５１】
パラメータ
StreamId
バッファが所望される特定のストリームを指す。特定のストリームに応じて、Input Imageバッファ、Motion Vectorバッファ等のような異なるタイプのバッファが返される。所与の構成のすべてのストリームＩＤが、この関数の有効な入力であるとは限らない。StreamIdの許容値は、パイプライン構成の一部として指定される。
StreamType
返されるバッファのタイプを指定する。通常、ストリームタイプはStreamIdによって暗に示され、作成時に取り決められる。StreamTypeがStreamIdと一致していない場合には、この関数はエラー値を返す。データバッファは、備考欄の表によって記載されたようなStremaTypeの値に基づいて解釈（タイプキャスト）される。
Blocking
飢餓又はスロットルする他の或る必要性が存在する時の関数の振る舞いを指定する。True（真）の値は、関数がブロックすべきことを示す一方、False（偽）は、関数がE_NOTAVAILABLEを返すべきであることを示す。
pBuffer
ReleaseBufferを介して解放されるデータバッファのポインタである。このポインタは、StreamTypeパラメータに基づいて再キャスト（解釈）され、これは、以下の備考欄の表に記載されている。
【００５２】
戻り値
S_OK
関数は成功した。
E_NOTAVAILABLE
これは、ドライバがバッファ飢餓状態にあるときに返され、Blockingフラグは偽にセットされる。
E_INVALIDPARAMETER
入力パラメータが正しくなかった。これは、たとえば、StreamTypeが所与のStreamIdの予測値と一致しないときに使用することができる。
VFW_E_UNSUPPORTED_STREAM
StreamIdが無効である。GetBufferは、指定されたストリームＩＤのバッファを供給しない。StreamIdの許容値は、パイプライン構成の一部して記述される。指定されたストリームＩＤに関して、アロケータは外部の場合もあるし、アロケータが全くない場合もある。
E_FAIL

関数は失敗した。

【００５３】

備考

標準的には、この関数は、バッファがアロケータキューに既に存在する時は、直ちに制御を返す。この関数がブロックすべき（又はE_NOTAVAILABLEを返すべき）唯一の状況は、アロケータキューからのすべてのバッファが、デバイスにサブミットされてしまっているか、又は、アプリケーションによって消費されており、したがって、解放されない時である。

【００５４】

【表１】

ストリームタイプ及びバッファフォーマット

StreamType_Video	IDirect3DSurface9**pBuffer.
StreamType_MV	IDirect3DSurface9**pBuffer
StreamType_Residues	IDirect3DSurface9*pBuffer[3]。すなわち、pBufferは、３つのD3Dサーフェスポインタへのポインタである。 pBuffer[0]は、ルマサーフェスへのポインタである。 pBuffer[1]は、Ｃｂサーフェスへのポインタであるか、又は、適用可能でない場合にはNULLである。 pBuffer[2]は、Ｃｒサーフェスへのポインタであるか、又は、適用可能でない場合にはNULLである。

３．２．２　ReleaseBuffer

【００５５】

この関数は、サーフェスを、GetBufferを介して再利用するために、解放してアロケータキューに戻すのに使用される。

【００５６】

【数６】

```
HRESULT ReleaseBuffer(
    [in] UINT8 StreamId,
    [in] UINT8 StreamType,
    [in] PVOID pBuffer
    );
```

【００５７】

パラメータ

StreamId

バッファに関連付けられているストリームＩＤである。

StreamType

バッファのストリームタイプである。

pBuffer

解放されてアロケータキューに戻されるバッファである。

【００５８】

戻り値

S_OK

関数は成功した。

E_FAIL

関数は失敗した。

【００５９】

３．２．３　Execute

この関数は、ハードウェアに要求をサブミットするのに使用される。この関数は、いくつかの構成情報と共に、GetBufferを介して取得された入力データバッファ及び出力デー

タバッファを含む。この関数は非同期であり、その完了は、イベントがシグナリングされることによって示される。完了ステータスは、pStatusパラメータを使用して示される。pStatusパラメータは、ヒープ上に割り当てられ、イベントがシグナリングされた後にのみチェックされる。

【００６０】

この関数にパラメータとして供給されたバッファは、関数が真に完了するまでアプリケーションによって（たとえば、LockRectを介して）読み出されることも書き込まれることもない。真の完了は、エラー値がこの関数によって返されることによって、又は、この関数が成功を返す場合には、hEvent（この関数へのパラメータ）のシグナリングによって、暗に示される。同じバッファが、Executeコールのうちのいくつかのインスタンスに入力されるとき、関連付けられるすべてのExecuteコールが完了するまで、そのバッファはアクセスされない。Executeによって使用中のサーフェスへのポインタは、それでも、ExecuteのようなＶＡ関数へのパラメータとして供給することができる。これは、データをロックすることを要しないからである。この最後のルールは、複数のExecuteコールにおいて同じ入力画像を同時にどのようにして使用することができるのかを説明している。

【００６１】

このコールに供給されるバッファは、作成時に取り決められたアロケータのセマンティクスに従う。GetBufferの使用が予測されるときにおいて、外部アロケータが使用される場合、この関数はE_FAILを返す。

【００６２】

【数７】

```
HRESULT Execute(
    [in] UINT8 Stage,
    [in] UINT32 NumInputDataParameters,
    [in, size_is(NumInputDataParameters)]
         VA2_Encode_ExecuteDataParameter** pInputData,
    [in] UINT32 NumOutputDataParameters,
    [out, size_is(NumOutputDataParameters)]
         VA2_Encode_ExecuteDataParameter** pOutputData,
    [in] UINT32 NumConfigurationParameters,
    [in, size_is(NumConfigurationParameters]
         VA2_Encode_ExecuteConfigurationParameter** pConfiguration,
    [in] HANDLE hEvent,
    [out] HRESULT* pStatus
    );
```

【００６３】

パラメータ

Stage

分割パイプライン構成では、このパラメータは、分割パイプラインの特定のステージを識別する。番号付けは１から始まり、非分割パイプラインでは、このパラメータは無視される。

NumInputDataParameters

入力データアレイ（次のパラメータ）のサイズである。

pInputData

入力データ値へのポインタのアレイである。個々のデータポインタは、作成時に指定された、関連付けられるStreamDescriptionを有するStreamId値に基づいて適切に再キャストされる。データバッファは、作成時に割り当てられ、GetBufferをコールすることによってストリーミングプロセス中に取得される。

NumOutputDataParameters

出力データアレイ（次のパラメータ）のサイズである。

pOutputData

出力データ値へのポインタのアレイである。個々のデータポインタは、作成時に指定された、関連付けられるStreamDescriptionを有するStreamId値に基づいて適切に再キャストされる。データバッファは、作成時に割り当てられ、GetBufferをコールすることによってストリーミングプロセス中に取得される。

NumConfigurationParameters
構成アレイ（次のパラメータ）のサイズである。
pConfiguration
パイプラインの実行を制御する構成パラメータのアレイである。全体的な構成は、この構造を、エンコーダが作成された時に供給された静的構成パラメータと結合したものである。
hEvent
出力データが準備されたことをシグナリングするイベントハンドルである。
pStatus
要求されたオペレーションの完了が成功したか否かを示すステータスである。許容値には、S_OK（完了の成功）、E_TIMEOUT（TimeLimitを超えた場合）、及びE_SCENECHANGE（シーンチェンジ検出が有効であり、検出された場合）が含まれる。双方のエラーの場合に、出力サーフェスのいずれも有用なデータを含まない。このパラメータは、ヒープ上に割り当てられ、戻り値は、hEventがシグナリングされた後にのみチェックされる。

【００６４】

戻り値
S_OK
関数は成功した。
E_FAIL
関数は失敗した。

【００６５】

備考
イベントハンドルがシグナリングされた場合、これは、出力サーフェスが準備されているので、LockRectが、出力サーフェスのいずれかにおいてコールされたときに、すぐに完了すべきことを意味する。詳細には、LockRectコールは、あらゆるイベントハンドルに応対することによって、どの時間の長さに関してもブロックしないものと予測される。また、ビジースピン（busy spin）を通じてＣＰＵ時間を浪費することも認められない。

【００６６】

３．３　データ構造：Execute
Executeコールは、データパラメータ及び構成パラメータを有する。特定のデータパラメータは、VA2_Encode_ExecuteDataParameter基本クラス（又は構造）から派生すると考えることができ、特定の構成パラメータは、VA2_Encode_ExecuteConfigurationParameter基本クラス（又は構造）から派生すると考えることができる。

【００６７】

３．３．１　VA2_Encode_ExecuteDataParameter

【００６８】

【数８】

```
typedef struct _VA2_Encode_ExecuteDataParameter {
    UINT32      Length;
    UINT32      StreamId;
} VA2_Encode_ExecuteDataParameter;
```

【００６９】

メンバー
Length
この構造のバイト数である。拡張性のために提供される。
StreamId
パイプライン構成で定義されたデータストリームのＩＤである。バッファフォーマットは、StreamDescriptionパラメータを使用して作成時に取り決められる。

【００７０】

３．３．２　VA2_Encode_ExecuteConfigurationParameter

【００７１】

【数９】

```
typedef struct _VA2_Encode_ExecuteConfigurationParameter {
    UINT32      Length;
    UINT32      StreamId;
    UINT32      ConfigurationType;
} VA2_Encode_ExecuteConfigurationParameter;

#define VA2_Encode_ConfigurationType_MotionEstimation     0x1
#define VA2_Encode_ConfigurationType_Quantization         0x2
```

【００７２】

メンバー

Length

この構造のバイト数である。拡張性のために提供される。

StreamId

パイプライン構成で定義されたデータストリームのＩＤである。これは、データが入力であるか又は出力であるかを推論するのに使用することができる。

ConfigurationType

このパラメータは、構成パラメータを表し、現在の構造を適切にタイプキャストするのに使用される。

【００７３】

備考

この構造は、より特殊化された構成情報の基本タイプとして機能する。この基本タイプは、ConfigurationTypeパラメータに基づいて、より特殊化されたタイプにタイプキャストされる。ConfigurationTypeと特殊化された構造との間のマッピングは、以下の表に記載されている。

【００７４】

【表２】

構成タイプ

ConfigurationType_MotionEstimation	ConfigurationParameter_MotionEstimation
ConfigurationType_Quantization	ConfigurationParameter_Quantization

３．３．３　DataParameter_MotiionVectors

【００７５】

【数１０】

```
typedef struct _VA2_Encode_ExecuteDataParameter_MotionVectors {
    UINT32                  Length;
    UINT32                  StreamId;
    VA2_Encode_MVSurface* pMVSurface;
} VA2_Encode_ExecuteDataParameter_MotionVectors;
```

【００７６】

メンバー

Length

この構造のバイト数である。拡張性のために提供される。

StreamId

パイプライン構成で定義されたデータストリームのＩＤである。これは、データが入力であるか又は出力であるかを推論するのに使用することができる。

pMVSurface

Motion Vector D3D Surface（動きベクトルＤ３Ｄサーフェス）を含む構造へのポインタである。

【００７７】

３．３．４　DataParameter_Residues

【００７８】

【数１１】

```
typedef struct _VA2_Encode_ExecuteDataParameter_Residues {
    UINT32                          Length;
    UINT32                          StreamId;
    VA2_Encode_ResidueSurface*     pResidueSurfaceY;
    VA2_Encode_ResidueSurface*     pResidueSurfaceCb;
    VA2_Encode_ResidueSurface*     pResidueSurfaceCr;
} VA2_Encode_ExecuteDataParameter_Residues;
```

【００７９】

メンバー

Length

この構造のバイト数である。拡張性のために提供される。

StreamId

パイプライン構成で定義されたデータストリームのＩＤである。これは、データが入力であるか又は出力であるかを推論するのに使用することができる。

pResidueSurfaceY

ルマ値を含む残差サーフェスである。

pResidueSurfaceCb

クロマＣｂ値を含む残差サーフェスである。

pResidueSurfaceCr

クロマＣｒ値を含む残差サーフェスである。

【００８０】

３．３．５　DataParameter_InputImage

【００８１】

【数１２】

```
typedef struct _VA2_Encode_ExecuteDataParameter_InputImage {
    UINT32                  Length;
    UINT32                  StreamId;

    VA2_Encode_ImageInfo* pImageData;
} VA2_Encode_ExecuteDataParameter_InputImage;
```

【００８２】

メンバー

Length

この構造のバイト数である。拡張性のために提供される。

StreamId

パイプライン構成で定義されたデータストリームのＩＤである。これは、データが入力であるか又は出力であるかを推論するのに使用することができる。

pImageData

入力画像D3D Surface（Ｄ３Ｄサーフェス）を含む構造へのポインタである。これは、動きベクトルが求められるサーフェスである。

【００８３】

３．３．６　DataPrameter_ReferenceImages

【００８４】

【数１３】

```
typedef struct _VA2_Encode_ExecuteDataParameter_ReferenceImages {
    UINT32                  Length;
    UINT32                  StreamId;
    UINT32                  NumReferenceImages;
    VA2_Encode_ImageInfo* pReferenceImages
} VA2_Encode_ExecuteDataParameter_ReferenceImages;
```

【００８５】

メンバー

Length

この構造のバイト数である。拡張性のために提供される。

StreamId
パイプライン構成で定義されたデータストリームのＩＤである。これは、データが入力であるか又は出力であるかを推論するのに使用することができる。
Datatype
NumReferenceImages
基準画像アレイ（次のパラメータ）のサイズである。
pReferenceImages
動きベクトルが基礎とする基準画像のアレイである。ＭＰＥＧ－２のような簡単なフォーマットでは、１つのプログレッシブフレーム（又は２つのフィールド）のみを使用することができる。他方、Ｈ．２６４及びＶＣ－１のようなフォーマットは、いくつかのフレームにまたがる動きベクトルをサポートする。ＭＰＥＧ－２のＰフレームは、１つの基準画像しか使用しない一方、インターレースされたビデオを有するＢフレーム及びフィールドタイプの動きは、４つの画像を使用する場合がある。これらの４つの画像のそれぞれは、フレーム又はフィールドを指す場合がある。

【００８６】

３．３．７　DataParameter_DecodedImage

【００８７】

【数１４】

```
typedef struct _VA2_Encode_ExecuteDataParameter_DecodedImage {
    UINT32                  Length;
    UINT32                  StreamId;
    VA2_Encode_ImageInfo* pYCbCrImage;
} VA2_Encode_ExecuteDataParameter_DecodedImage;
```

【００８８】

メンバー
Length
この構造のバイト数である。拡張性のために提供される。
StreamId
パイプライン構成で定義されたデータストリームのＩＤである。これは、データが入力であるか又は出力であるかを推論するのに使用することができる。
Datatype
pYCbCrImage
逆量子化、逆変換、及び動き補償後に取得される復号された出力画像である。パイプラインを良好にするために、関連付けられるD3D Surfaceはロックされるべきではなく、データも不必要にシステムメモリへ転送されるべきではない。サーフェスポインタは、依然としてReference Image（基準画像）として使用することができる。

【００８９】

３．３．８　VA2_Encode_ImageInfo

【００９０】

【数１５】

```
typedef struct _VA2_Encode_ImageInfo {
    IDirect3DSurface9*  pSurface;
    BOOL                Field;
    BOOL                Interlaced;
    RECT                Window;
} VA2_Encode_ImageInfo;
```

【００９１】

メンバー
pSurface
YCbCrフォーマットの画像を含むＤ３Ｄサーフェスへのポインタである。
Filed
１の値は、サーフェスがビデオデータのフィールドを含み、且つこのデータがインターレースされていると仮定されることを示す。０は、フルプログレッシブフレーム（full

progressive frame）を示す。
Interlaced
１の値は、画像データがインターレースされていることを示す。このフラグは、Field（上記パラメータ）が１にセットされているときにのみ使用されるべきである。Fieldが１にセットされている場合、データはインターレースされていると仮定される。
Window
画像内の長方形である。これは、画像全体内で長方形のみに関して動きベクトルを返すように、Motion Estimationコールを制限するのに使用することができる。
【００９２】
３．３．９　ConfigurationParameter_MotionEstimation
【００９３】
【数１６】

```
typedef struct _VA2_Encode_ExecuteConfigurationParameter_MotionEstimation {
    UINT32                      Length;
    UINT32                      StreamId;
    UINT32                      ConfigurationType;
    VA2_Encode_MEParameters*  pMEParams;
} VA2_Encode_ExecuteConfigurationParameter_MotionEstimation;
```

【００９４】
メンバー
Length
この構造のバイト数である。拡張性のために提供される。
StreamId
パイプライン構成で定義されたデータストリームのＩＤである。これは、データが入力であるか又は出力であるかを推論するのに使用することができる。
ConfigurationType
pMEParams
検索ウィンドウ等を含む動き検索を統制するさまざまなパラメータを定義する構造へのポインタである。
【００９５】
備考
図８は、一実施形態によるいくつかの例示的なMotion Estimationパラメータを示している。これらのパラメータは、以下の構造で使用するものである。
【００９６】
３．３．１０　VA2_Encode_SearchResolution
【００９７】
【数１７】

```
typedef enum {
    VA2_Encode_SearchResolution_FullPixel,
    VA2_Encode_SearchResolution_HalfPixel,
    VA2_Encode_SearchResolution_QuarterPixel
} VA2_Encode_SearchResolution;
```

【００９８】
説明
FullPixel
Motion Vectorsが、フルピクセル単位で計算される。
HalfPixel
Motion Vectorsが、ハーフピクセル単位で計算される。したがって、（５，５）のMotion Vector値は、（２．５，２．５）ピクセル離れたデータのマクロブロックを指す。
QuartrePixel
Motion Vectorsが、４分の１ピクセル単位で計算される。したがって、（１０，１０）のMotion Vector値は、（２．５，２．５）ピクセル離れたデータのマクロブロックを指す。

サブピクセルの動きベクトル値の計算において、エンコーダは、補間を使用してルマ値及びクロマ値を推定する。特定の補間方式は、フォーマットに依存し、以下のＧＵＩＤ（静的構成の一部）がその補間方式を制御する。

【００９９】

【数１８】

```
// {E9AF78CB-7A8A-4d62-887F-B6A418364C79}
cpp_quote( "DEFINE_GUID(VA2_Encode_Interpolation_MPEG2Bilinear, 0xe9af78cb, 0x7a8a, 0x4d62,
0x88, 0x7f, 0xb6, 0xa4, 0x18, 0x36, 0x4c, 0x79);" )

// {A94BBFCB-1BF1-475c-92DE-67298AF56BB0}
cpp_quote( "DEFINE_GUID(VA2_Encode_Interpolation_MPEG2Bicubic, 0xa94bbfcb, 0x1bf1, 0x475c,
0x92, 0xde, 0x67, 0x29, 0x8a, 0xf5, 0x6b, 0xb0);" )
```

【０１００】

３．３．１１　VA2_Encode_SearchProfile

【０１０１】

【数１９】

```
typedef struct _VA2_Encode_SearchProfile {
    UINT8   QualityLevel;
    UINT8   TimeTaken;
    GUID    SearchTechnique;
    GUID    SubpixelInterpolation;
} VA2_Encode_SearchProfile;
```

【０１０２】

メンバー

QualityLevel

デバイスによってサポートされる異なるプロファイル間におけるMotion Vectorsの相対的な品質を示す、範囲［０～１００］の数字である。

TimeTaken

異なる検索プロファイルに関して要する相対的な時間を示す、範囲［０～１００］の数字である。これによって、アプリケーションは、妥当な時間と品質とのトレードオフを行うことが可能になる。

SearchTechnique

使用される検索アルゴリズムを示すＧＵＩＤである。

SubpixelInterpolation

使用されるサブピクセル補間方式を示すＧＵＩＤである。

【０１０３】

備考

普遍的に受け入れられる絶対的品質の定義は存在せず、したがって、相対的な尺度で妥協している。TimeTakenに対して示された値は、厳密な比例則に従うべきである。プロファイル１が１０ｍｓを要し、プロファイル２が２０ｍｓを要する場合に、TimeTaken値は、比２０／１０＝２にあるべきである。

【０１０４】

３．３．１２　VA2_Encode_MBDescription

【０１０５】

【数２０】

```
typedef struct _VA2_Encode_MBDescription {
    BOOL    ConstantMBSize;
    UINT32  MBWidth;
    UINT32  MBHeight;
    UINT32  MBCount;
    RECT*   pMBRectangles;
} VA2_Encode_MBDescription;
```

【０１０６】

メンバー

ConstantMBSize

１の値は、現在の画像のすべてのMacroblocks（マクロブロック）が同じサイズを有することを示す。これは、Ｈ．２６４のようなフォーマットには当てはまらない場合がある。

MBWidth

マクロブロックの幅である。bConstantMBSizeが１である場合にのみ有効である。

MBHeight

マクロブロックの高さである。bConstantMBSizeが１である場合にのみ有効である。

MBCount

bConstantMBSizeが０である場合に、画像のマクロブロック（又はセグメント）は、長方形のアレイを使用して記述される。このパラメータは、次のpMBRectanglesパラメータのエレメントのサイズを表す。

pMBRectangles

画像がどのように分割されるのかを表す長方形のアレイである。

【０１０７】

３．３．１３　VA2_Encode_SearchBounds

【０１０８】

【数２１】

```
typedef struct _VA2_Encode_SearchBounds {
    BOOL    DetectSceneChange;
    UINT32  MaxDistanceInMetric;
    UINT32  TimeLimit;
    UINT32  MaxSearchWindowX;
    UINT32  MaxSearchWindowY;
} VA2_Encode_SearchBounds;
```

【０１０９】

メンバー

DetectSceneChange

この値が１である場合、シーンチェンジ検出が要求されている。このような場合に、シーンチェンジが検出されると、動きベクトルは、Executeコールによって計算されず、したがって、残差も復号された画像も計算されない。これは、この場合にE_SCENECHANGEにセットされるべきExecuteコールのpStatusパラメータを介して示される。

MaxDistanceInMetric

現在選択されている距離メトリックを使用して比較が行われるときのマクロブロック間の相違を指す。この距離がこのMaxDistanceInMetric値を超えている場合には、このような動きベクトルは拒否される。

TimeLimit

Motion Estimationステージでハードウェアが費やすことが可能な最大時間である。ハードウェアがこの時間よりも長い時間を要する場合、ExecuteコールのpStatusパラメータは、E_TIMEOUTにセットされる。

SearchWindowX

返された動きベクトルのｘ成分の最大値である。換言すれば、検索ウィンドウの（ｘ次元に沿った）サイズである。

SearchWindowY

動きベクトルのｙ成分の最大値である。換言すれば、検索ウィンドウの（ｙ次元に沿った）サイズである。

【０１１０】

備考

３．３．１４　VA2_Encode_ModeType

【０１１１】

【数２２】

```
typedef struct _VA2_Encode_ModeType {
    UINT32  SearchProfileIndex;
    GUID    DistanceMetric;
    INT16   HintX;
    INT16   HintY;
} VA2_Encode_ModeType;
```

【０１１２】

メンバー

SearchProfileIndex

GetSearchProfiles（検索プロファイル取得）ＡＰＩコールによって返された検索プロファイルのリスト内へのインデックスである。

DistanceMetric

２つのマクロブロックを比較するときに使用するメトリックである。一般に使用されるメトリックには、ＳＡＤ（絶対差の合計）及びＳＳＥ（自乗誤差の合計）が含まれる。

HintX

動き検索をガイドするための動きベクトルの予測される方向に関してのヒントである。これは、画像の全体的な動きを指し、各ＭＢ単位で適用可能でない場合がある。

HintY

動き検索をガイドするための動きベクトルの予測される方向に関してのヒントである。これは、画像の全体的な動きを指し、各ＭＢ単位で適用可能でない場合がある。

【０１１３】

３．３．１５　ConfigurationParameter_Quantization

【０１１４】

【数２３】

```
typedef struct _VA2_Encode_ExecuteConfigurationParameter_Quantization {
    UINT32 Length;
    UINT32 StreamId;
    UINT32 ConfigurationType;
    UINT32 StepSize;
} VA2_Encode_ExecuteConfigurationParameter_Quantization;
```

【０１１５】

メンバー

Length

この構造のバイト数である。拡張性のために提供される。

StreamId

パイプライン構成で定義されたデータストリームのＩＤである。これは、データが入力であるか又は出力であるかを推論するのに使用することができる。

ConfigurationType

StepSize

量子化を行うときに使用されるステップサイズである。この設計によって、Motion Vectors及びResiduesがこのコールで要求された画像の部分全体に関して１つのステップサイズのみを使用することが可能になる。

【０１１６】

３．４　メソッド：IVideoEncoderService

このインターフェースのメソッドによって、アプリケーションは、その機能に関してハードウェアに問い合わせることが可能になり、所与の構成でエンコーダオブジェクトを作成することが可能になる。

【０１１７】

３．４．１　GetPipelineConfigurations

【０１１８】

【数２４】

```
HRESULT GetPipelineConfigurations(
    [out] UINT32* pCount,
    [out, unique, size_is(*pCount)] GUID** pGuids
    );
```

【０１１９】

パラメータ

pCount

戻り値は、この関数によって返されたpGuidsアレイ（次のパラメータ）のサイズを表す。

pGuids

デバイスによってサポートされるさまざまなパイプライン構成を表すＧＵＩＤのアレイである。メモリが被コール側によって割り当てられ、CoTaskMemFreeを使用してコール側によって解放されるべきである。

【０１２０】

戻り値

S_OK

関数は成功した。

E_OUTOFMEMORY

関数は、メモリを割り当ててＧＵＩＤのリストを返すことができなかった。

E_FAIL

或るデバイスエラーによって、サポートされるパイプライン構成を決定することができなかった。

【０１２１】

３．４．２　GetFormats

【０１２２】

【数２５】

```
HRESULT GetFormats(
    [out] UINT32* pCount,
    [out, unique, size_is(*pCount)] GUID** pGuids
    );
```

【０１２３】

パラメータ

pCount

戻り値は、この関数によって返されたpGuidsアレイ（次のパラメータ）のサイズを表す。

pGuids

デバイスによってサポートされるさまざまなフォーマット（たとえば、ＷＭＶ９、ＭＰＥＧ－２等）を表すＧＵＩＤのアレイである。メモリが被コール側によって割り当てられ、CoTaskMemFreeを使用してコール側によって解放されるべきである。

【０１２４】

３．４．３　GetDistanceMetrics

【０１２５】

【数２６】

```
HRESULT GetDistanceMetrics(
    [out] UINT32* pCount,
    [out, unique, size_is(*pCount)] GUID** pGuids
    );
```

【０１２６】

パラメータ

pCount

戻り値は、この関数によって返されたpGuidsアレイ（次のパラメータ）のサイズを表す。

pGuids
動き推定のデバイスによってサポートされるさまざまな検索メトリックを表すＧＵＩＤのアレイである。メモリが被コール側によって割り当てられ、CoTaskMemFreeを使用してコール側によって解放される。
【０１２７】
戻り値
S_OK
関数は成功した。
E_OUTOFMEMORY
関数は、メモリを割り当ててＧＵＩＤのリストを返すことができなかった。
E_FAIL
或るデバイスエラーによって、サポートされるメトリックを決定することができなかった。
【０１２８】
３．４．４　GetSearchProfiles
【０１２９】
【数２７】

```
HRESULT GetSearchProfiles(
    [out] UINT32* pCount,
    [out, unique, size_is(*pCount)] VA2_Encode_SearchProfile** pSearchProfiles
    );
```

【０１３０】
パラメータ
pCount
戻り値は、この関数によって返されたpGuidsアレイ（次のパラメータ）のサイズを表す。
pSearchProfiles
デバイスによってサポートされる検索プロファイルを表すＧＵＩＤのアレイである。検索プロファイルによって、コーデックアプリケーションの時間と品質とのトレードオフが、より効率的に可能になる。メモリが被コール側によって割り当てられ、CoTaskMemFreeを使用してコール側によって解放される。
【０１３１】
戻り値
S_OK
関数は成功した。
E_OUTOFMEMORY
関数は、メモリを割り当ててＧＵＩＤのリストを返すことができなかった。
E_FAIL
或るデバイスエラーによって、サポートされる検索プロファイルを決定することができなかった。
【０１３２】
３．４．５　GetMECapabilities
【０１３３】
【数２８】

```
HRESULT GetMECapabilities(
    [out] VA2_Encode_MECaps* pMECaps
    );
```

【０１３４】
パラメータ
pMECaps
デバイスのMotion Estimation機能へのポインタである。これは、デバイスがハンドリングすることができる画像のサイズ、最大検索ウィンドウサイズ、及びデバイスが可変

マクロブロックサイズをサポートするか否かに関しての情報を含む。このパラメータ用のメモリは、コール側によって割り当てられる。

【０１３５】

戻り値

S_OK

関数は成功した。

E_FAIL

関数は、或るデバイスエラーによって失敗した。

【０１３６】

３．４．６　CreateVideoEncoder

この関数は、IVideoEncoderのインスタンスを作成する。

【０１３７】

【数２９】

```
HRESULT CreateVideoEncoder(
    [in] REFGUID PipelineGuid,
    [in] REFGUID FormatGuid,
    [in] UINT32 NumStreams,
    [in] VA2_Encode_StaticConfiguration* pConfiguration,
    [in, size_is(NumStreams)] VA2_Encode_StreamDescription* pStreamDescription,
    [in, size_is(NumStreams)] VA2_Encode_Allocator* SuggestedAllocatorProperties,
    [out, size_is(NumStreams)] VA2_Encode_Allocator* pActualAllocatorProperties,
    [out] IVideoEncoder** ppEncode
    );
```

【０１３８】

パラメータ

PipelineGuid

所望のパイプライン構成を表すＧＵＩＤである。構成のリストは、GetCapabilitiesを介して取得され、ＧＵＩＤのそれぞれは、構成に関しての必要な詳細を記載した公開ドキュメンテーションに関連付けられている。

【０１３９】

FormatGuid

結果として得られる符号化されたビットストリームのフォーマットを表すＧＵＩＤである。Transform及びQuantizationのような符号化オペレーションの多くは、それらのオペレーションに対してフォーマット固有のエレメントを有する。これらのフォーマット固有のエレメントは、ＣＰＵによって十分な速度でハンドリングすることができるが、情報の交換は、余分なパイプラインステージの使用を必要とし、高いパイプライン効率の達成をより難しくする。

NumStreams

パイプライン構成に関連付けられている入力データストリーム及び出力データストリームの個数である。これは、多くの場合に、パイプラインＧＵＩＤによって暗に示される。

pConfiguration

静的構成プロパティへのポインタである。

pStreamDescription

１つのストリーム当たり１つの構造のアレイであり、そのストリームを通じて流れるデータを表す構造のアレイである。

SuggestedAllocatorProperties

コール側（コーデックアプリケーション）は、そのパイプライン設計に基づいて異なるストリームに関連付けられる或る個数のバッファ（サーフェス）を提案する。

pActualAllocatorProperties

ドライバは、収集することができる資源及び他の考慮事項に基づいて、実際のアロケータキューサイズを指定する。前提として、アプリケーションは、バッファリング（アロケータキューサイズ）が利用可能である効率的なパイプラインを構築できない場合に、こ

のインターフェースの使用を中止する。
ppEncode
出力されるエンコーダオブジェクトである。コール側は、これを、IUnknown::Releaseを介して解放される通常のＣＯＭオブジェクトとみなすべきである。
【０１４０】
戻り値
S_OK
関数は成功した。
E_FAIL
関数は（おそらく資源の不足のために）失敗した。
【０１４１】
３．５　データ構造：VideoEncoderService
３．５．１　VA2_Encode_MECaps
【０１４２】
【数３０】

```
typedef struct _VA2_Encode_MECaps {
    BOOL    VariableMBSizeSupported;
    BOOL    MotionVectorHintSupported;
    UINT16  MaxSearchWindowX;
    UINT16  MaxSearchWindowY
    UINT32  MaxImageWidth;
    UINT32  MaxImageHeight;
    UINT32  MaxMBSizeX;
    UINT32  MaxMBSizeY;
} VA2_Encode_MECaps;
```

【０１４３】
メンバー
VariableMBSizeSupported
１の値は、ハードウェアが、動き推定を行うときに可変マクロブロックサイズをサポートすることを示す。詳細には、可変マクロブロックサイズがサポートされる場合には、このＡＰＩのコール側が、VA2_Encode_MBDescription構造でConstantMBSizeを０にセットすること、及び、pMBRectanglesパラメータを使用して画像のパーティショニングを表すことは適法である。
MotionVectorHintSupported
１の値は、ハードウェアが、コール側からのうちのいくつかのヒントをその動き検索アルゴリズムで使用することができることを示す。詳細には、コール側は、Executeの構成パラメータであるVA2_Encode_ModeTypeのHintXメンバー及びHintYメンバーをセットすることができる。
MaxSearchWindowX
VA2_Encode_SearchBoundsのメンバーであるSearchWindowXの最大適法値である。VA2_Encode_SearchBoundsは、Motion Estimationの構成パラメータである。
MaxSearchWindowY
VA2_Encode_SearchBoundsのメンバーであるSearchWindowYの最大適法値である。VA2_Encode_SearchBoundsは、Motion Estimationの構成パラメータである。
MaxImageWidth
入力画像の最大許容幅である。
MaxImageHeight
入力画像の最大許容高さである。
MaxMBSizeX
マクロブロックの最大許容幅である。
MaxMBSizeY
マクロブロックの最大許容高さである。
【０１４４】

３．５．２　VA2_Encode_StaticConfiguration

【０１４５】

【数３１】

```
typedef struct _VA2_Encode_StaticConfiguration {
    GUID                        TransformOperator;
    GUID                        PixelInterpolation;
    GUID                        Quantization;
    UINT8                       NumMotionVectorsPerMB;

    VA2_Encode_MVLayout       MVLayout;
    VA2_Encode_ResidueLayout  ResLayout;
} VA2_Encode_StaticConfiguration;
```

【０１４６】

メンバー

TransformOperator

ＭＰＥＧ－２　ＤＣＴ、ＷＭＶ９　Transform等のうちの１つであるTransformオペレータを表すＧＵＩＤである。

PixelInterpolation

使用されるサブピクセル補間方式を表すＧＵＩＤである。双一次補間システム及び双三次補間システムは、フォーマット固有の多数の係数を有する。

Quantization

使用される量子化方式を表すＧＵＩＤである。

NumMotionVectorsPerMB

マクロブロックごとに計算されるMotion Vectorsの個数である。このインターフェースの初期のバージョンによってサポートされる簡単なパイプライン構成は、この値が１であることを必要とし得る。

MVLayout

Motion Vectorサーフェスのレイアウトである。

ResidueLayout

残差サーフェスのレイアウトである。

【０１４７】

３．５．３　VA2_Encode_Allocator

【０１４８】

【数３２】

```
typedef struct _VA2_Encode_Allocator {
    BOOL ExternalAllocator;
    UINT32 NumSurfaces;
} VA2_Encode_Allocator;
```

【０１４９】

メンバー

ExternalAllocator

Falseは、バッファがGetBufferを介して取得されることを示す一方、Trueは、バッファが外部アロケータを介して取得されるか、又は当該ストリームに関連付けられているアロケータが存在しないことを示す。パイプライン構成は、多くの場合、このフィールドの値を（多くの場合、０に）強制する。注目すべき例外は、外部アロケータから来ることが認められたInput Imageストリームにある。

NumSurfaces

アロケータキューに関連付けられるサーフェスの個数である。

【０１５０】

３．５．４　VA2_Encode_StreamDescritpion

【０１５１】

【数３３】

```
typedef struct _VA2_Encode_StreamDescription {
    UINT32 Length;
    UINT32 StreamType;
} VA2_Encode_StreamDescription;
```

【０１５２】

メンバー

Length

タイプキャストの有効性を確認すると共に、拡張性を可能にするのに使用される構造全体の長さである。

StreamType

このストリームに関連付けられるデータのタイプを表す。タイプキャストに使用される。

【０１５３】

備考

この基本構造は、StreamTypeフィールドにおける派生したタイプにタイプキャストされる。タイプキャストは、VA2_Encode_StreamTypeのドキュメンテーションに記載されている。

【０１５４】

３．５．５　VA2_Encode_StreamType

【０１５５】

【数３４】

```
#define VA2_Encode_StreamType_Video        0x1
#define VA2_Encode_StreamType_MV           0x2
#define VA2_Encode_StreamType_Residues     0x3
```

【０１５６】

タイプ記述

VA2_Encode_StreamType_Video

関連付けられるストリーム記述構造は、VA2_Encode_StreamDescription_Videoにキャストすることができる。

VA2_Encode_StreamType_MV

関連付けられるストリーム記述構造は、VA2_Encode_StreamDescription_MVにキャストすることができる。

VA2_Encode_StreamType_Residues

関連付けられるストリーム記述構造は、VA2_Encode_StreamDescription_Residuesにキャストすることができる。

【０１５７】

３．５．６　VA2_Encode_StreamDescription_Video

【０１５８】

【数３５】

```
typedef struct _VA2_Encode_StreamDescription {
    UINT32 Length;
    UINT32 StreamType;
    VA2_VideoDesc VideoDescription;
} VA2_Encode_StreamDescription;
```

【０１５９】

メンバー

Length

タイプキャストの有効性を確認すると共に拡張性を可能にするのに使用される構造全体の長さである。

StreamType

このストリームに関連付けられるデータのタイプを表す。タイプキャストに使用され

る。
VideoDescription
次元、フレームレート、原色等を含むビデオストリームのさまざまなプロパティを表す。
【０１６０】
３．５．７　VA2_Encode_StreamDescription_MV
【０１６１】
【数３６】
```
typedef struct _VA2_Encode_StreamDescription {
    UINT32 Length;
    UINT32 StreamType;
    VA2_Encode_MVType MVType;
    VA2_Encode_MVLayout MVLayout;
} VA2_Encode_StreamDescription;
```
【０１６２】
メンバー
Length
タイプキャストの有効性を確認すると共に拡張性を可能にするのに使用される構造全体の長さである。
StreamType
このストリームに関連付けられるデータのタイプを表す。タイプキャストに使用される。
MVType
動きデータを返すのに使用されるMotion Vector構造のタイプを表す。Motion Vectorサーフェスのコンテンツの解釈において使用される。
MVLayout
所与の入力画像のMotion Vector構造がメモリにどのように配置されるのかを表す。
【０１６３】
３．５．８　VA2_Encode_StreamDescription_Residues
【０１６４】
【数３７】
```
typedef struct _VA2_Encode_StreamDescription {
    UINT32 Length;
    UINT32 StreamType;
    VA2_Encode_SamplingType SamplingType;
    UINT32 LumaWidth;
    UINT32 LumaHeight;
    UINT32 ChromaCbWidth;
    UINT32 ChromaCbHeight;
    UINT32 ChromaCrWidth;
    UINT32 ChromaCrHeight;
} VA2_Encode_StreamDescription;
```
【０１６５】
メンバー
Length
タイプキャストの有効性を確認すると共に拡張性を可能にするのに使用される構造全体の長さである。
StreamType
このストリームに関連付けられるデータのタイプを表す。タイプキャストに使用される。
SamplingType
残差データが４：４：４、４：２：２等であるか否かを指定する。
LumaWidth
ルマサーフェスの幅である。
LumaHeight
ルマサーフェスの高さである。

ChromaCbWidth
　Ｃｂ残差値を含むサーフェスの幅である。
ChromaCbHeight
　Ｃｂ残差値を含むサーフェスの高さである。
ChromaCrWidth
　Ｃｒ残差値を含むサーフェスの幅である。
ChromaCrHeight
　Ｃｒ残差値を含むサーフェスの高さである。

【０１６６】

３．６　データ構造：Motion Vectors

３．６．１　Motion Vectorレイアウト

図９は、一実施形態によるＤ３Ｄサーフェスに記憶されている例示的な動きベクトルデータを示している。「ＭＶ」として表されたセルのそれぞれは、Motion Vector構造である。VA2_Encode_MVType及びVA2_Encode_MVLayoutの値に応じて異なる表現が使用される。実際の構造及びレイアウトは、以下に記載されている。

【０１６７】

３．６．２　新しいD3D Format（Ｄ３Ｄフォーマット）

【０１６８】

【数３８】

```
typedef enum _D3DFORMAT
{

    D3DFMT_MOTIONVECTOR16 = 105,
    D3DFMT_MOTIONVECTOR32 = 106,
    D3DFMT_RESIDUE16      = 107,

} D3DFORMAT;
```

【０１６９】

Motion Vectors Surfaces及びResidue Surfacesは、個々のMotion Vectors及びResiduesのサイズを示す上記の新しいD3D Formatタイプに関連付けられている。このサイズ情報は、アプリケーションが、提供されたサーフェス又は資源作成ＡＰＩのうちの一方を使用してサーフェスを作成する時にドライバによって使用される。符号化されたサーフェスに関連付けられる資源フラグはVA2_EncodeBufferである。

【０１７０】

【数３９】

```
// Buffer Type
enum
{
    VA2_EncodeBuffer = 7,
};

typedef struct _D3DDDI_RESOURCEFLAGS
{
    union
    {
        struct
        {
            UINT    TextApi                 : 1;    // 0x20000000
            UINT    EncodeBuffer            : 1;    // 0x40000000
            UINT    Reserved                : 1;    // 0x80000000
        };
        UINT        Value;
    };
} D3DDDI_RESOURCEFLAGS;
```

【０１７１】

３．６．３　VA2_Encode_MVSurface

この構造は、IDirect3DSurface9から効率的に導出され、組み込まれたＤ３Ｄサーフェスのコンテンツを解釈することを可能にする状態情報を保持する。

【０１７２】

【数４０】

```
typedef struct _VA2_Encode_MVSurface {
    IDirect3DSurface9*      pMVSurface;
    VA2_Encode_MVType       MVType;
    VA2_Encode_MVLayout     MVLayout;
    GUID                    DistanceMetric;
} VA2_Encode_MVSurface;
```

【０１７３】

メンバー

pMVSurface

Motion Vectorsを含むD3D Surfaceへのポインタである。

MVType

この値は、構造（VA2_Encode_MotionVector8等）を用いて個々の動きベクトルを解釈するために、当該構造を特定するのに使用される。

MVLayout

この値は、個々のMotion Vector構造がD3D Surfaceにどのように配置されているのかを特定する。

DistanceMetric

２つのマクロブロック間の距離を測定するのに使用される距離メトリックを表すＧＵＩＤである。この距離メトリックは、最も近いマクロブロックを特定するのに使用され、したがって、最適な動きベクトルを特定するのに使用される。

【０１７４】

３．６．４　VA2_Encode_MVType

この列挙値は、Motion Vector D3D9 Surfaceのコンテンツを復号するのに使用される。Motion Vectorのタイプに応じて、いくつかの異なるMotion Vector構造のうちの１つが、そのサーフェスのコンテンツを解釈するのに使用される。

【０１７５】

【数４１】

```
typedef enum {
    VA2_Encode_MVType_Simple8,
    VA2_Encode_MVType_Simple16,
    VA2_Encode_MVType_Extended8,
    VA2_Encode_MVType_Extended16
} VA2_Encode_MVType;
```

【０１７６】

説明

VA2_Encode_MVType_Simple8

Motion Vector構造がVA2_Encode_MotionVector8である。

VA2_Encode_MVType_Simple16

Motion Vector構造がVA2_Encode_MotionVector16である。

VA2_Encode_MVType_Extended8

Motion Vector構造がVA2_Encode_MotionVectorEx8である。

VA2_Encode_MVType_ Extended16

Motion Vector構造がVA2_Encode_MotionVectorEx16である。

【０１７７】

３．６．５　VA2_Encode_MVLayout

【０１７８】

【数４２】

```
typedef enum {
    VA2_Encode_MVLayout_A,
    VA2_Encode_MVLayout_B,
    VA2_Encode_MVLayout_C
} VA2_Encode_MVLayout;
```

【０１７９】

説明

Type A

実際のＤ３Ｄサーフェスは、Macroblock Index（マクロブロックインデックス）及びRow Index（行インデックス）によってインデックスされるMotion Vector構造のアレイである。

Type B

これは、Macroblock ごとのMotion Vectorsの個数が一定でないパックされたレイアウトである。ＴＢＤを詳述する。

Type C

【０１８０】

３．６．６　VA2_Encode_MotionVector8

【０１８１】

【数４３】

```
typedef struct _VA2_Encode_MotionVector8 {
    INT8    x;
    INT8    y;
} VA2_Encode_MotionVector8;
```

【０１８２】

メンバー

x

Motion Vectorのｘ座標である。

y

Motion Vectorのｙ座標である。

【０１８３】

３．６．７　VA2_Encode_MotionVector16

【０１８４】

【数４４】

```
typedef struct _VA2_Encode_MotionVector16 {
    INT16    x;
    INT16    y;
} VA2_Encode_MotionVector16;
```

【０１８５】

メンバー

x

Motion Vectorのｘ座標である。

y

Motion Vectorのｙ座標である。

【０１８６】

３．６．８　VA2_Encode_MotionVectorEx8

【０１８７】

【数４５】

```
typedef struct _VA2_Encode_MotionVectorEx8 {
    INT8    x;
    INT8    y;
    UINT8   ImageIndex;
    UINT8   Distance;
} VA2_Encode_MotionVectorEx8;
```

【０１８８】

メンバー

x

Motion Vectorのｘ座標である。

y

Motion Vectorのｙ座標である。

ImageIndex
ComputeMotionVectors（動きベクトル計算）のコールで提供された基準画像のリスト内への０から始まるインデックスである。
Distance
測定の単位は、VA_Encode_MVSurfaceのDistanceMetricフィールドによって指定される。これは、現在のマクロブロックと、実際の動きベクトル（ｘ，ｙ）によって参照される基準マクロブロックとの距離を測定する。

【０１８９】

３．６．９　VA2_Encode_MotionVectorEx16

【０１９０】

【数４６】

```
typedef struct _VA2_Encode_MotionVectorEx16 {
    INT16   x;
    INT16   y;
    UINT16  ImageIndex;
    UINT16  Distance;
} VA2_Encode_MotionVectorEx16;
```

【０１９１】

メンバー
ｘ
Motion Vectorのｘ座標である。
ｙ
Motion Vectorのｙ座標である。
ImageIndex
ComputeMotionVectorsのコールで提供された基準画像のリスト内への０から始まるインデックスである。
Distance
測定の単位は、VA_Encode_MVSurfaceのDistanceMetricフィールドによって指定される。これは、現在のマクロブロックと、実際の動きベクトル（ｘ，ｙ）によって参照される基準マクロブロックとの距離を測定する。

【０１９２】

３．７　データ構造：Residues
残差サーフェスは、２バイト長の符号付き整数値のアレイである。換言すれば、符号付き整数は、タイプＩＮＴ１６である。この方式は、実際に重要となるすべての場合に十分であると考えられる。たとえば、ＭＰＥＧ－２は９ビットの残差値を扱い、Ｈ．２６４は１２ビットの残差を扱う。また、元のデータがＹＵＹ２であった場合に、ルマ値は、それぞれ１バイトを占有し、したがって、残差は９ビット（０－２５５＝－２５５）を使用する。さらに、ＤＣＴタイプの変換を適用することによって、所要データは、１つの残差値当たり１１ビットに増加する。これらの場合のすべてが、２バイト長の符号付き残差値を使用することによって十分にハンドリングされる。

【０１９３】

残差サーフェスの幅は、１ラインの残差値の個数である。たとえば、４：２：２のサンプリングによる６４０×４８０のプログレッシブ画像は、１ライン当たり６４０個のルマ値及び３２０個のクロマ値を有する。関連付けられるルマサーフェスのサイズは、６４０×４８０×２バイトであり、クロマサーフェスのサイズは３２０×４８０×２バイトである。

【０１９４】

Residue Surfacesは、D3DFMT_RESIDUE16フォーマットフラグ及びVA2_EncodeBuffer資源タイプを使用して作成される。

【０１９５】

３．７．１　ルマ平面
図１０は、ルマサーフェスの幅が元のＹＣｂＣｒ画像と一致することを示す１つの例示

的な図を示している。たとえば、６４０×４８０画像は、１ライン当たり４８０個のルマ値を有し、したがって、ルマサーフェスの幅は４８０である。したがって、ルマサーフェスのサイズは、６４０×４８０×２バイトである。

Plane = VA2_Encode_Residue_Y

Sampling = VA2_Encode_SamplingType_*

【０１９６】

３．７．２　クロマ４：２：２

図１１は、一実施形態による、ビデオの１ライン当たりの残差値の個数が、元のビデオ画像の幅の半分であることを示す１つの例示的な図を示している。したがって、６４０×４８０画像では、１ライン当たりの残差値の個数、したがって、サーフェスの幅は、３２０である。

Plane = VA2_Encode_Residue_U又はVA_Encode_Residue_V

Sampling = VA2_Encode_SamplingType_422

【０１９７】

３．７．３　クロマ４：２：０

このシナリオでは、残差サーフェスの幅は、元のプログレッシブフレームの幅の２分の１であり、高さも同様に２分の１である。したがって、６４０×４８０画像では、クロマサーフェス自体は、幅が３２０となり、長さが２４０となる。

Plane = VA2_Encode_Residue_U又はVA_Encode_Residue_V

Sampling = VA2_Encode_SamplingType_420

【０１９８】

４　例示的なＤＤＩドキュメンテーション

Extension Devices（拡張デバイス）は、既存のVideo Decoder（ビデオデコーダ）機能及びVideo Processor（ビデオプロセッサ）機能のほかに新たな機能を追加するためにVA Interfacesによって提供される通過メカニズムである。たとえば、それらのExtension Devicesは、新しいVideo Encoder機能をサポートするのに使用される。

【０１９９】

Extension Devicesは、アプリケーションがドライバへ／ドライバからデータを送受信することができる、タイプが定められていないファネル（untyped funnel）のように機能する。このデータの意味は、ＶＡスタックに知られておらず、CreateExtensionDeviceコールのpGuidパラメータ及びExtensionExecuteのFunctionパラメータに基づいてドライバによって解釈される。

【０２００】

VA Encodeは、（IVideoEncoderのuuidと同じ）以下のＧＵＩＤ値を使用する。

【０２０１】

【数４７】

```
{7AC3D93D-41FC-4c6c-A1CB-A875E4F57CA4}
DEFINE_GUID(VA_Encoder_Extension, 0x7ac3d93d, 0x41fc, 0x4c6c, 0xa1, 0xcb, 0xa8, 0x75, 0xe4,
0xf5, 0x7c, 0xa4);
```

【０２０２】

４．１　列挙及び機能

Extension Devicesは、タイプパラメータがGETEXTENSIONGUIDCOUNT又はGETEXTENSIONGUIDSにセットされているFND3DDDI_GETCAPSを使用して列挙される。コーデックアプリケーションは、GETEXTENSIONGUIDSによって返された拡張ＧＵＩＤのリストにおいてVA_Encoder_Extensionを探し、VA Encodeサポートが利用可能であるか否かを判断する。

【０２０３】

４．１．１　FND3DDDI_GETCAPS

【０２０４】

【数４８】

```
typedef HRESULT
(APIENTRY *PFND3DDI_GETCAPS)
(
    HANDLE hAdapter,
    CONST D3DDDIARG_GETCAPS*
);
```

【０２０５】

拡張デバイス（Encoderデバイス）の機能に関して問い合わせる時、GETEXTENSIONCAPSが、D3DDDIARG_GETCAPS構造のpInfoとして以下の構造と共に使用される。

【０２０６】

４．１．２　VADDI_ QUERYEXTENSIONCAPSINPUT

【０２０７】

【数４９】

```
typedef struct _VADDI_QUERYEXTENSIONCAPSINPUT
{
    CONST GUID*              pGuid;
    UINT                     CapType;
    VADDI_PRIVATEDATA*     pPrivate;
} VADDI_QUERYEXTENSIONCAPSINPUT;
```

【０２０８】

VADDI_QUERYEXTENSIONCAPSINPUTのpGuidパラメータは、VA_Encoder_Extensionにセットされる。

【０２０９】

【数５０】

```
#define VADDI_Encode_Captype_Guids           VADDI_EXTENSION_CAPTYPE_MIN
#define VADDI_Encode_Captype_DistanceMetrics VADDI_EXTENSION_CAPTYPE_MIN + 1
#define VADDI_Encode_Captype_SearchProfiles  VADDI_EXTENSION_CAPTYPE_MIN + 2
#define VADDI_Encode_Captype_MECaps          VADDI_EXTENSION_CAPTYPE_MIN + 3
```

【０２１０】

GETEXTENSIONCAPSの出力は、D3DDDIARG_GETCAPSのpDataパラメータにカプセル化される。pDataパラメータは、以下のように解釈される。

Captype_Guids：Type = (GUID*).DataSize = sizeof(GUID)*guid_count.

Captype_DistanceMetrics：Type = (GUID*).DataSize = sizeof(GUID)*guid_count.

Captype_SearchProfiles：Type = (VADDI_Encode_SearchProfile*).DataSize = sizeof(VADDI_Encode_SearchProfile).

Captype_MECaps：Type = (VADDI_Encode_MECaps).DataSize = sizeof(VADDI_Encode_MECaps).

【０２１１】

４．１．３　D3DDDI_CREATEEXTENSIONDEVICE

実際の作成は、D3DDDI_CREATEEXTENSIONDEVICEコールを介して行われる。このプライマリ引数は、以下に示される。

【０２１２】

【数５１】

```
typedef struct _D3DDDIARG_CREATEEXTENSIONDEVICE
{
    CONST GUID*                pGuid;
    VADDI_PRIVATEDATA*       pPrivate;
    HANDLE                     hExtension;
} D3DDDIARG_CREATEEXTENSIONDEVICE;
```

【０２１３】

４．２　符号化機能

実際の拡張ユニット関数は、D3DDDI_EXTENSIONEXECUTEコールを介して起動される。Extension Unit（拡張ユニット）のインスタンスは、ＧＵＩＤに既に関連付けられ、したがって、拡張ユニットのタイプは、実行コールが行われたときに既に知られている。唯一の追加パラメータは、実行する特定のオペレーションを示すFunction（関数）である。たと

えば、タイプEncoderのExtension Deviceは、その関数のうちの１つとしてMotionEstimationをサポートすることができる。通常、Extension Deviceは、Extension Deviceの機能を列挙する、それ自身のGetCaps関数を有する。

【０２１４】

【数５２】

```
typedef struct _D3DDDIARG_EXTENSIONEXECUTE
{
    HANDLE                      hExtension;
    UINT                        Function;
    VADDI_PRIVATEDATA*          pPrivateInput;
    VADDI_PRIVATEDATA*          pPrivateOutput;
    UINT                        NumBuffers;
    VADDI_PRIVATEBUFFER*        pBuffers;
} D3DDDIARG_EXTENSIONEXECUTE;
```

【０２１５】

pBuffersパラメータは、VA Encodeによって使用されず、予備パラメータとみなされるべきである。Functionパラメータは、VA Encoderに関して以下の値を取る。

【０２１６】

【数５３】

```
#define VADDI_Encode_Function_Execute 1
```

【０２１７】

D3DDDIARG_EXTENSIONEXECUTE のpPrivateInputパラメータ及びpPrivateOutputパラメータは、Execute APIコールのパラメータをカプセル化するのに使用される。以下の入力パラメータ及び出力パラメータに組み込まれた符号化特有のパラメータは、ＡＰＩランドからＤＤＩランドへまだマッピングされていない。しかし、それは、リネームの問題にすぎず、単一の定義で管理することができる場合がある。

【０２１８】

４．２．１　VADDI_Encode_Function_Execute_Input

このパラメータは、Execute APIコールの入力パラメータを含む。

【０２１９】

【数５４】

```
typedef struct _VADDI_Encode_Function_Execute_Input
{
    UINT32 NumDataParameters;
    VA2_Encode_ExecuteDataParameter** pData;
    UINT32 NumConfigurationParameters;
    VA2_Encode_ExecuteConfigurationParameter** pConfiguration;
} VADDI_Encode_Function_Execute_Input;
```

【０２２０】

４．２．２　VADDI_Encode_Function_Execute_Output

この構造は、Executeコールからの出力データをカプセル化する。

【０２２１】

【数５５】

```
typedef struct _VADDI_Encode_Function_Execute_Output
{
    UINT32 NumDataParameters;
    VA2_Encode_ExecuteDataParameter** pData;
} VADDI_Encode_Function_Execute_Output;
```

【０２２２】

４．３　拡張デバイス構造

以下のセクションは、VA Extensionメカニズムに関連付けられているさまざまな構造及び関数コールバックを記載している。

【０２２３】

４．３．１　VADDI_PRIVATEBUFFER

【０２２４】

【数５６】

```
typedef struct _VADDI_PRIVATEBUFFER
{
    HANDLE                  hResource;
    UINT                    SubResourceIndex;
    UINT                    DataOffset;
    UINT                    DataSize;
} VADDI_PRIVATEBUFFER;

typedef struct _VADDI_PRIVATEDATA
{
    VOID*             pData;
    UINT              DataSize;
} VADDI_PRIVATEDATA;
```

【０２２５】

４．３．２　D3DDDIARG_EXTENSIONEXECUTE

【０２２６】

【数５７】

```
typedef struct _D3DDDIARG_EXTENSIONEXECUTE
{
    HANDLE                          hExtension;
    UINT                            Function;
    VADDI_PRIVATEDATA*            pPrivateInput;
    VADDI_PRIVATEDATA*            pPrivateOutput;
    UINT                            NumBuffers;
    VADDI_PRIVATEBUFFER*          pBuffers;
} D3DDDIARG_EXTENSIONEXECUTE;

typedef HRESULT
 (APIENTRY *PFND3DDDI_CREATEEXTENSIONDEVICE)
(
    HANDLE hDevice,
    D3DDDIARG_CREATEEXTENSIONDEVICE*
);
```

【０２２７】

hDeviceパラメータは、Ｄ３Ｄ９デバイスを参照し、これは、D3DDDI_CREATEDEVICEのコールを使用して作成される。

【０２２８】

４．３．３　FND3DDDI_DESTROYEXTENSIONDEVICE

【０２２９】

【数５８】

```
typedef HRESULT
(APIENTRY *PFND3DDDI_DESTROYEXTENSIONDEVICE)
(
    HANDLE hDevice,
    HANDLE hExtension
);
```

【０２３０】

４．３．４　FND3DDDI_EXTENSIONEXECUTE

【０２３１】

【数５９】

```
typedef HRESULT
(APIENTRY *PFND3DDDI_EXTENSIONEXECUTE)
(
    HANDLE hDevice,
    CONST D3DDDIARG_EXTENSIONEXECUTE*
);
```

【０２３２】

４．３．５　D3DDDI_DEVICEFUNCS

【０２３３】

【数６０】

```
typedef struct _D3DDDI_DEVICEFUNCS
{
    PFND3DDDI_CREATEEXTENSIONDEVICE        pfnCreateExtensionDevice;
    PFND3DDDI_DESTROYEXTENSIONDEVICE       pfnDestroyExtensionDevice;
    PFND3DDDI_EXTENSIONEXECUTE             pfnExtensionExecute;
} D3DDDI_DEVICEFUNCS;
```

【０２３４】

４．４　Ｄ３Ｄ９構造及び関数

以下のＤ３Ｄ構造及びコールバックは、拡張デバイスの機能を取得するための一般的なＤ３Ｄメカニズムを表している。

【０２３５】

【数６１】

```
typedef enum _D3DDDICAPS_TYPE
{
    D3DDDICAPS_GETEXTENSIONGUIDCOUNT          =31,
    D3DDDICAPS_GETEXTENSIONGUIDS              =32,
    D3DDDICAPS_GETEXTENSIONCAPS               =33,
} D3DDDICAPS_TYPE;

typedef struct _D3DDDIARG_GETCAPS
{
    D3DDDICAPS_TYPE     Type;
    VOID*               pInfo;
    VOID*               pData;
    UINT                DataSize;
} D3DDDIARG_GETCAPS;
```

【０２３６】

５　例示的なプログラミングモデル

５．１　パイプライン効率

最大効率を達成するために、エンコーダアプリケーションは、ＣＰＵ及びグラフィックスハードウェアの双方が十分に利用されるように構造化される。したがって、Motion Estimationは、或るフレームに関して進行中である間、異なるフレームに対してQuantization Step（量子化ステップ）を実行するのに有益なものとなり得る。

ハードウェアのフル利用の取得は、マルチスレッド化されたエンコーダで容易になる。

【０２３７】

５．１．１　例：単一の動きベクトル（パイプラインフル）

以下の（擬似コードによる）２スレッドアプリケーションは、エンコーダが２ステージソフトウェアパイプラインを実施する１つの方法を示し、VA Encodeインターフェースを有効に使用する方法に関してのうちのいくつかのガイドラインを提供する。詳細には、この２スレッドアプリケーションは、ソフトウェアスレッドで見られるようなｋ＝AllocatorSizeのバッファリングを実施する。これは、ハードウェア要求のサブミットに非同期性が存在することを考慮したものである。すなわち、ハードウェアスレッドは要求をサブミットする一方、ソフトウェアスレッドは、しばらくしてからその結果をピックアップして処理する。

【０２３８】

【数６２】

```
HardwareThread()
{
    while (Streaming)
    {
        LoadFrame(ppInputBuffer[n]);
        Codec->ProcessInput(ppInputBuffer[n]); // blocking GetBuffer + Execute
    }
}

SoftwareThread()
{
    k = AllocatorSize();

    while (Streaming)
    {
        // k represents the buffer between pipeline stages
        Codec->ProcessOutput(ppOutputBuffer[n-k]); // Wait, ReleaseBuffer
        VLE();
        Bitstream();
    }
}
```

【０２３９】

上記ProcessInput（プロセス入力）は、Execute及びGetBufferを取り囲むラッパとみなすことができる一方、ProcessOutput（プロセス出力）は、その後に適切なReleaseBufferコールが続く実行イベントのWaitを取り囲むラッパとみなすことができる。

【０２４０】

パイプラインステージ間のバッファを表すパラメータｋを決定する方法は明らかではない。これは、アロケータサイズを示し、開始ポイントとして、CodecとVA Encoderオブジェクトとの間のアロケータ取り決めで使用される同じ値（キュー長）を使用することができる。ｋがアロケータサイズよりも大きい場合には、ProcessInputコールは、ｋ個のバッファが使用される前であっても、何らかの方法でブロックする可能性がある。

【０２４１】

アプリケーションの目的は、ProcessOutputでブロックすることなく、SoftwareThreadで費やされる時間を最大にすることであるべきである。換言すれば、アプリケーションは、時間のほとんどをVLE()関数及びBitsream()関数で動作しているべきである。ハードウェアが非常に低速である場合、ProcessOutput()は、「ｋ」のアロケータサイズにもかかわらずブロックする。ソフトウェアは、常に「先行」している。上記パイプラインは、ハードウェアがバッファの処理に要する時間が、ソフトウェアがVLE及びBitstreamを実行するのに要する時間とほぼ同じである限りにおいてのみ効率的である。「ｋ」のバッファリングが達成するものは、ジッタ用のパッドのみである。

【０２４２】

以下のコードフラグメントは、GetBuffer及びReleaseBufferの概略の一実施態様を示している。

【０２４３】

【数６３】

```
IVideoEncoder::GetBuffer(Type, ppBuffer, Blocking)
{
    if (Empty)
    {
        if (Blocking) Wait(NotEmptyEvent);
        else return STATUS_EMPTY;
    }

    *ppBuffer = pQueue[Type][Head];
    Head++;
    if (Head == Tail)
    {
        Empty = 1;
        ResetEvent(NotEmptyEvent);
    }

    return S_OK;
}

IVideoEncoder::ReleaseBuffer(Type, pBuffer)
{
    if ((Tail == Head) && !Empty) return STATUS_FULL;

    pQueue[Type][Tail] = pBuffer;
    Tail++;

    if (Empty)
    {
        Empty = false;
        SetEvent(NotEmptyEvent);
    }

    return S_OK;
}
```

【０２４４】

以下は、ProcessInput及びProcessOutputのコーデックの実施態様を略述している。

【０２４５】

【数６４】

```
// this implementation is blocking contrary to normal semantics
Codec::ProcessInput(IMediaBuffer pInput)
{

    GetBuffer(TypeUncompressed, pYUVBuffer, true);
    GetBuffer(TypeMotionVector, pMVBuffer, true);
    GetBuffer(TypeResidues, pResidueBuffer, true);

    memcpy(pYUVBuffer, pInput.Image);
    Execute(pYUVBuffer, pMVBuffer, pResidueBuffer, pEvent);

    CodecQueue.Enqueue(pYUVBuffer, pMVBuffer, pResidueBuffer, pEvent);
}

Codec::ProcessOutput(IMediaBuffer pOutput)
{
    if (CodecQueue.Empty())
    {
        pOutput.dwFlags = DMO_OUTPUT_DATABUFFERF_INCOMPLETE;
        return S_FALSE;
    }
    CodecQueue.Dequeue(pYUVBuffer, pMVBuffer, pResidueBuffer, pEvent);

    Wait(pEvent);

    memcpy(pOutput.MVBuffer, pMVBuffer);
    memcpy(pOutput.ResidueBuffer, pResidueBuffer);

    ReleaseBuffer(TypeUncompressed, pYUVBuffer);
    ReleaseBuffer(TypeMotionVector, pMVBuffer);
    ReleaseBuffer(TypeResidues, pResidueBuffer);

    return S_OK;
}
```

【０２４６】

これは、標準的な非ブロッキングのCodec::ProcessInputの代替的な一実施態様である。

【０２４７】
【数６５】

```
Codec::ProcessInput(IMediaBuffer pInput)
{

    if (GetBuffer(TypeUncompressed, pYUVBuffer, false) == STATUS_EMPTY)
    {
        return DMO_E_NOTACCEPTING;
    }
    if (GetBuffer(TypeMotionVector, pMVBuffer, false) == STATUS_EMPTY)
    {
        return DMO_E_NOTACCEPTING;
    }
    if (GetBuffer(TypeResidues, pResidueBuffer, false) == STATUS_EMPTY)
    {
        return DMO_E_NOTACCEPTING;
    }

    memcpy(pYUVBuffer, pInput.Image);
    Execute(pYUVBuffer, pMVBuffer, pResidueBuffer, pEvent);

    CodecQueue.Enqueue(pYUVBuffer, pMVBuffer, pResidueBuffer, pEvent);
}
```

【０２４８】
５．１．２　例：複数の動きベクトル
　このセクションでは、エンコーダが、ハードウェアから複数の動きベクトルを要求し、さまざまなパラメータに基づいて１つを選び、処理のためにそれらの動きベクトルを再サブミットする、より複雑なパイプラインを見る。以下のコードは、引き続きこれまでのように２ステージパイプラインを単純に使用し、複数の動きベクトルを要求し、最もよいものを再サブミットする。これには、本来備わっている直列化が含まれている。
【０２４９】
【数６６】

```
HardwareThread()
{
    while (Streaming)
    {
        LoadFrame(ppInputBuffer[n]);
        ProcessInput(ppInputBuffer[n]);
        ProcessOutput(ppOutputBuffer[n]);

        SelectMV(ppOutputBuffer[n], ppOutputBuffer2[n]);
        ProcessInput2(ppOutputBuffer2[n]);
        n++;
    }
}

SoftwareThread()
{
    while (Streaming)
    {
        ProcessOutput2(ppOutputBuffer2[n - k]);
        VLE(ppOutputBuffer2[n - k]);
        Bitstream(ppOutputBuffer2[n - k]);
    }
}
```

【０２５０】
　上記例では、ソフトウェアは、ProcessOutput及びProcessOutput2においてその時間の半分でブロックされ、パイプライン効率は明らかに悪くなる。他方、ＣＰＵ利用は非常に低く、全体のスループットは加速されていない符号化よりもなお高い。３つのスレッドに基づく３ステージパイプラインが、以下のように、直列化の問題を解決する。
【０２５１】

【数６７】

```
HardwareThread1()
{
    while (Streaming)
    {
        LoadFrame(ppInputBuffer[n]);
        ProcessInput(ppInputBuffer[n]);
    }
}

HardwareThread2()
{
    while (Streaming)
    {
        ProcessOutput(ppOutputBuffer[n - k1]);
        SelectMV(ppOutputBuffer[n - k1], ppOutputBuffer2[n - k1]);
        ProcessInput2(ppOutputBuffer2[n - k1]);
    }
}

SoftwareThread()
{
    while (Streaming)
    {
        ProcessOutput2(ppOutputBuffer2[n - k1 - k2]);
        VLE(ppOutputBuffer2[n - k1 - k2]);
        Bitstream(ppOutputBuffer2[n - k1 - k2]);
    }
}
```

【０２５２】

３つのパイプラインステージが存在するので、２つのハードウェアステージ間をパッドするために、追加バッファが加えられる。したがって、２つの値ｋ１及びｋ２がある。

【図面の簡単な説明】

【０２５３】

【図１】一実施形態による加速ビデオ符号化の１つの例示的なシステムを示す図である。

【図２】符号化プロセスのうちのいくつかがハードウェアで加速されるビデオ符号化パイプライン構成の１つの例示的な実施形態を示す図である。

【図３】一実施形態による加速ビデオ符号化の１つの例示的なプロシージャを示す図である。

【図４】付録における、一実施形態による、ビデオ符号化加速アプリケーションプログラミングインターフェースを利用することができる方法を示すための１つの例示的なビデオエンコーダアプリケーションを示す図である。

【図５】付録における、一実施形態による、加速ハードウェアが、動き推定、変換、量子化、及びそれらの逆のプロセスを加速して、符号化された画像を生成する１つの例示的なビデオ符号化パイプライン構成を示す図である。

【図６】付録における、一実施形態による、ハードウェアが動き推定のみを加速する１つの例示的なビデオ符号化パイプライン構成を示す図である。

【図７】付録における、一実施形態によるいくつかの例示的な動き推定パラメータを示す図である。

【図８】付録における、一実施形態によるDisplay 3-Dimensional（Ｄ３Ｄ）サーフェスに記憶されている例示的な動きベクトルデータを示す図である。

【図９】付録における、一実施形態による、ルマサーフェスの幅が元のＹＣｂＣｒ画像と一致することを示す１つの例示的な図である。

【図１０】付録における、一実施形態による、ビデオの１ライン当たりの残差値の個数が、元のビデオ画像の幅の２分の１であることを示す１つの例示的な図である。

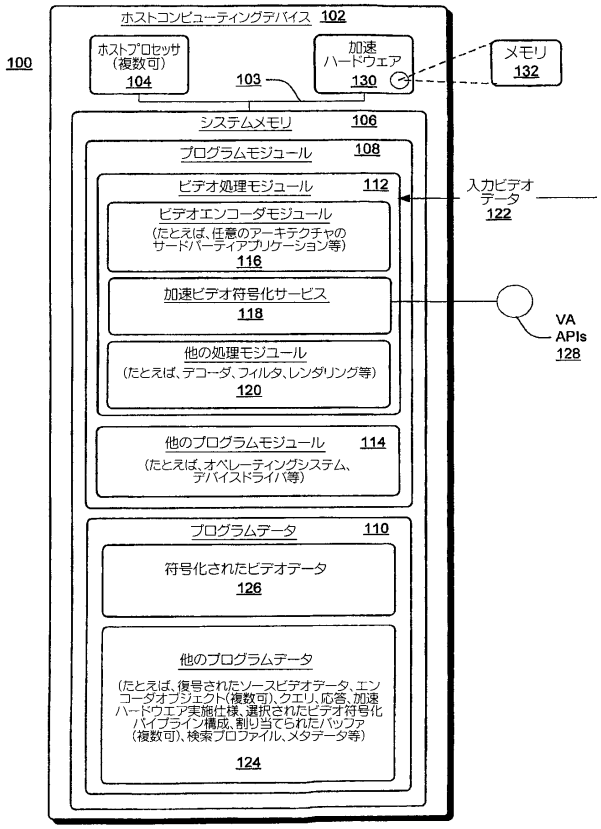
【図１１】付録における、一実施形態による、残差サーフェスの幅が、元のプログレッシブフレームの幅の４分の１であることを示す１つの例示的な図である。

【符合の説明】

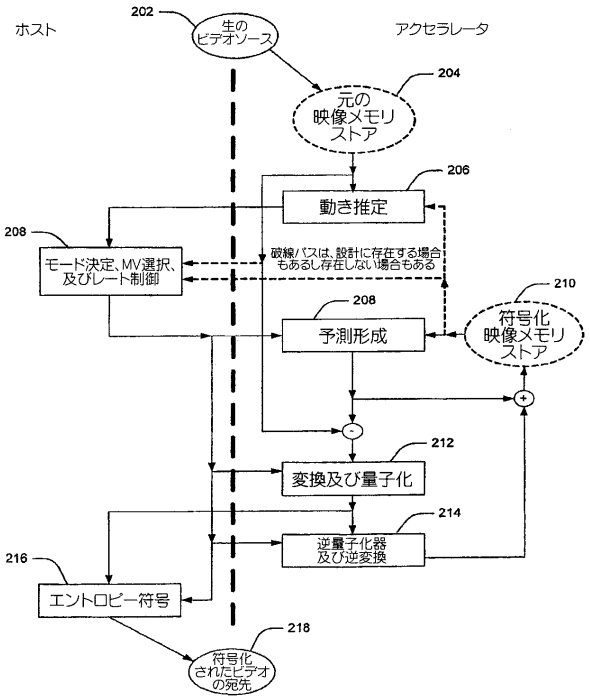
【０２５４】

図において、コンポーネントの参照符号の左端の桁は、そのコンポーネントが最初に登場する特定の図を識別するものである。

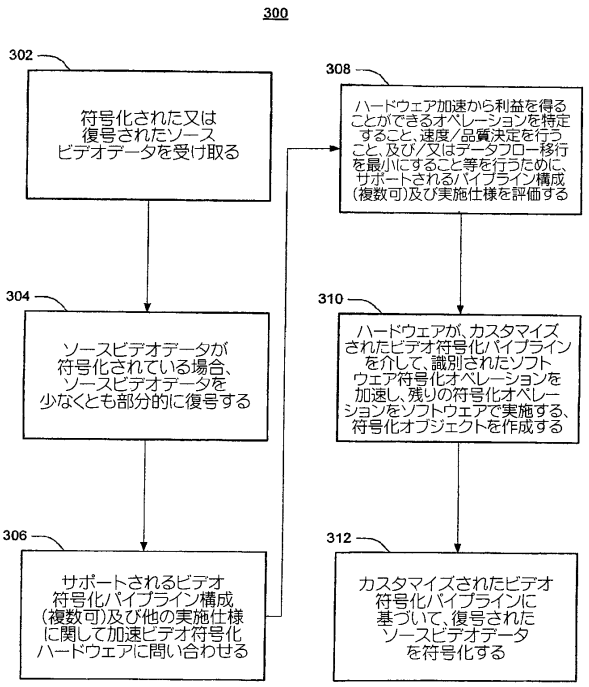
【図１】


100
ホストコンピューティングデバイス 102
ホストプロセッサ（複数可） 104
103
加速ハードウェア 130
メモリ 132
システムメモリ 106
プログラムモジュール 108
ビデオ処理モジュール 112
ビデオエンコーダモジュール（たとえば、任意のアーキテクチャのサードパーティアプリケーション等） 116
加速ビデオ符号化サービス 118
他の処理モジュール（たとえば、デコーダ、フィルタ、レンダリング等） 120
他のプログラムモジュール 114（たとえば、オペレーティングシステム、デバイスドライバ等）
プログラムデータ 110
符号化されたビデオデータ 126
他のプログラムデータ（たとえば、復号されたソースビデオデータ、エンコーダオブジェクト（複数可）、クエリ、応答、加速ハードウェア実施仕様、選択されたビデオ符号化パイプライン構成、割り当てられたバッファ（複数可）、検索プロファイル、メタデータ等） 124
入力ビデオデータ 122
VA APIs 128

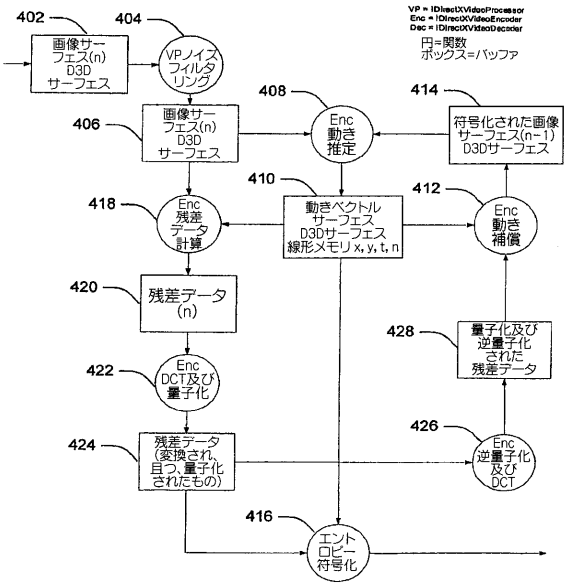
【図２】


ホスト
アクセラレータ
202 生のビデオソース
204 元の映像メモリストア
206 動き推定
208 モード決定、MV選択、及びレート制御
破線パスは、設計に存在する場合もあるし存在しない場合もある
208 予測形成
210 符号化映像メモリストア
212 変換及び量子化
214 逆量子化器及び逆変換
216 エントロピー符号
218 符号化されたビデオの宛先

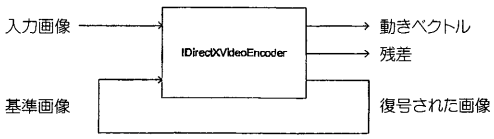
【図３】


300
302 符号化された又は復号されたソースビデオデータを受け取る
304 ソースビデオデータが符号化されている場合、ソースビデオデータを少なくとも部分的に復号する
306 サポートされるビデオ符号化パイプライン構成（複数可）及び他の実施仕様に関して加速ビデオ符号化ハードウェアに問い合わせる
308 ハードウェア加速から利益を得ることができるオペレーションを特定すること、速度／品質決定を行うこと、及び／又はデータフロー移行を最小にすること等を行うために、サポートされるパイプライン構成（複数可）及び実施仕様を評価する
310 ハードウェアが、カスタマイズされたビデオ符号化パイプラインを介して、識別されたソフトウェア符号化オペレーションを加速し、残りの符号化オペレーションをソフトウェアで実施する、符号化オブジェクトを作成する
312 カスタマイズされたビデオ符号化パイプラインに基づいて、復号されたソースビデオデータを符号化する

【図４】

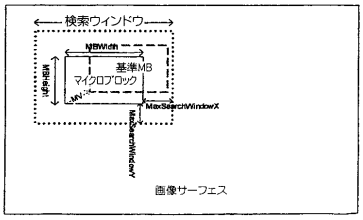

VP = IDirectXVideoProcessor
Enc = IDirectXVideoEncoder
Dec = IDirectXVideoDecoder
円=関数
ボックス=バッファ
402 画像サーフェス(n) D3D サーフェス
404 VPノイズフィルタリング
406 画像サーフェス(n) D3D サーフェス
408 Enc 動き推定
414 符号化された画像サーフェス(n-1) D3Dサーフェス
410 動きベクトル サーフェス D3Dサーフェス 線形メモリ x, y, t, n
412 Enc 動き補償
418 Enc 残差データ 計算
420 残差データ (n)
422 Enc DCT及び量子化
424 残差データ（変換され、且つ、量子化されたもの）
426 Enc 逆量子化及び DCT
428 量子化及び逆量子化された残差データ
416 エントロピー符号化

【図５】


入力画像
基準画像
IDirectXVideoEncoder
動きベクトル
残差
復号された画像

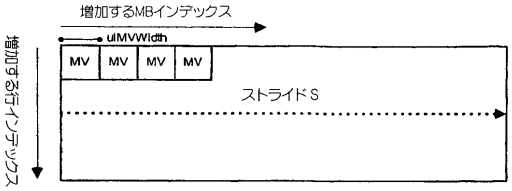
【図６】

【図７】

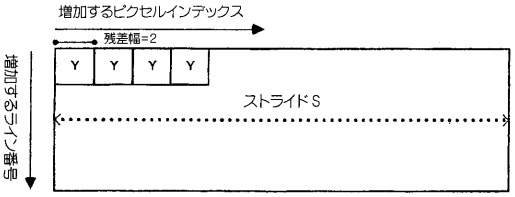


動き設定パラメータ

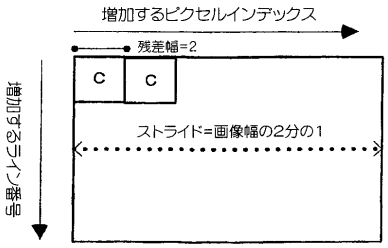
【図８】



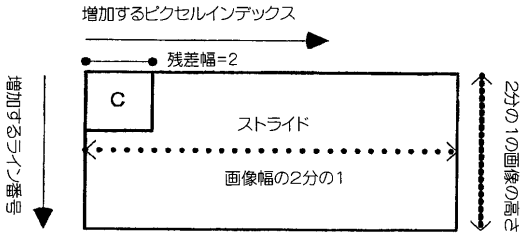
【図９】



【図１０】



【図１１】



フロントページの続き

(74)代理人 100153028
弁理士 上田 忠
(72)発明者 ガネシュ, アナンド
アメリカ合衆国ワシントン州９８０５２, レッドモンド, ワン・マイクロソフト・ウェイ, マイクロソフト コーポレーション, インターナショナル・パテンツ
(72)発明者 マンシル, ドナルド・ジェイ
アメリカ合衆国ワシントン州９８０５２, レッドモンド, ワン・マイクロソフト・ウェイ, マイクロソフト コーポレーション, インターナショナル・パテンツ
(72)発明者 サリバン, ゲーリー・ジェイ
アメリカ合衆国ワシントン州９８０５２, レッドモンド, ワン・マイクロソフト・ウェイ, マイクロソフト コーポレーション, インターナショナル・パテンツ
(72)発明者 エヴァンス, グレン・エフ
アメリカ合衆国ワシントン州９８０５２, レッドモンド, ワン・マイクロソフト・ウェイ, マイクロソフト コーポレーション, インターナショナル・パテンツ
(72)発明者 サドワニ, シャン
アメリカ合衆国ワシントン州９８０５２, レッドモンド, ワン・マイクロソフト・ウェイ, マイクロソフト コーポレーション, インターナショナル・パテンツ
(72)発明者 エストロップ, スティーブン・ジェイ
アメリカ合衆国ワシントン州９８０５２, レッドモンド, ワン・マイクロソフト・ウェイ, マイクロソフト コーポレーション, インターナショナル・パテンツ

審査官 畑中 高行

(56)参考文献 米国特許出願公開第２００２／００６５９５２（ＵＳ, Ａ１）
特表２００３－５１０６６９（ＪＰ, Ａ）
Gary Sullivan and Chad Fogg, Windows Platform Design Notes: Designing Hardware for the Microsoft Windows Family of Operating Systems - Microsoft DirectX VA: Video Acceleration API/DDI, DirectX VA Version 1.01, Microsoft Corporation, ２００１年 1月２３日, [平成24年12月19日検索], インターネット<URL:http://download.microsoft.com/download/1/6/1/161ba512-40e2-4cc9-843a-923143f3456c/DXVA_1.01.doc>
Guobin Shen et al., Accelerate Video Decoding With Generic GPU, IEEE Transaction on Circuits and Systems for Video Technology, IEEE, ２００５年 5月, Vol.15, No.5, p.685-693

(58)調査した分野(Int.Cl., ＤＢ名)
Ｈ０４Ｎ７／２４－７／６８
Ｈ０３Ｍ３／００－１１／００