República Federativa do Brasil
Ministério da Economia
Instituto Nacional da Propriedade Industrial

(11) PI 0708265-7 B1

(22) Data do Depósito: 21/02/2007

(45) Data de Concessão: 03/03/2020

(54) Título: DISPOSITIVO DE COMPUTAÇÃO E MÉTODO IMPLEMENTADO PELO MESMO

(51) Int.Cl.: H04N 19/42; H04N 19/127; H04N 19/156.

(52) CPC: H04N 19/42; H04N 19/127; H04N 19/156.

(30) Prioridade Unionista: 09/02/2007 US 11/673.423; 24/02/2006 US 11/276.336.

(73) Titular(es): MICROSOFT TECHNOLOGY LICENSING, LLC.

(72) Inventor(es): ANAND GANESH; DONALD J. MUNSIL; GARY J. SULLIVAN; GLENN F. EVANS; SHYAM SADHWANI; STEPHEN J. ESTROP.

(86) Pedido PCT: PCT US2007004638 de 21/02/2007

(87) Publicação PCT: WO 2007/100616 de 07/09/2007

(85) Data do Início da Fase Nacional: 25/08/2008

(57) Resumo: CODIFICAÇÃO DE ViDEO ACELERADA. Um serviço de aceleração de codificação de vídeo para aumentar uma ou mais da velocidade e qualidade da codificação de vídeo é descrito. O serviço age como um intermediário entre uma aplicação de programa de computador de codificador de vídeo arbitrária e hardware de aceleração de vídeo arbitrário. O serviço recebe uma ou mais consultas do codificador de vídeo para identificar condições específicas de implementação do hardware de aceleração de vídeo. O serviço faz interface com o hardware de aceleração de vídeo para obter as condições específicas de implementação. O serviço comunica as condições específicas de implementação para o codificador de vídeo. As condições especificas de implementação possibilitam que o codificador de vídeo: (a) determine se uma ou mais da velocidade e qualidade das operações de configuração de codificação de software associadas com o codificador de vídeo podem ser aumentadas com implementação de uma canalização de uma ou mais configurações e capacidades de canalização de codificação suportadas e (b) implemente a canalização pela interface com o serviço.

Relatório Descritivo da Patente de Invenção para "**DISPOSITIVO DE COMPUTAÇÃO E MÉTODO IMPLEMENTADO PELO MESMO**"

PEDIDOS RELACIONADOS

[001] Esse pedido é uma continuação em parte do Pedido de Patente U.S. co-pendente 11/276.336 depositado em 24 de fevereiro de 2006, intitulado "Accelerated Video Encoding", e aqui incorporado por referência.

ANTECEDENTES

[002] As operações de produção e distribuição de conteúdo de multimídia tipicamente incluem codificação de vídeo. Processos de codificação de vídeo têm tipicamente muitos dados e são intensivos no sentido computacional. Como um resultado, os processos de codificação de vídeo podem ser muito consumidores de tempo. Por exemplo, pode levar várias dezenas de horas para um codificador de software codificar um filme de alta definição em alta qualidade. Desde que a qualidade e a velocidade dos processos de codificação de vídeo são fatores significativos para canalizações de produção e distribuição de conteúdo de multimídia bem-sucedidas, sistemas e técnicas para aumentar a velocidade na qual o conteúdo de vídeo em alta qualidade pode ser codificado seriam úteis.

SUMÁRIO

[003] Esse sumário é provido para apresentar uma seleção de conceitos em uma forma simplificada que são também descritos abaixo na descrição detalhada. Esse sumário não é planejado para identificar aspectos chaves ou aspectos essenciais da matéria exposta reivindicada, nem ele é planejado para ser usado como um auxílio na determinação do escopo da matéria exposta reivindicada.

[004] Em vista do acima, um serviço de aceleração de codificação de vídeo para aumentar um ou mais da velocidade e qualidade da

codificação de vídeo é descrito. O serviço age como um intermediário entre uma aplicação de programa de computador de codificador de vídeo arbitrário e hardware de aceleração de vídeo arbitrário. O serviço recebe uma ou mais consultas do codificador de vídeo para identificar condições específicas de implementação do hardware de aceleração de vídeo. O serviço faz interface com o hardware de aceleração de vídeo para obter as condições específicas da implementação. O serviço comunica as condições específicas da implementação para o codificador de vídeo. As condições específicas da implementação possibilitam que o codificador de vídeo: (a) determine se um ou mais de velocidade e qualidade das operações de codificação de software associadas com o codificador de vídeo podem ser aumentadas com a implementação de uma canalização de uma ou mais configurações e capacidades de canalização de codificação suportadas e (b) implemente a canalização pela interface com o serviço.

BREVE DESCRIÇÃO DOS DESENHOS

[005] Nas figuras, o dígito mais a esquerda de um número de referência de componente identifica a figura particular na qual o componente aparece em primeiro lugar.

[006] A figura 1 ilustra um sistema exemplar para codificação de vídeo acelerada, de acordo com uma modalidade.

[007] A figura 2 mostra uma modalidade exemplar de uma configuração de canalização de codificação de vídeo, onde alguns dos processos de codificação são acelerados em hardware.

[008] A figura 3 mostra um procedimento exemplar para codificação de vídeo acelerada, de acordo com uma modalidade.

[009] A figura 4 no apêndice mostra uma aplicação de codificador de vídeo exemplar para ilustrar a maneira na qual as interfaces de programação da aplicação de aceleração de codificação de vídeo podem ser utilizadas, de acordo com uma modalidade.

[0010] A figura 5 no apêndice mostra uma configuração de canalização de codificação de vídeo exemplar, onde o hardware de aceleração acelera a estimativa do movimento, transformação, quantização e o processo inverso para produzir imagens codificadas, de acordo com uma modalidade.

[0011] A figura 6 no apêndice mostra uma configuração de canalização de codificação de vídeo exemplar na qual o hardware acelera somente a estimativa do movimento, de acordo com uma modalidade.

[0012] A figura 7 no apêndice mostra vários parâmetros de estimativa de movimento exemplar, de acordo com uma modalidade.

[0013] A figura 8 no apêndice mostra dados do vetor de movimento exemplar armazenado em uma superfície 3-Dimensional (D3D) de exibição, de acordo com uma modalidade.

[0014] A figura 9 no apêndice mostra um diagrama exemplar indicando que a largura de uma superfície iguala uma imagem YCbCr original, de acordo com uma modalidade.

[0015] A figura 10 no apêndice mostra um diagrama exemplar indicando que o número de valor de resíduo por linha de vídeo é ½ da largura da imagem de vídeo original, de acordo com uma modalidade.

[0016] A figura 11 no apêndice mostra um diagrama exemplar indicando que a largura da superfície de resíduo é ¼ da largura do quadro progressivo original, de acordo com uma modalidade.

DESCRIÇÃO DETALHADA

Visão geral

[0017] Sistemas e métodos para codificação de vídeo acelerada provêem um serviço de aceleração da codificação de vídeo. Esse serviço permite que uma aplicação de codificador de vídeo arbitrária faça interface, em uma maneira independente do dispositivo, com o hardware de aceleração de vídeo arbitrário para definir e implementar uma canalização de codificação de vídeo substancialmente ótima. Pa-

ra realizar isso, o serviço expõe as interfaces do programa de aplicação (APIs) de aceleração de vídeo (VA). Essas APIs encapsulam um modelo do processo de codificação de vídeo. Para definir uma canalização de codificação, a aplicação do codificador de vídeo usa as APIs de VA para consultar condições específicas da implementação (por exemplo, capacidades, etc.) do hardware de aceleração de vídeo (gráficos) disponível. O codificador de vídeo avalia essas condições específicas em vista da arquitetura de codificação de vídeo particular da aplicação (implementada em software) para identificar quaisquer operações de codificação que poderiam se beneficiar (por exemplo, benefícios de velocidade e/ou qualidade) por serem aceleradas em hardware. Tais operações incluem, por exemplo, estimativa do movimento, transformação e operações de quantização e operações inversas tal como compensação de movimento, transformações inversas e quantização inversa. A API também permite que o codificador de vídeo projete uma canalização de codificação que minimiza substancialmente as transições de fluxo de dados através dos barramentos e processadores associados com o dispositivo de computação hospedeiro e o hardware de aceleração, e dessa maneira, aumente mais as velocidades de codificação. A API também permite que o hardware de aceleração influencie a localização dos dados para melhorar o caching local (por exemplo, o hardware de aceleração de vídeo pode funcionar mais eficientemente na memória local para o hardware de vídeo).

[0018] Com base nessas avaliações, o codificador de vídeo projeta uma canalização de codificação de vídeo personalizada que executa algum número de operações de codificação em software e algum número de operações de codificação usando o hardware de aceleração (isto é, pelo menos um subconjunto das operações que poderiam se beneficiar de serem aceleradas pelo hardware). A aplicação do codificador então usa a API para criar a canalização e codificar o conteúdo

de vídeo. Essa canalização personalizada é substancialmente otimizada quando comparada com uma canalização completamente implementada em software porque certas operações de codificação são aceleradas e as transições de dados entre o hospedeiro e o hardware de aceleração são minimizadas. Adicionalmente, o tempo de processamento liberado pela aceleração de certos aspectos do processo de codificação e minimização das transições de dados permite que o(s) processador(es) hospedeiro(s) execute(m) operações de codificação de qualidade superior com ciclos de processamento liberados. A API é também projetada para permitir que os componentes operem em paralelo, de modo que o uso do recurso computacional pode ser maximizado.

[0019] Esses e outros aspectos dos sistemas e métodos para codificação de vídeo acelerada são agora descritos em mais detalhes.

Um Sistema Exemplar

[0020] Embora não requerido, os sistemas e métodos para codificação de vídeo acelerada são descritos no contexto geral de instruções (módulos de programa) executáveis por computador sendo executadas por um dispositivo de computação tal como um computador pessoal e hardware de aceleração de codificação gráfica (vídeo). Módulos de programa geralmente incluem rotinas, programas, objetos, componentes, estruturas de dados, etc., que executam tarefas particulares ou implementam tipos de dados abstratos particulares.

[0021] A figura 1 mostra um sistema exemplar 100 para codificação de vídeo acelerada, de acordo com uma modalidade. O sistema 100 inclui dispositivo de computação hospedeiro 102. O dispositivo de computação hospedeiro 102 representa qualquer tipo de dispositivo de computação tais como um computador pessoal, um laptop, um servidor, dispositivo de computação portátil ou móvel, etc. O dispositivo de computação hospedeiro 102 inclui uma ou mais unidades de proces-

samento 104 acopladas através de um barramento 103 na memória do sistema 106. A memória do sistema 106 inclui módulos ("módulos de programa") de programa do computador 108 e dados do programa 110. Um processador 104 recupera e executa instruções do programa de computador dos módulos respectivos dos módulos de programa 108. Módulos do programa 108 incluem módulos de processamento de vídeo 112 para processar o conteúdo de vídeo, e outros módulos do programa 114 tais como um sistema operacional, acionadores de dispositivo (por exemplo, para fazer interface com o hardware de aceleração de codificação de vídeo, etc.) e/ou assim por diante. Módulos de processamento de vídeo 112 incluem, por exemplo, codificador de vídeo 116, serviço de aceleração de codificação de vídeo 118 e outros módulos de processamento 120, por exemplo, um decodificador de vídeo, filtro(s) de vídeo, um renderizador de vídeo, etc.

[0022] Nessa implementação, o codificador de vídeo 116 é um codificador de vídeo arbitrário. Isso significa que a arquitetura particular, operação, formatos de dados, etc., implementados e/ou utilizados pelo codificador de vídeo 116 são arbitrários. Por exemplo, o codificador de vídeo 116 pode ser distribuído por uma terceira parte, um OEM, etc. Adicionalmente, embora a figura 1 mostre o serviço de aceleração de codificação de vídeo 118 independente da porção do sistema operacional de "outros módulos do programa" 114, em uma implementação, o serviço de aceleração de codificação de vídeo 118 é parte do sistema operacional.

[0023] Os módulos de processamento de vídeo 112 recebem dados de vídeo de entrada compactados ou descompactados 122. Quando os dados de vídeo de entrada 122 estão compactados (já codificados), os módulos de processamento do vídeo 112 decodificam os dados do vídeo de entrada 122 para produzir dados de vídeo de origem decodificados. Tais operações de decodificação são executadas

por um módulo de decodificador. Em uma outra implementação, dados parcialmente decodificados poderiam também ser retidos para auxiliar mais no processo de codificação. Para as finalidades de ilustração exemplar, um tal módulo de decodificador é mostrado como uma porção respectiva dos "outros módulos de processamento de vídeo" 120. Assim, os dados de vídeo de origem decodificados são representados por dados de vídeo de entrada 122 que foram recebidos em um estado decodificado ou representados com resultados da decodificação dos dados de vídeo de entrada 122 que foram recebidos em um estado codificado. Os dados de vídeo de origem decodificados são mostrados como uma porção respectiva de "outros dados do programa" 124.

[0024] Para projetar e implementar uma canalização de codificação de vídeo personalizada que pode ser usada para codificar dados de vídeo de origem decodificados em dados de vídeo codificados 126, o codificador de vídeo 116 faz interface com o serviço de aceleração de codificação de vídeo 118 via as APIs de aceleração de vídeo (VA) 128. Uma implementação exemplar de múltiplas implementações possíveis das APIs de VA 128 é descrita no apêndice. Para definir uma canalização de codificação, a aplicação do codificador de vídeo usa respectivas da API de VA 128 (por exemplo, favor ver o apêndice, §3.4, IVideoEncoderService) para obter condições específicas da implementação de hardware de aceleração disponível 130. Tais condições específicas da implementação incluem, por exemplo:

- um arranjo enumerado identificando configurações de canalização de codificação de vídeo suportadas do hardware de aceleração 130 (por exemplo, obtido via a interface GetCapabilities descrita no apêndice, § 3.4.1),
- uma indicação dos formatos de vídeo suportados (por exemplo, MPEG, WMV, etc.: favor ver o apêndice, GetSupportedFormats, §3.4.2),

•métricas de pesquisa suportadas para operações de estimativa do movimento (ME) (favor ver o apêndice, GetDistanceMetrics, §3.4.3),

•perfis de pesquisa suportados para tempo de processamento versus decisões de permuta de qualidade (favor ver o apêndice, GetSearchProfiles, §3.4.4) e/ou

[0025] capacidades de ME suportadas, por exemplo, informação do tamanho da imagem, tamanho da janela de pesquisa máxima, indicação de suporte de macrobloco variável, etc. (favor, ver o apêndice, GetMECapabilities, §3.4.5).

[0026] Responsivo ao recebimento de tais solicitações do codificador de vídeo 116, o serviço de aceleração de codificação de vídeo 118 consulta o hardware de aceleração de vídeo 130 pelas condições específicas da implementação solicitada e retorna informação associada com as respostas correspondentes do hardware de aceleração 130 para o codificador de vídeo 116. O serviço de aceleração de codificação de vídeo 118 faz interface com o hardware de aceleração de vídeo 130 usando um acionador de dispositivo correspondente. Um tal acionador de dispositivo é mostrado como a porção respectiva de “outros módulos do programa” 114.

[0027] O codificador de vídeo 116 avalia as condições específicas da implementação suportadas pelo hardware de aceleração 130 em vista da arquitetura de codificação de vídeo particular da aplicação (implementada em software) para identificar quaisquer operações de codificação que poderiam se beneficiar (por exemplo, benefícios de velocidade e/ou qualidade) por serem aceleradas em hardware, selecionar um perfil de pesquisa para encapsular uma permuta entre qualidade e velocidade de codificação de vídeo, minimizar transições de dados através de barramentos e entre processadores, etc. Operações exemplares que podem se beneficiar da aceleração do hardware in-

cluem, por exemplo, estimativa de movimento, transformação e quantização. Por exemplo, uma razão para executar a quantização em hardware é minimizar o fluxo de dados entre estágios da canalização.

[0028] A figura 2 mostra uma modalidade exemplar de uma configuração de canalização de codificação de vídeo, onde alguns dos processos de codificação são acelerados em hardware. Por finalidades de ilustração exemplar e descrição, as operações e o fluxo de dados associados com a figura 2 são descritos com relação aos particulares dos componentes da figura 1. Na descrição, o número mais a esquerda de um numeral de referência indica a figura particular onde o componente/trajetória de dados/item referenciado foi apresentado primeiro. Por exemplo, o número mais a esquerda da canalização 200 é, por exemplo, “2”, indicando que ele é primeiro apresentado na figura 2. Nesse exemplo, a canalização de codificação 200 foi configurada/personalizada pelo codificador de vídeo 116 (figura 1) fazendo interface com o serviço de codificação de vídeo 118, tal que respectivas das operações de processamento implementadas no hospedeiro 102 são aceleradas em hardware 130. Por finalidades de ilustração, as operações de processamento ilustradas no lado direito da linha pontilhada em negrito na figura 2 são aceleradas por hardware (por exemplo, hardware de aceleração 130 da figura 1) e as operações de processamento ilustradas no lado esquerdo da figura são executadas pelo dispositivo de computação hospedeiro 102 (figura 1). Na canalização de codificação 200, trajetos de acesso de dados configurados opcionais são mostrados com linhas pontilhadas sem negrito. Os ovais 204 e 212 representam armazenamentos de memória de imagem original e codificada respectivos.

[0029] Nessa implementação exemplar, o codificador de vídeo 116 (figura 1) toma como entrada alguma forma de dados de vídeo compactados ou descompactados 202 (favor também ver os dados do ví-

deo de entrada 122 da figura 1). Favor observar que a configuração de canalização exemplar da figura 2 não copia o vídeo da origem de entrada 202 ("origem de vídeo bruto") para o dispositivo de computação hospedeiro 102 se a origem 202 não está se originando do hospedeiro 102 e se o mecanismo de tomada de decisão do hospedeiro (por exemplo, codificador de vídeo 116) não usa o vídeo de origem. Por exemplo, se decisões de quantização não exigem que o hospedeiro toque nos dados de vídeo, os dados não serão transferidos. Nesse exemplo, a canalização 200 é configurada para converter os dados de entrada 202 para uma outra forma compactada usando as operações respectivas dos blocos 206,208 e 214 até 218.

[0030] Tais operações podem incluir converter dados de vídeo descompactados (YUV) para MPEG-2 compactados, ou elas podem incluir transcodificar os dados de vídeo do formato de dados MPEG-2 para o formato de dados WMV. Por finalidades de ilustração exemplar, assuma que as operações de transcodificação incluem um estágio de descompactação total ou parcial seguido por um estágio de codificação (existem modelos mais eficientes que se desviam da descompactação e trabalham puramente no espaço de transformação (DCT)). Um número de formatos de compactação de vídeo faz uso da estimativa do movimento, transformação e quantização para realizar a compactação. Dos estágios de compactação, a estimativa do movimento é tipicamente a etapa mais lenta, incluindo uma operação de pesquisa massiva onde um codificador (por exemplo, codificador de vídeo 116) tenta encontrar o macrobloco de referência correspondente mais próximo para os macroblocos em uma dada imagem.

[0031] Depois que os vetores de movimento ótimos são determinados (por exemplo, via o bloco 206) para cada um dos macroblocos, o codificador 116 calcula os resíduos diferenciais (por exemplo, via o bloco 208) com base na imagem previamente codificada e no vetor de

movimento ótimo. O vetor de movimento, junto com o resíduo diferencial é uma representação compacta da imagem atual. Os dados do vetor de movimento são também representados diferencialmente. O codificador hospedeiro pode opcionalmente solicitar a reavaliação dos vetores de movimento pelo hardware de aceleração de vídeo para encontrar um macrobloco com um vetor de movimento combinado e/ou residual menor. Os dados do vetor de movimento diferencial resultantes, e os dados residuais são compactados (por exemplo, via o bloco 218), por exemplo, usando técnicas como a codificação run-length (RLE) e a codificação diferencial (por exemplo, codificação Huffman e aritmética) para gerar o fluxo de bits codificado final (dados de vídeo codificados 126) para comunicar para um destino (bloco 218) para apresentação para um usuário. Nesse exemplo, as operações dos blocos 206, 208 e 214 até 218 (por exemplo, operações tais como estimativa de movimento (206), decisão de modo, seleção do vetor de movimento (MV) e controle de taxa (208), formação de predição (210), operações de transformação e quantização (214), inversão e transformação do quantizador e versão (216) e codificação por entropia (218)) são bem conhecidos na técnica e não são assim descritos mais aqui.

[0032] Com referência novamente à figura 1, em uma implementação, o codificador de vídeo 116 é uma aplicação de múltiplos encadeamentos provendo utilização completa do hardware de aceleração 130. Nessa implementação, quando determinando quais operações de codificação de vídeo devem ser aceleradas em hardware, o codificador de vídeo 116 pode estruturar a configuração da canalização particular tal que ambos o processador 104 e o hardware de aceleração 113 são totalmente utilizados. Por exemplo, quando as operações de estimativa de movimento da canalização de codificação de vídeo estão sendo executadas pelo hardware para um quadro particular de dados de vídeo, a canalização pode ser configurada para executar as operações

de codificação por entropia (ou aritmética ou de Huffman) no software pelo hospedeiro em um quadro diferente dos dados de vídeo. Uma canalização de vetor de movimento única exemplar representando a configuração de canalização particular selecionada/estruturada é descrita abaixo no apêndice na seção 5.1.1. Canalizações de vetor de movimento múltiplo exemplar (relativamente complexas) onde o codificador de vídeo 116 solicita vetores de movimento múltiplos do hardware de aceleração 130 e seleciona uma canalização de vetor de movimento com base em vários parâmetros são descritas abaixo no apêndice na seção 5.1.2.

[0033] Com relação à seleção de um perfil de pesquisa, a qualidade dos vetores de movimento se refere a uma taxa de transferência de um fluxo gerado pelo uso dos vetores de movimento. Vetores de movimento de alta qualidade são associados com fluxos de taxa de transferência baixa. A qualidade é determinada pela integralidade da pesquisa do bloco, a qualidade do algoritmo, a distância métrica usada, etc. Vetores de movimento de alta qualidade devem ser usados para executar operações de codificação de vídeo de alta qualidade. Para tratar disso, o serviço de aceleração de codificação de vídeo 118 provê uma construção genérica chamada um perfil de pesquisa para encapsular uma permuta entre qualidade e tempo. O perfil de pesquisa também inclui metadados para identificar o algoritmo de pesquisa usado pelo hardware de aceleração 130, etc. O codificador de vídeo 116 escolhe um perfil de pesquisa particular com base nas exigências particulares da implementação do codificador.

[0034] Com relação à minimização das transições de dados através de barramentos e entre processadores, um processo de codificação implementado por uma configuração de canalização de codificação de vídeo tipicamente incluirá vários estágios de processamento, cada um dos quais pode ou não ser acelerado via o hardware de ace-

leração 130. Nos casos onde o codificador de vídeo 116 determina utilizar a aceleração do hardware em estágios sucessivos da canalização de codificação, pode não ser necessário mover os dados da memória 132 baseada no hardware de aceleração 130 para a memória do sistema 106 associada com o dispositivo de computação hospedeiro 102 e a seguir de volta para a memória baseada no hardware de aceleração 132 para o próximo estágio e assim por diante.

[0035] Mais particularmente, embora ponteiros para vários tipos de dados de vetor de movimento e vídeo possam ser transferidos de um lado para outro entre o dispositivo de computação hospedeiro 102 e o hardware de aceleração 130, em uma implementação, os dados reais são copiados para a memória do sistema 106 somente quando o ponteiro de dados (um ponteiro de superfície D3D9) é explicitamente bloqueado usando, por exemplo, IDirect3DSurface9::LockRect. Interfaces exemplares para bloquear uma superfície são conhecidas (por exemplo, a IDirect3DSurface9::LockRect.interface bem conhecida). Assim, nos casos onde dois estágios de canalização de codificação se seguem, e o dispositivo de computação hospedeiro 102 não precisa executar qualquer processamento intermediário, o dispositivo de computação hospedeiro 102 pode decidir não “bloquear” o armazenamento temporário alocado entre os estágios de processamento. Isso impedirá uma cópia de memória redundante dos dados, e dessa maneira, evitará movimentos/transferências de dados desnecessários. Dessa maneira, o codificador de vídeo 116 projeta uma canalização de codificação de vídeo que substancialmente minimiza as transferências de dados através dos barramentos e entre processadores, e por meio disso, também aumenta as velocidades de codificação de vídeo.

[0036] Nesse ponto, o codificador de vídeo 116 evoluiu as condições específicas da implementação suportadas pelo hardware de aceleração 130 em vista da arquitetura de codificação de vídeo particular

da aplicação (implementada em software) para identificar quaisquer operações de codificação que poderiam se beneficiar de serem aceleradas em hardware, selecionou um perfil de pesquisa, minimizou as transições de dados através dos barramentos e entre processadores e/ou assim por diante. Com base nessas determinações, o codificador de vídeo 116 seleciona uma configuração de canalização particular para codificar os dados de vídeo de origem decodificados, e dessa forma, gera dados de vídeo codificados 126. A seguir, o codificador de vídeo 116 faz interface com o serviço de aceleração de codificação de vídeo 118 para criar um objeto de codificador para implementar a canalização selecionada (favor ver o apêndice, CreateVideoEncoder API, §3.4.6). Nessa implementação, um objeto de codificador (por exemplo, um objeto COM regular) é criado pela identificação da configuração de canalização selecionada e um ou mais dos seguintes: um formato para o fluxo de bits codificado de saída, o número de fluxos de dados de entrada e saída associados com a configuração da canalização, propriedades da configuração estática, um número sugerido de armazenamentos temporários (superfícies) para associação com os fluxos de I/O diferentes com base na configuração de canalização selecionada e um tamanho de fila de alocador especificada pelo acionador com base em recursos que um acionador de dispositivo gráfico é capaz de reunir, e outros parâmetros. (O tamanho da fila e o número de armazenamentos temporários de dados estão essencialmente se referindo a mesma coisa: um é “sugerido”, o outro é “real”).

[0037] A seguir, o codificador de vídeo 116 usa o objeto do codificador criado para fazer interface com o serviço de aceleração de codificação de vídeo 118 para codificar os dados de vídeo de origem decodificados. Para esse fim, o objeto do codificador submete solicitações de execução para o hardware de aceleração 130 (favor ver o apêndice, IVideoEncode:Execute API, §3.2.1).

[0038] Em vista do acima, o sistema 100 permite implementações arbitrárias de aplicações de codificador de vídeo 116 para definir e criar configurações de canalização de codificação de vídeo durante o tempo de execução para tirar vantagem completa do hardware de aceleração de codificação de vídeo disponível para aumentar a velocidade e a qualidade da codificação. Como parte dessas operações de configuração do tempo de execução, o codificador de vídeo 116 pode usar APIs de VA 128 para especificar que a canalização de codificação é para implementar a pesquisa direcionada iterativa (múltiplas passagens de pesquisa de refinamento crescente), definir e usar estratégias de pesquisa genericamente selecionáveis (por exemplo, selecionar um algoritmo de pesquisa com base nas métricas de qualidade independente de qualquer conhecimento dos detalhes sobre o algoritmo real utilizado), utilizar metodologias independentes do formato (por exemplo, onde um codificador de vídeo 116 não está consciente do formato de imagem particular dos dados do vídeo de entrada 122 e o hardware de aceleração 130 não está ciente do formato de saída compactado para os dados de vídeo codificados 126) para controlar a pesquisa, adaptar tamanhos de dados (por exemplo, onde o codificador de vídeo 116 seleciona um tamanho de macrobloco com base em um algoritmo de pesquisa) e assim por diante.

Um Procedimento Exemplar

[0039] A figura 3 mostra um procedimento exemplar 300 para codificação de vídeo acelerada, de acordo com uma modalidade. Por finalidades de descrição exemplar, as operações do procedimento são descritas com relação aos componentes do sistema 100 da figura 1. O numeral mais a esquerda de um número de referência do componente indica a figura particular onde o componente é descrito em primeiro lugar.

[0040] No bloco 302, o codificador de vídeo 116 (figura 1) recebe

dados de vídeo de entrada 122. Se os dados de vídeo de entrada 122 não estão compactados, os dados de vídeo de entrada representam dados de vídeo de origem decodificados. No bloco 304, se os dados de vídeo de entrada 122 estão compactados, o codificador de vídeo 116 descompacta os dados de vídeo de entrada para gerar dados de vídeo de origem decodificados. No bloco 306, o codificador de vídeo 116 faz interface com a API de VA 128 para consultar o hardware de aceleração 130 com relação às capacidades e condições específicas de implementação da configuração da canalização de codificação de vídeo. No bloco 308, o codificador de vídeo 116 avalia as capacidade suportadas e as condições específicas de implementação dentro do contexto da implementação do codificador de vídeo 116, para identificar as operações de codificação de vídeo associadas com a implementação particular do codificador de vídeo 116 que podem se beneficiar da aceleração de hardware, tomar decisões de qualidade e/ou velocidade de decodificação, minimizar transições de dados através dos barramentos e entre processadores e/ou assim por diante.

[0041] No bloco 310, o codificador de vídeo 116 cria um objeto de codificação que implementa uma canalização de codificação configurada para executar as operações de codificação de vídeo identificadas que podem se beneficiar da aceleração de hardware no hardware de aceleração 130, implementar as permutas de velocidade/qualidade (por exemplo, via um perfil de pesquisa selecionado) e minimizar as transições de fluxo de dados. No bloco 312, o codificador de vídeo usa o objeto do codificador criado para codificar os dados de vídeo de origem decodificados de acordo com a seqüência das operações e arquitetura de codificação delineada pela canalização de codificação de vídeo personalizada no bloco 310. Essas operações de codificação do bloco 312 geram dados de vídeo codificados 126 (figura 1).

Conclusão

[0042] Embora os sistemas e métodos para codificação de vídeo acelerada tenham sido descritos em linguagem específica para aspectos estruturais e/ou operações ou ações metodológicas, é entendido que as implementações definidas nas reivindicações anexas não são necessariamente limitadas aos aspectos ou ações específicas descritos.

[0043] Por exemplo, embora as API's 128 da figura 1 tenham sido descritas dentro do contexto da codificação de dados de vídeo, as APIs 128 podem ser usadas fora do contexto de codificação para aceleração de hardware de outras funções tais como detecção de borda, redução de ruído baseada no vetor de movimento, estabilização da imagem, nitidez, conversão da taxa de quadros, computação da velocidade para aplicações de visão de computador, etc. Por exemplo com referência à redução do ruído, em uma implementação, o codificador de vídeo 116 (figura 1) computa vetores de movimento para todos os macroblocos dos dados da imagem de origem decodificados. A seguir, o codificador de vídeo 116 utiliza a magnitude do movimento, direção e correlação com vetores de movimento de macroblocos circundantes para determinar se existe um movimento de objeto local na imagem de entrada. Nessa implementação, o codificador de vídeo 116 então utiliza a magnitude do vetor para direcionar o acompanhamento do objeto / agressividade da filtragem ou mudanças médias de um objeto particular para reduzir o ruído estaticamente aleatório.

[0044] Em um outro exemplo com relação à estabilização da imagem, em uma implementação, o codificador de vídeo 116 calcula vetores de movimento para todos os macroblocos e dados de origem decodificados. O codificador de vídeo 116 então determina se existe movimento global na imagem. Isso é realizado correlacionando todos os valores do vetor de movimento e determinando se os valores correlacionados são similares. Se afirmativo, então o codificador de vídeo 116

conclui que existe movimento global. Alternativamente, o codificador de vídeo 116 utiliza um tamanho de macrobloco grande e determina se existe movimento geral do macrobloco grande. Depois de determinar se o movimento global está presente, se o codificador de vídeo 116 também verifica que o vetor de movimento global tende a ser convulsivo através dos quadros, o codificador de vídeo 116 conclui que existe contração de câmera e compensa isso antes de começar a filtragem do ruído e as operações de codificação.

[0045] Dessa maneira, os aspectos específicos e as operações do sistema 100 são revelados como formas exemplares de implementação da matéria exposta reivindicada.

Apêndice A

[0046] "API de Aceleração de Codificação de Vídeo Exemplar"

Codificação de Vídeo

[0047] Esse apêndice descreve aspectos de uma implementação exemplar das APIs de aceleração de codificação de vídeo 128 (figura 1) para a codificação de vídeo acelerada, também citada como codificação VA. Nessa implementação, APIs 128 são projetadas para possibilitar que as aplicações de processamento de vídeo e codificação (por exemplo, um módulo do codificador de vídeo 116) alavanquem o suporte do hardware de aceleração 130 (por exemplo, um GPU) para aceleração estimativa de movimento, computação de resíduo, compensação e transformação do movimento.

1 Tabela de Conteúdos

[0048] Codificação de vídeo

4.2 Funcionalidade da codificação

4.2.1 VADDI Encode_Function_Execute_Input

4.2.2 VADDI Encode_Function_Execute_Output

4.3 Estruturas do dispositivo de extensão

4.3.1 VADDI_PRIVATEBUFFER

4.3.2 D3DDDIARG_EXTENSIONEXECUTE

4.3.3 FND3DDDI_DESTROYEXTENSIONDEVICE

4.3.4 FND3DDDI_EXTENSIONEXECUTE

4.3.5 D3DDDI_DEVICEFUNCS

4.4 Estruturas e funções D3D9

5 Modelo de programação exemplar

5.1 Eficiência da canalização

5.1.1 Exemplo: vetor de movimento único (canalização completa)

5.1.2 Exemplo: vetores de movimento múltiplos

2 Projeto exemplar

2.1 Leiaute do codificador

[0049] A figura 5 mostra uma aplicação do codificador de vídeo exemplar para ilustrar a maneira na qual as APIs de aceleração de codificação de vídeo podem ser utilizadas, de acordo com uma modalidade. Nesse exemplo, o codificador de vídeo 116 é implementado na forma de um DMO ou MFT. A figura 5 mostra os dados de entrada (correspondendo com uma “recepção”) e os dados de saída depois de vários estágios do processamento. As caixas representam dados enquanto os círculos representam funções da API invocadas pela aplicação do codificador. Os círculos assim representam o núcleo da API como visto pela aplicação do codificador.

2.2 Configurações da canalização ou modo

[0050] O hardware de aceleração é visto como uma canalização, e o GUID da canalização é usado para descrever os elementos de con-

figuração mais básicos da canalização. A meta da aceleração da codificação pode ser imaginada como intimamente ligada à meta da eficiência da canalização.

[0051] O projeto permite canalizações divididas (ou de múltiplos estágios) onde os dados passam de um lado para o outro entre o PC hospedeiro e o hardware, antes da saída final ser obtida. Nenhuma configuração de canalização de múltiplos estágios foi projetada ainda, as configurações abaixo descrevem canalizações de estágio único não divididas.

2.2.1 VA2_EncodePipe_FuLL

// (BFC87EA2-63B6-4378-A619-5B451EDC8940)

cpp_quote{"DEFINE_GUID(VA2_EncodePipe_Full, Ox-bfc87ea2, Ox63b6, 0x4378, Oxa6, 0x19, OxSb, 0x45, Oxie, Oxdc, Oxb9, 0x40);" }

[0052] A figura 6 mostra uma configuração de canalização de codificação de vídeo exemplar, onde o hardware de aceleração acelera a estimativa do movimento, transformação, quantização e o processo inverso para produzir imagens decodificadas, de acordo com uma modalidade. O hardware produz como saída vetores de movimento, resíduos (brilho e saturação) e uma imagem decodificada. A imagem decodificada não precisa ser transferida para a memória do sistema já que a sua única finalidade é calcular os vetores de movimento para o próximo quadro.

[0053] A documentação posterior se referirá a um parâmetro chamado NumStreams. Para essa configuração de canalização, o NumStreams é cinco. Os StreamIds reais são mostrados no diagrama em parênteses.

[0054] Essa é uma canalização de estágio única, não dividida e, portanto o parâmetro de estágio da execução não se aplica.

Descrições de fluxo

[0055] Observe que StreamType_* é uma abreviação para VA2_Encode_StreamType_*.

Imagem de entrada

[0056] O ID do fluxo é um, e o tipo de fluxo é StreamType_Video. Esse fluxo representa a imagem para a qual os dados de movimento são buscados. O alocador para esse fluxo é negociável – ou a interface atual pode fornecer um ou um alocador externo pode ser usado. A escolha do alocador é feita no momento da criação, e se um alocador externo é escolhido, um ID do fluxo de um será considerado um valor de entrada ilegal para GetBuffer.

Imagens de referência

[0057] O ID do fluxo é dois, e o tipo do fluxo é StreamType. Esse fluxo representa uma lista de imagens de referência usadas para calcular vetores de movimento. A interface atual não supre um alocador separado para esse fluxo. As superfícies de entrada são recicladas a partir do fluxo de imagem decodificada (ID = 5) ou obtidas de outro lugar.

Vetores de movimento

[0058] O ID do fluxo é três, e o tipo do fluxo é StreamType_MV. Esse fluxo representa um parâmetro de saída contendo dados do vetor de movimento. Armazenamentos temporários para esse fluxo são obtidos através de GetBuffer somente.

Resíduos

[0059] O ID do fluxo é quatro, e o tipo do fluxo é StreamType_Residues. Esse fluxo representa um parâmetro de saída contendo valores de resíduo para todos os três planos. Armazenamentos temporários para esse fluxo são obtidos através de GetBuffer somente.

Imagem decodificada

[0060] O ID do fluxo é cinco, e o tipo do fluxo é Stream-

Type_Video. Esse fluxo representa um parâmetro de saída contendo a imagem decodificada obtida dos resíduos quantizados e valores do vetor de movimento. Armazenamentos temporários para esse fluxo são obtidos através de GetBuffer somente.

2.2.2 VA2_EncodePipe_MotionEstimation

// {F18B3D19-CA3E-4a6b-AC10-53F86D509E04}

cpp_quote("DEFINE_GUID {VA2_EncodePipe_MotionEstimation, Oxfl8b3d19 Oxca3e, Ox4a6b, Oxac, Ox10, Ox53, Oxf8, Ox6d, 0x50, Ox9e, 0x4);" }

[0061] A figura 7 mostra uma configuração de canalização de codificação de vídeo exemplar na qual o hardware acelera somente a estimativa do movimento, de acordo com uma modalidade. Essa configuração de canalização adota um conjunto de imagens de referência como entrada, e descarrega vetores de movimento como saída. A imagem decodificada nesse caso tem que ser gerada e suprida pelo software do hospedeiro.

[0062] NumStreams para essa configuração de canalização é três. Os StreamIds para os vários fluxos são mostrados no diagrama em parênteses.

[0063] Essa é uma canalização de estágio único não dividida e o parâmetro de estágio de execução não se aplica.

3 API Exemplar

3.1 Definição de interface

3.1.1 IVideoEncoder

```
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 NumOutputDataParametere,
[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
};
}

3.1.2 IVideoEncoderService

interface IVideoEncoderService : IVideoAccelerationService
{
HRESULT GetPipelineConfigurations(
[out] UINT32* pCount,
[out, unique, size_is(*pCount)] QUID** 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_Search Profile**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

);

};

3.2 Métodos: IvideoEncoder

3.2.1 GetBuffer

[0064] Essa função retorna armazenamentos temporários (superfícies de codificação) para uso na chamada de execução. Os armaze-

namentos temporários são liberados prontamente depois do uso chamando ReleaseBuffer, para evitar o bloqueio da canalização.

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)

Parâmetros

StreamId

[0065] Refere-se ao fluxo particular para o qual os armazenamentos temporários são desejados. Dependendo do fluxo particular, tipos diferentes de armazenamentos temporários como armazenamentos temporários da imagem de entrada, armazenamentos temporários do vetor de movimento, etc. serão retornados. Nem todos os IDs do fluxo para uma dada configuração são entradas válidas para essa função. Valores permitidos para StreamId são especificados como parte da configuração da canalização.

StreamType

[0066] Especifica o tipo de armazenamento temporário a ser retornado. Tipicamente, o tipo de fluxo será indicado por StreamId, e negociado no momento da criação. Se StreamType não é consistente com StreamId, a função retorna um valor de erro. O armazenamento temporário dos dados é interpretado (estereotipado) com base no valor de StreamType como descrito pela tabela na seção de Observações.

Blocking

[0067] Especifica o comportamento da função quando existe inanição ou alguma outra necessidade para o estrangulamento. Um valor de verdadeiro indica que a função deve bloquear, enquanto falso indica que a função deve retornar E_NOTAVAILABLE.

pBuffer

[0068] O ponteiro para um armazenamento temporário de dados a ser liberado através de ReleaseBuffer. O ponteiro é remodelado (interpretado) com base no parâmetro StreamType, e isso é descrito na tabela na seção de Observações abaixo.

Valores de retorno

S_OK

[0069] Função bem-sucedida.

E_NOTAVAILABLE

[0070] Esse é retornado quando o acionador precisa de armazenamentos temporários, e o indicador de bloqueio foi ajustado para falso.

E_INVALIDPARAMETER

[0071] Os parâmetros de entrada estavam incorretos. Isso pode ser usado, por exemplo, quando StreamType não iguala o valor esperado para o dado StreamId.

E _UNSUPPORTED _ STREAM

[0072] StreamId é inválido. GetBuffer não supre armazenamentos temporários para o ID do fluxo especificado. Valores permitidos para StreamId são descritos como parte da configuração da canalização. Para o ID do fluxo especificado, o alocador pode ser externo ou pode não existir alocador absolutamente.

E_FAIL

[0073] Função falha.

Observações

[0074] Normalmente essa função retorna o controle muito rápido

quando os armazenamentos temporários já estão presentes na fila do alocador. As únicas condições sob as quais essa função deve bloquear (ou retornar E_NOTAVAILABLE) são quando todos os armazenamentos temporários da fila do alocador foram submetidos ao dispositivo, ou foram consumidos pela aplicação e, portanto não liberados.

[0075] Tipos de fluxo e formatos do armazenamento temporário

StreamType_Video	IDirect3DSurface9** pBuffer.
StreamType_MV	IDirect3DSurface9** pBuffer
StreamType_Residues	IDirect3DSurface9* pBuffer[3]., isto é, pBuffer é um ponteiro para três ponteiros de superfície D3D. pBuffer[0] é um ponteiro para a superfície de brilho. pBuffer[1] é um ponteiro para a superfície Cb, ou NULL se não aplicável. pBuffer[2] é um ponteiro para a superfície Cr, ou NULL se não aplicável.

3.2.2 ReleaseBuffer

[0076] Essa função é usada para liberar uma superfície de volta para a fila do alocador para reutilização via GetBuffer.

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

Parâmetros

StreamId

[0077] ID do fluxo associado com o armazenamento temporário.

StreamType

[0078] Tipo do fluxo para o armazenamento temporário.

pBuffer

[0079] Armazenamento temporário a ser liberado de volta para a fila do alocador.

Valores de retorno

S_OK

[0080] Função bem sucedida.

E_FAIL

Função falha.

3.2.3 Execute

[0081] Essa função é usada para submeter solicitações para o hardware. Ela contém armazenamentos temporários de dados de entrada e saída obtidos através de GetBuffer, bem como alguma informação de configuração. A função é assíncrona e sua conclusão é indicada pelo evento sendo sinalizado. O estado de conclusão é indicado usando o parâmetro pStatus que é alocado no monte e verificado somente depois que o evento foi sinalizado.

[0082] Os armazenamentos temporários supridos como parâmetros para essa função não são lidos de (por exemplo, através de LockRect), ou escritos pela aplicação até que a função tenha verdadeiramente concluído. A conclusão verdadeira é implicada por um valor de erro sendo retornado pela função ou se essa função retorna sucesso, então pela sinalização de hEvent (parâmetro para essa função). Quando o mesmo armazenamento temporário é inserido em várias instâncias da chamada Execute, ele não será acessado até que todas as chamadas Execute associadas tenham concluído. O ponteiro para uma superfície em uso por Execute pode ainda ser suprido como um parâmetro para as funções VA como Execute desde que isso não re-

quer que os dados fiquem bloqueados. Essa última regra explica como a mesma imagem de entrada pode ser usada em múltiplas chamadas Execute ao mesmo tempo.

[0083] Os armazenamentos temporários supridos para essa chamada obedecem à semântica do alocador negociada no tempo da criação. Se um alocador externo é usado quando GetBuffer é esperado de ser usado, essa função retornará E_FAIL.

HRESULT Execute(

[in] UINTB 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

};

Parâmetros

Stage

[0084] Para configurações de canalização dividida, esse parâmetro identifica o estágio específico da canalização dividida. A numeração é baseada em um e para canalizações não divididas esse parâmetro é ignorado.

NumInputDataParameters

[0085] Tamanho da formação dos dados de entrada (próximo parâmetro).

pInputData

[0086] Formação de ponteiros para valores dos dados de entrada. Ponteiros de dados individuais são remodelados apropriadamente com base no valor StreamId que tem um StreamDescription associado especificado na criação. Os armazenamentos temporários de dados são alocados na criação e obtidos durante o processo de transferência contínua chamando GetBuffer.

NumOutputDataParameters

[0087] Tamanho da formação dos dados de saída (próximo parâmetro).

pOutputData

[0088] Formação de ponteiros para valores dos dados de saída. Os ponteiros de dados individuais são remodelados apropriadamente com base no valor StreamId que tem um StreamDescription associado especificado na criação. Os armazenamentos temporários de dados são alocados na criação e obtidos durante o processo de transferência contínua chamando GetBuffer.

NumConfigurationParameters

[0089] Tamanho da formação de configuração (próximo parâmetro).

pConfiguration

[0090] Formação de parâmetros de configuração controlando a execução da canalização. A configuração geral é a união dessa estrutura junto com parâmetros de configuração estáticos supridos quando o codificador foi criado.

hEvent

[0091] Alça de evento sinalizando que os dados de saída estão prontos.

pStatus

[0092] Estado indicando se a operação solicitada completou com

sucesso. Valores permitidos incluem S_OK (conclusão bem-sucedida), E_TIMEOUT (se o limite de tempo foi excedido) e E_SCENECHANGE (se a detecção de mudança de cena foi habilitada e detectada). Em ambos os casos de erro, nenhuma das superfícies de saída contém quaisquer dados úteis. Esse parâmetro é alocado no monte, e o valor de retorno é verificado somente depois que hEvent foi sinalizado.

Valores de retorno

S_OK

[0093] Função bem-sucedida.

E_FAIL

Função falha.

Observações

[0094] Se a alça do evento foi sinalizada, isso significa que LockRect deve completar instantaneamente quando chamado em qualquer uma das superfícies de saída desde que elas estão prontas. Em particular, espera-se que a chamada LockRect não bloqueie qualquer duração de tempo aguardando quaisquer alças de evento. Nem ela pode desperdiçar tempo de CPU através de voltas ocupadas.

3.3 Estruturas de dados: Execute

[0095] A chamada Execute tem parâmetros de dados e parâmetros de configuração. Parâmetros de dados específicos podem ser imaginados como derivando da classe de base (ou estrutura) VA2_Encode_ExecuteDataParameter e parâmetros de configuração específicos podem ser imaginados como derivando da classe de base (ou estrutura) VA2_Encode_ExecuteConfi-gurationParameter.

3.3.1 VA2_Encode_ExecuteDataParameter

```
typedef struct _VA2_Encode_ExecuteDataParameter {
UINT32Length;
UINT32StreamId;
} VA2_Encode_ExecuteDataParameter;
```

Elementos

Length

[0096] Número de bytes nessa estrutura. Provido por extensibilidade.

StreamId

[0097] O ID do fluxo de dados como definido na configuração da canalização. Os formatos do armazenamento temporário são negociados no tempo de criação usando o parâmetro StreamDescription.

3.3.2 VA2_Encode_ExecuteConfigurationParameter

typedef struct _VA2_Encode__ExecuteConfiguration-Parameter {

UINT32Length;

UINT32Streamid;

UINT32ConfigurationType;

} VA2 Encode_ ExecuteConfigurationParameter;

#define VA2_Encode_ConfigurationType_ MotionEstimation 0x1

#define VA2_Encode_ConfigurationType_ Quantization 0x2

Elementos

Length

[0098] Número de bytes nessa estrutura. Provido por extensibilidade.

StreamId

[0099] O ID do fluxo de dados como definido na configuração da canalização. Esse pode ser usado para deduzir se os dados são de entrada ou de saída.

ConfigurationType

[00100] Esse parâmetro descreve o parâmetro de configuração e é usado para estereotipar a estrutura atual apropriadamente.

Observações

[00101] Essa estrutura age como um tipo de base para informação de configuração mais especializada. O tipo de base é estereotipado para um tipo mais especializado com base no parâmetro ConfigurationType. O mapeamento entre ConfigurationType e as estruturas especializadas é descrito na tabela abaixo.

Tipos de configuração

ConfigurationTypeMotion_Estimation	ConfigurationTypeMotion_Estimation
ConfigurationType_Quantization	ConfigurationParameter_Quantization

3.3.3 DataParameter_MotionVectors

typedef struct_VA2_Encode_ExecuteDataParameter_MotionVectors {

UINT32Length;

UINT32StreamId; VA2_Encode_MVSurfaCe* pMVSurface;

} VA2_Encode_ExecuteDataParameter_MotionVectors;

Elementos

Length

[00102] Número de bytes nessa estrutura. Provido por extensibilidade.

StreamId

[00103] O ID do fluxo de dados como definido na configuração da canalização. Isso pode ser usado para deduzir se os dados são de entrada ou de saída.

pMVSurface

[00104] Ponteiro para uma estrutura contendo a superfície D3D do vetor de movimento.

3.3.4 DataParameter Residues

typedef struct _VA2_Encode_ExecuteDataParameter_ Residues {

UINT32Length;

UINT32StreamId;

VA2_Encode_ResidueSurface*pResidueSurfaceY;

VA2_Encode_ResidueSurface*pResidueSurfaceCb;

VA2_Encode_ResidueSurface*pResiduesurfaceCr;

} VA2_Encode_ExecuteDataParameter_Residues;

Elementos

Length

[00105] Número de bytes nessa estrutura. Provido por extensibilidade.

StreamId

[00106] O ID do fluxo de dados como definido na configuração da canalização. Isso pode ser usado para deduzir se os dados são de entrada ou de saída.

pResidueSurfaceY

[00107] Superfície do resíduo contendo valores de brilho.

pResidueSurfaceCb

[00108] Superfície do resíduo contendo valores Cb de saturação.

pResidueSurfaceCr

[00109] Superfície do resíduo contendo valores Cr de saturação.

3.3.5 DataParameter_inputimage

typedef struct _VA2_Encode_ExecuteDataParameter_ InputImage {

UINT32Length;

UINT32StreamId;

VA2_Encode_ImageInfo* pImageData;

} VA2_Encode_ExecuteDataParameter_InputImage;

Elementos

Length

[00110] Número de bytes nessa estrutura. Provido por extensibilidade.

StreamId

[00111] O ID do fluxo de dados como definido na configuração da canalização. Isso pode ser usado para deduzir se os dados são de entrada ou de saída.

pImageData

[00112] Ponteiro para uma estrutura contendo a superfície D3D de imagem de entrada. Essa é a superfície para a qual os vetores de movimento são buscados.

3.3.6 DataParameter_ReferenceImages

typedef struct _VA2_Encode_ExecuteDataParameter_ReferenceImages {

UINT32Length;

UINT32StreamId;

UINT32NumReferenceImages; VA2_Encode_ImageInfo* pReferenceImages

} VA2_Encode_ExecuteDataParameter_ReferenceImages;

Elementos

Length

[00113] Número de bytes nessa estrutura. Provido por extensibilidade.

StreamId

[00114] O ID do fluxo de dados como definido na configuração da canalização. Isso pode ser usado para deduzir se os dados são de entrada ou de saída.

DataType

NumReferenceImages

[00115] Tamanho da formação das imagens de referência (próximo

parâmetro).

pReferenceImages

[00116] Formação das imagens de referência na qual basear os vetores de movimento. Para formatos simples como MPEG-2 somente um quadro progressivo (ou dois campos) podem ser usados. Por outro lado, formatos como H.264 e VC-1 suportam vetores de movimento atravessando vários quadros. Um P-quadro no MPEG-2 usa somente uma imagem de referência enquanto um B-quadro com vídeo entrelaçado, e movimento do tipo de campo poderiam usar 4 imagens, cada uma das quais pode se referir a um quadro ou um campo.

3.3.7 DataParameter DecodedImage

typedef struct _VA2_Encode_ExecuteDataParameter_ DecodedImage {

UINT32Length;

UINT32StreamId;

VA2_Encode_ImageInfo* pYCbCrImage;

} VA2_Encode_ExecuteDataParameter_DecodedImage;

Elementos

Length

[00117] Número de bytes nessa estrutura. Provido por extensibilidade.

StreamId

[00118] O ID do fluxo de dados como definido na configuração da canalização. Isso pode ser usado para deduzir se os dados são de entrada ou de saída.

DataType

pYCBCrImage

[00119] Imagem decodificada de saída obtida depois da quantização inversa, transformação inversa e compensação de movimento. Para boa canalização, a superfície D3D associada não deve ser blo-

queada ou os dados transferidos para a memória do sistema desnecessariamente. Os ponteiros de superfície podem ainda ser usados como uma imagem de referência.

3.3.8 VA2_Encode_ImageInfo

typedef struct VA2_Encode_ImageInfo{IDirect3 DSurface9* pSurface;

BOOLField;

BOOLInterlaced;

RECTWindow;

} VA2_Encode_ImageInfo;

Elementos

pSurface

[00120] Ponteiro para uma superfície D3D contendo a imagem no formato YCbCr.

Field

[00121] Um valor de um indica que a superfície contém um campo de dados de vídeo, e os dados são assumidos como sendo entrelaçados. Zero indica um quadro progressivo total.

Interlaced

[00122] Um valor de um indica que os dados da imagem são entrelaçados. Esse indicador deve ser usado somente quando Field (parâmetro acima) é ajustado para um. Se Field está ajustado para um, é assumido que os dados estão entrelaçados.

Windows

[00123] Um retângulo dentro da imagem. Isso poderia ser usado para restringir a chamada de estimativa do movimento para retornar vetores de movimento para apenas um retângulo dentro de toda a imagem.

3.3.9 ConfigurationParameter_MotionEstimation

typedef struct _VA2_Encode_ExecuteConfiguration Pa-

rameter_MotionEstimation {

UINT32Length;

UINT32StreamId;

UINT32ConfigurationType;

VA2_Encode_MEParameters* pMEParams;

} VA2_Encode_ExecuteConfigurationParameter_Motion Estimation;

Elementos

Length

[00124] Número de bytes nessa estrutura. Provido por extensibilidade.

StreamId

O ID do fluxo de dados como definido na configuração da canalização. Isso pode ser usado para deduzir se os dados são de entrada ou de saída.

ConfigurationType

pMEParams

[00125] Ponteiro para uma estrutura definindo vários parâmetros governando a pesquisa de movimento incluindo a janela de pesquisa, etc.

Observações

[00126] A figura 8 mostra vários parâmetros exemplares de estimativa do movimento, de acordo com uma modalidade. Esses parâmetros são para uso nas estruturas abaixo.

3.3.10 VA2_Encode_SearchResolution

typedef enum {

VA2_Encode_SearchResolution_FullPixel,

VA2_Encode_SearchResolution_HalfPixel,

VA2_Encode_SearchResolution_QuarterPixel

} VA2_Encode_SearchResolution;

Descrição

FullPixel

[00127] Vetores de movimento são calculados em unidades de pixel completas.

HalfPixel

[00128] Vetores de movimento são calculados em unidades de meio pixel. Assim um valor do vetor de movimento de (5,5) se refere a um macrobloco de dados que está (2,5, 2,5) pixels distante.

QuarterPixel

[00129] Vetores de movimento são calculados em unidades de quarto de pixel. Assim, um valor do vetor de movimento de (10, 10) se refere a um macrobloco de dados que está (2,5, 2,5) pixels distante.

[00130] Na computação dos valores do vetor de movimento de sub-pixel, o codificador estima os valores de brilho e saturação usando interpolação. O esquema de interpolação específico é dependente do formato e os seguintes GUIDs (parte da configuração estática) controlam o esquema de interpolação.

// {E9AF78CB-7A8A-4d62-887F-86A418364C79}

cpp_quote

("DEFINE_GUID(VA2_Encode_Interpolation_MPEG2Bilinear,

Oxe9af78cb, Ox7a8a, 0x4d62, Ox88, Ox7f, Oxb6, Oxa4, 0x18, 0x36, Ox4c, 0x79);")

// {A94BBFCB-1BF1-475c-92DE-67298AF56BB0}

cpp_quote("DEFINE_GUID(VA2_Encode_interpolation_MPEG 213icubic, Oxa94bbfcb, Oxlbf1, Ox475c, 0x92, Oxde, 0x67, 0x29, Ox8a, OxfS, Ox6b, OxbO};")

3.3.11 VA2_Encode_SearchProfile

typedef struct_VA2_Encode_SearchProfile

{

UINT8 QualityLevel;

UINT8 TimeTaken;

GUID SearchTechnique;

GUID SubpixelInterpolation;

} VA2_Encode_SearchProfile

Elementos

QualityLevel

[00131] Um número na faixa [0-100] que indica a qualidade relativa dos vetores de movimento entre os perfis diferentes suportados pelo dispositivo.

TimeTaken

[00132] Um número na faixa [0-100] que indica o tempo relativo consumido para perfis de pesquisa diferentes. Isso possibilita que a aplicação faça uma troca de qualidade-tempo razoável.

SearchTechnique

[00133] Um GUID indicando o algoritmo de pesquisa usado.

SubpixelInterpolation

[00134] Um GUID indicando o esquema de interpolação de subpixel usado.

Observações

[00135] Não existe definição universalmente aceita de qualidade absoluta, então nós estamos estabelecendo uma medida relativa. Os valores indicados contra TimeTaken devem seguir uma regra de proporção restrita. Se perfil 1 consome 10ms e perfil 2 consome 20ms, os valores de TimeTaken devem estar na relação de 20/10 = 2.

3.3.12 VA2_Encode_MBDescription

typedef struct_VA2_Encode_MBDescription {

BOOL ConstantMSSize;

UINT32 MBWidth;

UINT32 MBHeight;

UINT32 MBCOUnt;

RECT* pMBRectangles;

} VA2_Encode_MBDescription;

Elementos

ConstantMBSize

[00136] Um valor de um indica que todos os macroblocos na imagem atual têm o mesmo tamanho. Isso pode não ser verdadeiro para formatos como H.264.

MBWidth

[00137] Largura de um macrobloco. Válido somente se bConstantMBSize é um.

MBHeight

[00138] Altura de um macrobloco. Válido somente se bConstantMBSize é um.

MBCount

[00139] Se bConstantMBSize é zero, então os macroblocos (ou segmentos) na imagem são descritos usando uma formação de retângulos. Esse parâmetro descreve o tamanho nos elementos do parâmetro pMBRectangles seguinte.

pMBRectangles

[00140] Uma formação de retângulos descrevendo como a imagem é para ser cortada.

3.3.13 VA2_Encode_SearchBounds

typedef struct_VA2_Encode_SearchBounds {

BOOL DetectSceneChange;

UINT32 MaxDistanceInMetric;

UINT32 TimeLimit;

UINT32 MaxSearchWindowX;

UINT32 MaxSearchWindowY;

} VA2_Encode_SearchBounds;

Elementos

DetectSceneChange

[00141] Se esse valor é um, então a detecção da mudança da cena está sendo solicitada. Em um tal caso, se a mudança da cena é detectada, nenhum vetor de movimento será calculado pela chamada Execute e, portanto, nenhum resíduo ou imagens decodificadas será calculado também. Isso é indicado através do parâmetro pStatus da chamada Execute que deve ser ajustada para E_SCENECHANGE nesse caso.

MaxDistanceInMetric

[00142] Refere-se à diferença entre macroblocos quando comparações são feitas usando a métrica de distância de escolha atual. Se essa distância excede esse valor de MaxDistanceInMetric, então um tal vetor de movimento é rejeitado.

TimeLimit

[00143] Tempo máximo que o hardware pode gastar no estágio da estimativa do movimento. Se ele é mais longo do que esse tempo, o parâmetro pStatus da chamada Execute é ajustado para E_TIMEOUT.

E_TIMEOUT.

[00144] Valor máximo do componente x do vetor de movimento retornado. Em outras palavras, o tamanho (ao longo da dimensão x) da janela de pesquisa.

SearchWindowY

[00145] Valor máximo do componente y do vetor de movimento. Em outras palavras, o tamanho (ao longo da dimensão y) da janela de pesquisa.

Observações

3.3.14 VA2_Encode_ModeType

typedef struct_VA2_Encode_ModeType {

UINT32 SearchProfileindex;

GUID DistanceMetric;

INT16 HintX;

INT16 HintY;

} VA2_Encode_ModeType;

Elementos

SearchProfileindex

[00146] Índice na lista de perfis de pesquisa como retornado pela chamada da API GetSearchProfiles.

DistanceMetric

[00147] Métrica a usar quando comparando dois macroblocos. Métricas geralmente usadas incluem SAD (soma de diferenças absolutas) e SSE (soma de erros quadrados).

HintX

[00148] Palpite sobre a direção esperada de movimento para guiar a pesquisa do movimento. Isso se refere ao movimento geral na imagem e pode não ser aplicável em uma base por MB.

HintY

[00149] Palpite sobre a direção esperada de movimento para guiar a pesquisa do movimento. Isso se refere ao movimento geral na imagem e pode não ser aplicável em uma base por MB.

3.3.15 ConfigurationParameter Quantization

typedef struct_VA2_Encode_ExecuteConfiguration Parameter_ Quantization {

UINT32 Length;

UINT32 Streamid;

UINT32 ConfigurationType;

UINT32 StepSize;

}

VA2_Encode_ExecuteConfigurationParameter_Quantization;

Elementos

Length

[00150] Número de bytes nessa estrutura. Provido por extensibilidade.

StreamId

[00151] O ID do fluxo de dados como definido na configuração da canalização. Isso pode ser usado para deduzir se os dados são de entrada ou de saída.

ConfigurationType

StepSize

[00152] Tamanho da etapa a ser usada quando executando a quantização. Esse projeto permite que somente um tamanho de etapa seja usado para toda a porção da imagem para a qual vetores de movimento e resíduos foram solicitados nessa chamada.

3.4 Métodos: IVideoEncoderService

[00153] Os métodos nessa interface permitem que uma aplicação consulte o hardware por suas capacidades e crie um objeto de codificador com uma dada configuração.

3.4.1 GetPipelineConfigurations

HRESULT GetPipelineConfigurations(

[out] UINT32* pCount,

(out, unique, size_is(*pCount)] GUID** pGuids

};

Parâmetros

pCount

[00154] Valor de retorno descreve o tamanho da formação pGuids (próximo parâmetro) retornado pela função.

pGuids

[00155] Uma formação de GUIDs descrevendo as várias configurações de canalização suportadas pelo dispositivo. A memória é alocada pelo chamado, e deve ser liberada pelo chamador usando CoTaskMemFree.

Valores de retorno

S_OK

[00156] Função foi bem-sucedida

E_OUTOFMEMORY

[00157] Função foi incapaz de alocar memória para retornar a lista de GUIDs

E_FAIL

[00158] Incapaz de determinar as configurações de canalização suportadas por causa de algum erro de dispositivo.

3.4.2 GetFormats

HRESULT GetFormats(

[out] UINT32* pCount,

(out, unique, size_is(*pCount)] GUID** pGuids

};

Parâmetros

pCount

[00159] Valor de retorno descreve o tamanho da formação de pGuids (próximo parâmetro) retornado pela função.

pGuids

[00160] Uma formação de GUIDs descrevendo os vários formatos suportados pelo dispositivo (por exemplo, WMV9, MPEG-2, etc.). A memória é alocada pelo chamado, e deve ser liberada pelo chamador usando CoTaskMemFree.

3.4.3 GetDistanceMetrics

HRESULT GetDistanceMetrics(

[out] UINT32* pCount,

(out, unique, size_is(*pCount)] GUID** pGuids

);

Parâmetros

pCount

[00161] Valor de retorno descreve o tamanho da formação de pGuids (próximo parâmetro) retornado pela função.

pGuids

[00162] Uma formação de GUIDs descrevendo as várias métricas de pesquisa suportadas pelo dispositivo para estimativa do movimento. A memória é alocada pelo chamado e é liberada pelo chamador usando CoTaskMemFree.

Valores de retorno

S_OK

[00163] Função foi bem-sucedida

E_OUTOFMEMORY

[00164] Função foi incapaz de alocar memória para retornar a lista de GUIDs

E_FAIL

[00165] Incapaz de determinar a métrica suportada por causa de algum erro de dispositivo.

3.4.4 GetSearchProfiles

HRESULT GetSearchProfiles(

[out] UINT32* pCount,

(out, unique, size_is(*pCount)] VA2_Encode_SearchProfile** pSearchProfiles

);

Parâmetros

pCount

[00166] Valor de retorno descreve o tamanho da formação de pGuids (próximo parâmetro) retornado pela função.

pSearchProfiles

[00167] Uma formação de GUIDs representando os perfis de pesquisa suportados pelo dispositivo. Os perfis de pesquisa permitem trocas de qualidade-tempo de aplicação do codec mais efetivamente. A

memória é alocada pelo chamado e é liberada pelo chamador usando CoTaskMemFree.

Valores de retorno

S_OK

[00168] Função foi bem-sucedida

E_OUTOFMEMORY

[00169] Função foi incapaz de alocar memória para retornar a lista de GUIDs

E_FAIL

[00170] Incapaz de determinar os perfis de pesquisa suportados por causa de algum erro de dispositivo.

3.4.5 GetMECapabilities

HRESULT GetMECapabilities(

[out] VA2_Encode_MECaps* pMECaps

);

Parâmetros

pMECaps

[00171] Um ponteiro para as capacidades de estimativa do movimento do dispositivo. Isso inclui informação sobre o tamanho da imagem que o dispositivo pode manipular, o máximo tamanho de janela de pesquisa e se o dispositivo suporta tamanhos de macrobloco variáveis. A memória para isso é alocada pelo chamador.

Valores de retorno

S_OK

[00172] Função foi bem-sucedida

E_FAIL

[00173] Função falhou devido a algum erro de dispositivo.

3.4.6 CreateVideoEncoder

[00174] Essa função cria uma instância de IVideoEncoder.

HRESULT CreateVideoEncoder(

[in] REFGUID PipelineGuid,

[in] REFGUID FormatGuid,

[in] UINT32 NumStreams,

[in] VA2_Encode_StaticConfiguration* pConfiguration,

[in, size_is(NumStreams)] VA2_Encode_StreamDescrip-tion* pStreamDescription,

[in, size_is(NumStreams)] VA2_Encode_Allocator* SuggestedAllocatorProperties,

[out, size_is(NumStreams)] VA2_Encode_Allocator* pActualAllocatorProperties,

[out] IVideoEncoder** ppEncode

);

Parâmetros

PipelineGuid

[00175] Um GUID representando a configuração de canalização desejada. A lista de configurações é obtida através de GetCapabilities, e cada um dos GUIDs é associado com a documentação pública que descreve detalhes necessários sobre a configuração.

FormatGuid

[00176] Um GUID representando o formato do fluxo de bits codificado eventual. Muitas das operações de codificação como transformação e quantização têm elementos de formato específico para elas. Embora esses elementos de formato específico possam ser manipulados pela CPU com velocidade suficiente, a troca de informação precisará do uso de estágios extras de canalização e tornará mais difícil obter alta eficiência da canalização.

NumStreams

[00177] Número de fluxos de dados de entrada e saída associados com a configuração da canalização. Isso é implicado pelo GUID da canalização em muitos casos.

pConfiguration

[00178] Um ponteiro para as propriedades de configuração estática.

pStreamDescription

[00179] Uma formação de estruturas, uma por fluxo, que descreve os dados fluindo através desse fluxo.

SuggestedAllocatorProperties

[00180] O chamador (aplicação do codec) sugere um certo número de armazenamentos temporários (superfícies) a serem associados com os fluxos diferentes com base no seu projeto de canalização.

pActualAllocatorProperties

[00181] O acionador especifica o tamanho de fila do alocador atual com base nos recursos que ele é capaz de reunir e outras considerações. A suposição é que a aplicação abortará o uso dessa interface se ela não pode construir uma canalização eficiente com o armazenamento temporário (tamanho de fila do alocador) disponível.

ppEncode

[00182] Objeto do codificador de saída. O chamador deve considerar isso para ser um objeto COM regular a ser liberado através de IUnknown::Release.

Valores de retorno

S_OK

[00183] Função bem-sucedida.

E_FAIL

[00184] Função falha (provavelmente por carência de recursos)

3.5 Estruturas de dados: Data Structures: IVideoEncoderService

3.5.1 VA2_Encode_MECaps

typedef struct_VA2_Encode_MECaps

BOOB VariableM13SizeSupportedr

BOOL MotionvectorHintSupported;

UINT16 MaxSearchWindowX;

UINT16 MaxSearchWindowY;

UINT32 MaxImageWidth;

UINT32 MaxImageHeight;

UINT32 MaxMBSizeX;

UINT32 MaxMBSizeY;

1 VA2_Encode_MECaps;

Elementos

Yana bleMBSizeSupported

[00185] Um valor de um indica que o hardware suporta tamanhos de macrobloco variáveis quando executando estimativa de movimento. Em particular, se tamanhos de macrobloco variáveis são suportados, é legal que o chamador dessa API ajuste ConstantMBSize para zero na estrutura VA2_Encode_MBDescription, e use o parâmetro pMBRectangles para descrever a partição da imagem.

Motion VectorHintSupported

[00186] Um valor de um indica que o hardware é capaz de usar alguns palpites do chamador no seu algoritmo de pesquisa de movimento. Em particular, o chamador pode ajustar os elementos HintX e HintY de VA2_Encode_ModeType que é um parâmetro de configuração do Execute.

MaxSearch WindowX

[00187] Valor legal máximo de SearchWindowX, um elemento de VA2_Encode_SearchBounds, que é um parâmetro de configuração de estimativa de movimento.

MaxSearch Window I'

[00188] Valor legal máximo de SearchWindowY, um elemento de VA2_Encode_SearchBounds, que é um parâmetro de configuração de estimativa de movimento.

MaximageWidth

[00189] Largura máxima permitida da imagem de entrada.

MaximageHeigh t

[00190] Altura máxima permitida da imagem de entrada.

MaxMBSizeX

[00191] Largura máxima permitida do macrobloco.

MaxMBSizeY

[00192] Altura máxima permitida do macrobloco.

3.5.2 VA2_Encode_StaticConfiguration

typedef struct_VA2_Encode_StaticConfiguration (

GUID TransformOperator;

GUID PixelInterpolation;

GUID Quantization;

UINTB NumMotionVectorsPerMB;

VA2_Encode MVLayout MVLayout

VA2_Encode__ResidueLayout ResLayout;

} VA2_Encode_StaticConfiguration;

Elementos

Transform Operator

[00193] Um GUID representando o operador da transformação - um de MPEG-2 DCT, transformação WMV9, etc.

PixelInterpolation

[00194] Um GUID representando o esquema de interpolação de sub-pixel a ser usado. Os sistemas de interpolação bicúbico e bilinear têm um número de coeficientes que são específicos do formato.

Quantization

[00195] Um GUID representando o esquema de quantização a ser usado.

NumMotion VectorsPerMB

[00196] O número de vetores de movimento a ser calculado por macrobloco. As configurações de canalização simples suportadas por versões anteriores dessa interface podem exigir que esse valor seja

um.

MVLayout

[00197] O leiaute da superfície do vetor de movimento.

ResidueLayout

[00198] O leiaute da superfície de resíduo.

3.5.3 VA2_Encode_Allocator

typedef struct_VA2_Encode_Allocator {

BOOL Externa1Allocator;

UINT32 NumSurfaces;

} VA2_Encode_Allocator;

Elementos

ExternalAllocator

[00199] Falso indica que armazenamentos temporários são obtidos através de GetBuffer enquanto verdadeiro indica que os armazenamentos temporários são obtidos via um alocador externo, ou que não existe alocador associado com o fluxo em questão. A configuração da canalização força o valor desse campo em muitos casos (freqüentemente para zero). Uma exceção notável está no fluxo da imagem de entrada que pode vir de um alocador externo.

NumSurfaces

[00200] Número de superfícies a serem associadas com a fila do alocador.

3.5.4 VA2_Encode_StreamDescription

typedef struct_VA2_Encode_StreamDescription {

UINT32 Length;

UINT32 StreamType;

} VA2_Encode_StreamDescription;

Elementos

Length

[00201] Comprimento de toda a estrutura usada para validar este-

reótipos e permitir a extensibilidade.

StreamType

[00202] Descreve o tipo de dados associados com esse fluxo. Usado para estereotipar.

Observações

[00203] Essa estrutura de base é estereotipada para um tipo derivado no campo StreamType. Os estereótipos são descritos na documentação para VA2_Encode_StreamType.

3.5.5 VA2_Encode_StreamType

#define VA2_Encode_StreamType Video0x1

#define VA2_Encode_StreamType_MV0x2

#define VA2_Encode_StreamType_Residues0x3

Descrições de tipo

VA 2Encode StreamType_Video

[00204] A estrutura de descrição de fluxo associada pode ser modelada para VA2_Encode_StreamType MV

[00205] A estrutura de descrição de fluxo associada pode ser modelada para VA2_Encode_Stream_Type_Residues

[00206] A estrutura de descrição de fluxo associada pode ser modelada para VA2_Encode_Stream_Type_Residues.

3.5.6 VA2_Encode_StreamDescription_Video

typedef struct_VA2_Encode_StreamDescription {

UINT32 Length;

UINT32 StreamType;

VA2_VideoDesc_VideoDescription;

} VA2_Encode_StreamDescription;

Elementos

Length

[00207] Comprimento de toda a estrutura usada para validar estereótipos e permitir a extensibilidade.

StreamType

[00208] Descreve o tipo de dados associados com esse fluxo. Usado para estereotipar.

VideoDescription

[00209] Descreve várias propriedades do fluxo de vídeo incluindo as dimensões, taxa de quadros, primários de cor, etc.

3.5.7 VA2_Encode_StreamDescription_MV

typedef struct_VA2_Encode_StreamDescription {

UINT32 Length;

UINT32 StreamType;

VA2_Encode_MVType_MVType;

VA2_Encode_MVLayout_MVLayout;

} VA2_Encode_StreamDescription;

Elementos

Length

[00210] Comprimento de toda a estrutura usada para validar estereótipos e permitir a extensibilidade.

StreamType

[00211] Descreve o tipo de dados associados com esse fluxo. Usado para estereotipar.

MVType

[00212] Descreve o tipo de estrutura do vetor de movimento usado para retornar dados de movimento. Usado na interpretação dos conteúdos da superfície do vetor de movimento.

MVLayout

[00213] Descreve como as estruturas do vetor de movimento para uma dada imagem de entrada são dispostas na memória.

3.5.8 VA2_Encode_StreamDescriptionResidues

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;

Elementos

Length

[00214] Comprimento de toda a estrutura usada para validar estereótipos e permitir a extensibilidade.

StreamType

[00215] Descreve o tipo de dados associados com esse fluxo. Usado para estereotipar.

SamplingType

[00216] Especifica se os dados de resíduos são 4:4:4, 4:2:2, etc.

LumaWidth

[00217] Largura da superfície de brilho.

LumaHeight

[00218] Altura da superfície de brilho.

ChromaCbWidth

[00219] Largura da superfície contendo os valores de resíduo Cb.

ChromaCbHeight

[00220] Altura da superfície contendo os valores de resíduo Cb.

ChromaCrWidth

[00221] Largura da superfície contendo os valores de resíduo Cr.

ChromaCbHeight

[00222] Altura da superfície contendo os valores de resíduo Cr.

3.6 Estruturas de dados: Vetores de movimento

3.6.1 Leiaute do vetor de movimento

[00223] A figura 9 mostra dados do vetor de movimento exemplar armazenados em uma superfície D3D, de acordo com uma modalidade. Cada uma das células descritas como “MV” é uma estrutura do vetor de movimento. Representações diferentes são usadas dependendo dos valores de VA2_Encode_MVType e VA2_Encode_MVLayout. As estruturas reais e os leiautes são descritos abaixo.

3.6.2 Novos formatos D3D

```
typedef enum_D30FORMAT
{
D3DFMT_MOTIONVECTOR16 = 105,
D3DFMT_MOTIONVECTOR32 = 106,
D3DFMT_RESIDUE1      =107,
} D3DFORMAT;
```

[00224] Superfícies dos vetores de movimento e superfícies de resíduo são associadas com os novos tipos de formato D3D acima que indicam o tamanho dos vetores de movimento individuais e resíduos. Essa informação de tamanho é usada pelo acionador quando a aplicação cria superfícies usando uma das APIs de superfície ou de criação de recurso provida por. O indicador de recurso associado com as superfícies de codificação é VA2_EncodeBuffer.

```
// Huffer Type
enum
{
VA2_EncodeBuffer = 7,
typedef struct_D3DDDI_RESOURCEFLAGS
{
union {
struct
```

{

UINTTextApi1;// 0x20000000

UINTEncodeBuffer1;// 0x40000000

DINTReserved1;// 0x80000000 };

UINTValue;

};

} D3DDDI_RESOURCEFLAGS;

3.6.3 VA2_Encode_MVSurface

[00225] Essa estrutura é efetivamente derivada de IDirect3DSurface9 e transporta informação de estado que permite que alguém interprete os conteúdos da superfície D3D embutida.

typedef struct_VA2_Encode_MVSurface {

IDirect3DSurface9*pMVSurface;

VA2_Encode_MVTypeMVType;

VA2_Encode_MVLayout MVLayout;

GUID DistanceMetric;

} VA2_Encode_MVSurface;

Elementos

pMVSurface

[00226] Ponteiro para uma superfície D3D contendo vetores de movimento.

MVType

[00227] Esse valor é usado para identificar a estrutura (VA2_Encode_MotionVector8, etc.) com a qual interpretar os vetores de movimento individuais.

MVLayout

[00228] Esse valor identifica como as estruturas individuais do vetor de movimento são dispostas na superfície D3D.

DistanceMetric

[00229] Um GUID representando a métrica de distância usada para

medir a distância entre dois macroblocos. A métrica de distância é usada para identificar o macrobloco mais próximo e, portanto, o vetor de movimento ótimo.

3.6.4 VA2_Encode_MVType

[00230] Esse valor de enumeração é usado para decodificar os conteúdos da superfície D3D9 do vetor de movimento. Dependendo do tipo do vetor de movimento, uma de várias estruturas de vetor de movimento diferentes é usada para interpretar os conteúdos da superfície.

typedef enum {

VA2_Encode MVType_Simple8,

VA2_Encode_MVType_Simplel6,

VA2_Encode_MVType_Extended8,

VA2_Encode_MVType_Extended16

} VA2_Encode_MVType;

Descrição

VA2_Encode_MVType_Sirnple8

[00231] A estrutura do vetor de movimento é VA2_Encode_ MotionVector8.

VA2 Encode_MVType_Sirnplel6

[00232] A estrutura do vetor de movimento é VA2_Encode_ MotionVector16.

VA2_Encode_MVType_Extended8

[00233] A estrutura do vetor de movimento é VA2_Encode_ MotionVectorEx8.

VA2_Encode_MVType_Extendedl6

[00234] A estrutura do vetor de movimento é VA2_Encode_ MotionVectorEx16.

3.6.5 VA2_Encode_MVLayout

typedef enum {

VA2_Encode_MVLayout_A,

VA2_Encode_MVLayout_B,

VA2_Encode_MVLayout_C

} VA2_Encode_MVLayout;

Descrição

Tipo A

[00235] A superfície D3D real é uma formação de estruturas do vetor de movimento indexadas pelo índice do macrobloco e índice da linha.

Tipo B

[00236] Esse é um leiaute acondicionado onde o número de vetores de movimento por macrobloco não é constante. Detalha TBD.

Tipo C

3.6.6 VA2_Encode_MotionVector8

typedef struct_VA2_Encode_MotionVector8 {

INTOx;

INTOy;

} VA2_Encode_MotionVector8;

Elementos

x

coordenada x do vetor de movimento.

y

coordenada y do vetor de movimento.

3.6.7 VA2_Encode_MotionVectorl6

typedef struct VA2_Encode_MotionVectorl6 {

INT16 x;

INT16 Y;

} VA2_Encode_MotionVectorl6;

Elementos

x

coordenada x do vetor de movimento.

y

coordenada y do vetor de movimento.

3.6.8 VA2_Encode_MotionVectorEx8

typedef struct_VA2_Encode_MotionVectorEx8 {

INT8x;

INTBY;

UINT8 ImageIndex;

UINT8 Distance;

} VA2_Encode_MotionVectorEx8;

Elementos

x

coordenada x do vetor de movimento.

y

coordenada y do vetor de movimento.

ImageIndex

[00237] Um índice com base em zero na lista das imagens de referência que foi provido na chamada para os vetores ComputeMotion

Distance

[00238] a unidade de medição é especificada pelo campo DistanceMetric de VA_Encode_MVSurface. Ela mede a distância do macrobloco atual com o macrobloco de referência citado pelo vetor de movimento real (x, y).

3.6.9 VA2_Encode_MotionVectorExl6

typedef struct_VA2-Encode_MotionVectorExl6 {

INT16x;

INT16y;

UINT16 ImageIndex;

UINT16 Distance;

} VA2_Encode_MotionVectorExl6;

Elementos

x

coordenada x do vetor de movimento.

y

coordenada y do vetor de movimento.

ImageIndex

[00239] um índice com base em zero na lista das imagens de referência que foi provido na chamada para os vetores ComputeMotion

Distance

[00240] a unidade de medição é especificada pelo campo DistanceMetric de VA_Encode_MVSurface. Ela mede a distância do macrobloco atual com o macrobloco de referência citado pelo vetor de movimento real (x, y).

3.7 Estruturas de dados: resíduos

[00241] A superfície do resíduo é uma formação de valores de número inteiro assinados que são de dois bytes de comprimento - em outras palavras, eles são do tipo INT16. Esse esquema parece ser adequado em todos os casos de importância prática. Por exemplo, MPEG-2 lida com 9 valores de resíduo de bit e H.264 lida com 12 resíduos de bit. Também, se os dados originais eram YUY2, os valores de brilho ocupam um byte cada, e, portanto, os resíduos usam 9 bits (0 – 255 = -255). Além do que, a aplicação de uma transformação do tipo DCT aumenta a exigência de dados para 11 bits por valor de resíduo. Todos esses casos são manipulados adequadamente usando valores de resíduo assinados de 2 bytes de comprimento.

[00242] A largura de uma superfície de resíduo é o número de valores de resíduo em uma linha. Por exemplo, uma imagem progressiva de 640x480 com amostragem 4:2:2 tem 640 valores de brilho e 320 valores de saturação por linha. O tamanho da superfície de brilho associada é 640x480x2 e essa da superfície de saturação é 320x480x2

bytes.

[00243] As superfícies do resíduo são criadas usando o indicador de formato D3DFMT_RESIDUE16 e o tipo de recurso VA2_EncodeBuffer.

3.7.1 Plano de brilho

[00244] A figura 10 mostra um diagrama exemplar indicando que a largura da superfície de brilho iguala a imagem YCbCr original. Por exemplo, uma imagem de 640x480 tem 480 valores de brilho por linha e assim a largura da superfície de brilho é 480. Então, o tamanho da superfície de brilho é 640x480x2 bytes.

[00245] Plano = VA2_Encode_Residue_Y

[00246] Amostragem = VA2_Encode_SamplingType_*

3.7.2 Saturação 4:2:2

[00247] A figura 11 mostra um diagrama exemplar indicando que o número do valor de resíduo por linha de vídeo é metade com a largura da imagem de vídeo original, de acordo com uma modalidade. Assim, para uma imagem de 640x480, o número de valores de resíduo por linha e, portanto a largura da superfície é 320.

[00248] Plano = VA2_Encode_Residue_U ou VA_Encode_Residue_V

[00249] Amostragem = VA2_Encode_SamplingType 422

3.7.3 Saturação 4:2:0

[00250] Nesse cenário, a largura da superfície do resíduo é metade da largura do quadro progressivo original, e a altura é metade também. Assim, para uma imagem de 640x480, a própria superfície de saturação seria de 320 de largura e 240 de comprimento.

[00251] Plano = VA2_Encode_Residue_U ou VA_Encode_Residue_V

[00252] Amostragem = VA2_Encode_SamplingType 420

4 Documentação DDI exemplar

[00253] Dispositivos de extensão são um mecanismo de passagem provido pelas interfaces VA a fim de adicionar nova funcionalidade além das funções existentes de decodificador de vídeo e processador de vídeo. Por exemplo, eles serão usados para suportar uma nova função do codificador de vídeo.

[00254] Dispositivos de extensão agem como um funil não tipificado através do qual a aplicação pode enviar/receber dados para/do acionador. O significado dos dados é desconhecido para a pilha VA, e é interpretado pelo acionador com base no parâmetro pGuid da chamada CreateExtensionDevice e o parâmetro da função de ExtensionExecute.

[00255] A codificação VA usa o valor de GUID seguinte (o mesmo que o uuid de IVideoEncoder):

{7AC3D93D-41FC-4c6c-A1CS-A875E4F57CA4}

DEFINE_GUID(VA_Encoder_Extension, Ox7ac3d93d, Ox4lfc, Ox4c6c, Oxal, Oxcb, OxaB, 0x75, Oxe4, OxfS, Ox7c, Oxa4);

4.1 Enumeração e capacidades

[00256] Dispositivos de extensão são enumerados usando FND3DDDIGETCAPS com o parâmetro de tipo sendo ajustado para GETEXTENSIONGUIDCOUNT ou GETEXTENSIONGUIDS. A aplicação do codec busca VA_Encoder_Extension na lista de guids de extensão retornada por GETEXTENSIONGUIDS para determinar se o suporte da codificação VA está disponível.

4.1.1 FND3DDDLGETCAPS

```
typedef HRESULT
(APIENTRY *PFND3DDDI_GETCAPS)
(
HANDLE hAdapter,
CONST DIDDDIARG GETCAPS*
);
```

[00257] Quando consultando capacidades do dispositivo de extensão (o dispositivo do codificador), o GETEXTENSIONCAPS é usado com a estrutura seguinte como pInfo na estrutura D3DDDIARG GETCAPS.

4.1.2 VADDI_QUERYEXTENSIONCAPSINPUT

typedef etruct_VADDI_QUERYEXTENSIONCAPSINPUT {

CONST GUID*pGuid;

UINTCapType;

VADDI_PRIVATEDATA*pPrivate;

} VADDI_QUERYEXTENSIONCAPSINPUT;

O parâmetro pGuid de VADDI_QUERYEXTENSIONCAPSINPUT é ajustado para VA_Encoder_Extension.

#define VADDI_Encode_Captype_Guids

VADDI_EXTENSION CAPTYPE MIN

#define VADDI_Encode_Captype_DietanceMetrics

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

[00258] A saída de GETEXTENSIONCAPS é encapsulada no parâmetro pData de D3DDDIARG_GETCAPS. O parâmetro pData é interpretado como segue:

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

Captype_DistanceMetrics: Type = (GUID *). DataSize = sizeof(GUID) * quid count.

Captype_SearchProfiles: Type = (VADDI_Encode_ SearchProfile *) . DataSize = sizeof(VADDI_Encode_ SearchProftle).

Captype_MECaps: Type = (VADDI_Encode_MECaps). DataSize = sizeof(VADDI Encode MECaps).

4.1.3 D3DDDIARG_CREATEEXTENSIONDEVICE

[00259] A criação real acontece através de uma chamada D3DDDI CREATEEXTENSIONDEVICE, cujo argumento primário é mostrado abaixo:

```
typedef struct_D3DDDIARG_CREATEEXTENSIONDEVICE
{
CONST GUID**pGuid;
VADDI_PRIVATEDATA*pPrivate;
HANDLEhExtension;
} D3DDDIARG_CREATEEXTENSIONDEVICE;
```

4.2 Funcionalidade de codificação

[00260] As funções da unidade de extensão reais são invocadas através de uma chamada D3DDDI EXTENSIONEXECUTE. A instância da unidade de extensão já está associada com um GUID, então o tipo da unidade de extensão já é conhecido quando a chamada de execução é feita. O único parâmetro adicional é Function que indica a operação particular a executar. Por exemplo, um dispositivo de extensão do tipo codificador pode suportar MotionEstimation como uma de suas funções. Tipicamente, o dispositivo de extensão terá uma função GetCaps sua própria que enumera as capacidades do dispositivo de extensão.

```
typedef struct_D3DDDIARG_EXTENSIONEXECUTE
{
HANDLEhExtension;
UINTFunction;
VADDI_PRIVATEDATA*pPrivateInput;
VADDI_PRIVATEDATA*pPrivateOutput;
UINTNumBuffers;
```

VADDI_PRIVATEBUFFER*pBuffers;

} D3DDDIARG_EXTENSIONEXECUTE;

[00261] O parâmetro pBuffers não é usado pela codificação VA e deve ser considerado um parâmetro reservado. O parâmetro Function adota os seguintes valores para codificação VA:

#define VADDI_ Encode_Function_Execute 1

[00262] Os parâmetros pPrivateInput e pPrivateOutput de D3DDDIARG_EXTENSIONEXECUTE são usados para encapsular os parâmetros da chamada da API de execução. Os parâmetros específicos da codificação embutidos nos parâmetros de entrada e saída abaixo ainda não foram mapeados da região da API para a região da DDI - mas isso é apenas uma questão de renome, e nós poderíamos ser capazes de gerenciar com uma única definição.

4.2.1 VADDI_Encode_Function_Execute_Input

[00263] Esse parâmetro contém os parâmetros de entrada para a chamada da API de execução.

typedef struct_VADDI_Encode_Function_Execute_Input

{

UINT32 NumDataParameters;

VA2_Encode_ExecuteDataParameter** pData;

UINT32 NumConfigurationParameters;

VA2_Encode_ExecuteConfigurationParameter** pConfiguration;

} VADDI_Encode_Function_Execute_Input;

4.2.2 VADDI_Encode_Function_Execute_Output

[00264] Essa estrutura encapsula os dados de saída da chamada de execute.

typedef struct_VADDI_Encode_Function_Execute_Output

{

UINT32 NumDataParameters;

VA2_Encode_ ExecuteDataParameter** pData;

} VADDI_Encode_Function_Execute_Output;

4.3 Estruturas do dispositivo de extensão

[00265] As seções seguintes descrevem várias estruturas e as chamadas de autenticação de função associadas com o mecanismo de extensão VA.

4.3.1 VADDI_PRIVATEBUFFER

typedef struct -VADDI_PRIVATEBUFFER {

HANDLEhResource;

UINTSubResourceIndex;

UINTDataOffset;

UINTDataSize;

} VADDI_PRIVATEBUFFER;

typedef struct _VADDI_PRIVATEDATA {

VOID*pData;

UINTDataSize;

} VADDI_PRIVATEDATA;

4.3.2 D3DDDIARG_EXTENSIONEXECUTE

typedef struct _D3DDDIARG_EXTENSIONEXECUTE {

HANDLEhExtension;

UINTFunction;

VADDI_PRIVATEDATA*pPrivate2nput;

VADDI_PRIVATEDATA*pPrivateOUtpUt;

UINTNumBuffers;

VADDI_PRIVATEBUFFER*pBuffers;

} D3DDDIARG_EXTENSIONEXECUTE;

typedef HRESULT

(APIENTRY *PFND3DDDI-CREATEEXTENSIONDEVICE)

{

HANDLE hDevice,

D3DDDIARG_CREATEEXTENSIONDEVICE*

};

[00266] O parâmetro hDevice se refere a um dispositivo D3D9, e ele é criado usando uma chamada para D3DDDI_CREATEDEVICE.

4.3.3 FND3DDDI_DESTROYEXTENSIONDEVICE

typedef HRESULT

(APIENTRY

*PFND3DDDI_DESTROYEXTENSIONDEVICE)

(

HANDLE hDevice,

HANDLE hExtension

);

4.3.4 FND3DDDI_EXTENSIONEXECUTE

typedef HRESULT

(

(APIENTRY *PFND3DDDI_EXTENSIONEXECUTE)

HANDLE hDevice,

CONST D3DDDIARG_EXTENSIONEXECUTE*

);

4.3.5 D3DDDI_DEVICEFUNCS

typedef struct _D3DDDI DEVICEFUNCS

{

PFND3DDDI_CREATEEXTENSIONDEVICEpfnCreateEx-

tensionDevice;

PFND3DDDI_DESTROYEXTENSIONDEVICEpfnDe-

stroyExtensionDevice;

PFND3DDDI_EXTENSIONEXECUTEpfnExtensionExecute;

} D3DDDI_DEVICEFUNCS;

4.4 Estruturas e funções D3D9

[00267] As estruturas D3D seguintes e a chamada de autenticação

representam um mecanismo D3D genérico para obter as capacidades de um dispositivo de extensão.

```
typedef enum_D3DDDICAPS_TYPE
{
D3DDDICAPS_GETEXTENSIONGUIDCOUNT=31,
D3DDDICAPS_GETEXTENSIONGUIDS=32,
D3DDDICAPS_GETEXTENSIONCAPS=33,
} D3DDDICAPS_TYPE;
typedef struct _D3DDDIARG_GETCAPS
{
D3DDDICAPS_TYPEType;
VOID*pInfo;
VOID*pData;
UINTDataSize;
} D3DDDIARG_GETCAPS;
```

5 Modelo de programação exemplar

5.1 Eficiência da canalização

[00268] A fim de obter máxima eficiência, a aplicação do codificador é estruturada em uma tal maneira que ambos a CPU, bem como o hardware gráfico são totalmente utilizados. Assim, enquanto a estimativa do movimento está em progresso para um certo quadro, poderia ser benéfico executar a etapa de quantização em um quadro diferente.

[00269] A obtenção da utilização completa do hardware é facilitada com um codificador de múltiplos encadeamentos.

5.1.1 Exemplo: vetor de movimento único (canalização completa)

[00270] A seguinte aplicação encadeada em 2 (em pseudocódigo) ilustra uma maneira para o codificador implementar uma canalização de software de 2 estágios, e oferece algumas orientações sobre como usar as interfaces de codificação VA efetivamente. Em particular, ela

força um armazenamento temporário de k = AllocatorSize como observado no encadeamento do software. Isso responde pelo fato que existe assincronismo na submissão de uma solicitação de hardware: o encadeamento do hardware submete solicitações enquanto o encadeamento do software escolhe os resultados depois de um tempo e os processa.

```
HardwareThread()
{
while (Streaming)
{
LoadFrame(ppInputBuffer[n));
Codec->ProcessInput(ppInputBuffer[n]); If blocking GetBuffer + Execute
}
SoftwareThread() {
k = AllocatorSize();
while (Streaming)
{
// k represents the buffer between pipeline stages Codec->ProcessOutput(ppOutputBuffer(n-k]); // Wait, ReleaseBuffar VLE();
Bitstream();
}
```

[00271] ProcessInput acima pode ser considerado um envoltório ao redor de Execute e GetBuffer, enquanto ProcessOutput pode ser considerado um envoltório ao redor de um evento de Esperar na execução, seguido com chamadas de ReleaseBuffer apropriadas.

[00272] Não é evidente como determinar o parâmetro k que representa o armazenamento temporário entre os estágios da canalização. Ele representa o tamanho do alocador, e como um ponto de partida, nós podemos usar o mesmo valor usado na negociação do alocador

entre o Codec e o objeto do codificador VA (o comprimento da fila). Se k é maior do que o tamanho do alocador, então a chamada ProcessInput é provável de bloquear de qualquer forma mesmo antes que os armazenamentos temporários k sejam usados.

[00273] A meta da aplicação deve ser maximizar o tempo gasto no SoftwareThread sem bloquear no ProcessOutput. Em outras palavras, a aplicação deve estar funcionando nas funções VLE() e Bitstream() na maior parte do tempo. Se o hardware é muito lento, então ProcessOutput() bloqueará a despeito do tamanho do alocador de “k”. O software sempre estará “à frente”. A canalização acima é eficiente somente até a extensão que o hardware leva aproximadamente tanto tempo para processar um armazenamento temporário quanto o software leva para executar VLE e o fluxo de bits. Tudo o que o armazenamento temporário de “k” consegue é preencher as instabilidades.

[00274] O fragmento de código seguinte mostra uma implementação esboçada de GetBuffer e ReleaseBuffer.

```
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;
}
```

[00275] O seguinte esboça a implementação do codec de ProcessInput e ProcessOutput:

```
// essa implementação está bloqueando contrária à semântica normal
GetBuffer(TypeUncompressed, pYUVBuffer, true);
GetBuffer(TypeMotionVector, pMVBuffer, true);
GetBuffer(TypeResidues, pResidueBuffer, true);
memory(pYUVBuffer, pInput.Image);
Execute(pYUVBuffer, pMVBuffer, pResidueBuffer, pEvent);
CodecQueue.Enqueue(pYUVBuffer, pMVBuffer, pResidueBuffer, pEvent);
Codec::ProceseOutput(IMediaBuffer pOutput)
{
if (CodecQueue.Empty())
{
pOutput.dwFlags = DMO_OUTPUT_DATABUFFERF_INCOMPLETE;
```

return S,FALSE;

}

CodecQueue.Dequeue(pYUVBuffer, pMVBuffer, pResidueBuffer, pEvent);

Wait(pEvent);

memory(pOutput.MVBuffer, pMVBUffar);

memory(pOutput.ResidueBuffer, pResidueBuffer);

Release8uffer(TypeUncompressed, pYUVBuffer);

ReleaseBuffer(TypeMotionvector, pMVBuffer);

ReleaseBuffer(TypeResidues, pResidueBuffer);

return S _OK;

}

[00276] Aqui é uma implementação alternada de Codec:: ProcessInput que é não bloqueadora como é a norma.

Codec:: ProcessInput(IMediaBuffEr pInput)

{

if (GetBuffer(TypeUncompreesed, 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;

```
}
memory (pYUVBuffer, pInput.Image);
Execute(pYUVBuffer, pMVBuffer, pResidueBuffer, pEvent);
CodecQueue.Enqueue(pYUVBuffer, pMVBuffer, pResidueBuffer, pEvent);
```

5.1.2 Exemplo: vetores de movimento múltiplos

[00277] Nessa seção, nós observamos uma canalização mais complexa onde o codificador solicita múltiplos vetores de movimento do hardware e escolhe um baseado nos vários parâmetros e os submete novamente para processamento. O código seguinte simplesmente continua a usar uma canalização de 2 estágios como antes, solicita múltiplos vetores de movimento e submete novamente o melhor. Existe seriação inerente envolvida nisso.

```
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]);
}
}
```

[00278] No exemplo acima, o software será bloqueado em ProcessOutput e ProcessOutput2 metade do tempo, claramente ruim para a eficiência da canalização. Por outro lado, a utilização da CPU será muito baixa e a velocidade geral é ainda maior do que a codificação não acelerada. Uma canalização de 3 estágios com base em 3 encadeamentos resolverá o problema de seriação como segue:

```
HardwareThread1()
{
while (Streaming)
{
LoadFrame(ppInputBuffer[n]);
Process Input(ppInputBuffer[n]);
}
}
HardwareThread2()
{
while (Streaming)
{
ProcessOutput(ppOutputBuffer[n - ki]);
SelectMV(ppOutputBuffer[n - kl]; ppOutputBuffer2[n - kl]);
ProcessInput2(ppOutputBuffer2[n - kl]);
}
}
SoftwareThread()
```

```
{
while (Streaming)
{
Processoutput2(ppoutputBuffer2[n - kl - k2]);
VLE(ppOutputBuffer2[n - kl - k2]);
Bitstream(ppOutputBuffer2[n - kl - k2]);
}
}
```

[00279] Desde que existem 3 estágios de canalização, armazenamento temporário adicional é adicionado para enchimento entre os dois estágios do hardware. Portanto, os dois valores k1 e k2.

REIVINDICAÇÕES

1. Método pelo menos parcialmente implementado por um ou mais processadores de um dispositivo de computação, o método **caracterizado pelo fato de que** compreende as etapas de:

receber, por um serviço de aceleração de codificação de vídeo implementado pelos um ou mais processadores do dispositivo de computação, uma ou mais consultas de um codificador de vídeo para identificar condições específicas de implementação do hardware de aceleração;

responsivo à recepção das uma ou mais consultas, o serviço de aceleração de codificação de vídeo:

fazer interface com o hardware de aceleração para obter as condições específicas de implementação;

responsivo à recepção das condições específicas de implementação, comunicar as condições específicas de implementação para o codificador de vídeo; e

em que as condições específicas de implementação possibilitam que o codificador de vídeo durante o tempo de execução:

(a) determine se uma ou mais de velocidade e qualidade das operações de codificação de software associadas com o codificador de vídeo podem ser aumentadas com a implementação de uma canalização de codificação particular das uma ou mais configurações e capacidades de canalização de codificação suportadas; e

(b) implemente a canalização de codificação particular pela interface com o serviço de aceleração de codificação de vídeo;

receber, pelo serviço de aceleração de codificação de vídeo, uma solicitação incluindo um conjunto de parâmetros de configuração para criar um objeto de codificador que implementa a canalização de codificação particular; e

responsivo à recepção da solicitação, criar o objeto com

base nos parâmetros de configuração, o objeto do codificador para codificar dados de vídeo de origem decodificados usando a canalização de codificação particular.

2. Método, de acordo com a reivindicação 1, **caracterizado pelo fato de que** as operações de codificação de software compreendem uma ou mais de estimativa do movimento, computação de resíduo, compensação de movimento e operações de transformação.

3. Método, de acordo com a reivindicação 1, **caracterizado pelo fato de que** as operações de codificação de software compreendem uma ou mais de redução do ruído, estabilização da imagem, detecção de borda, nitidez e operações de conversão de taxa de quadros.

4. Método, de acordo com a reivindicação 1, **caracterizado pelo fato de que** as uma ou mais consultas compreendem uma consulta de obter capacidades, e onde as condições específicas de implementação recebidas incluem informação associada com as uma ou mais configurações de canalização de codificação suportadas.

5. Método, de acordo com a reivindicação 1, **caracterizado pelo fato de que** as uma ou mais consultas compreendem uma consulta de obter métricas de distância, e onde as condições específicas de implementação recebidas incluem uma descrição de uma ou mais métricas de pesquisa suportadas pelo hardware de aceleração de codificação de vídeo para operações de estimativa de movimento.

6. Método, de acordo com a reivindicação 1, **caracterizado pelo fato de que** as uma ou mais consultas compreendem uma consulta de obter perfis de pesquisa, e onde as condições específicas de implementação recebidas incluem uma descrição de um ou mais perfis de pesquisa suportados pelo hardware de aceleração de codificação de vídeo, os um ou mais perfis de pesquisa permitindo que o codificador de vídeo avalie as permutas específicas da implementação entre

os tempos de processamento de codificação de vídeo e as métricas de qualidade de codificação de vídeo.

7. Método, de acordo com a reivindicação 1, **caracterizado pelo fato de que** a uma ou mais consultas compreendem uma consulta de obter capacidades de estimativa de movimento, e onde as condições específicas de implementação recebidas incluem dados indicando um ou mais de tamanho máximo de imagem suportada, tamanho máximo de janela de pesquisa suportada e uma indicação de se o hardware de aceleração suporta tamanhos de macrobloco variáveis.

8. Método, de acordo com a reivindicação 1, **caracterizado pelo fato de que** os parâmetros de configuração especificam um ou mais dentre a canalização de codificação particular, um formato de saída para o vídeo codificado, um número de fluxos de dados de I/O para associação com a canalização de codificação particular, propriedades de configuração estática para interpolação de valores de brilho e saturação, um número sugerido de armazenamentos temporários de dados para os fluxos de dados de I/O e um tamanho de fila especificada pelo acionador de dispositivo com base nos recursos disponíveis.

9. Método, de acordo com a reivindicação 1, **caracterizado pelo fato de que** ainda compreende:

receber, pelo serviço de aceleração de codificação de vídeo, solicitações de execução e um conjunto de parâmetros do codificador de vídeo, as solicitações de execução correspondendo com operações associadas com a canalização de codificação particular para codificar os dados de vídeo de origem decodificados,

responsivo à recepção das solicitações de execução, o serviço de aceleração de codificação de vídeo:

comunicar as solicitações de execução e os parâmetros para o hardware de aceleração;

receber respostas associadas com as solicitações de exe-

cução comunicadas do hardware de aceleração; e

enviar as respostas para o codificador de vídeo.

10. Método, **caracterizado pelo fato de que** compreende as etapas de:

comunicar, por um módulo de programa de codificador de vídeo implementado por um ou mais processadores configurados com instruções executáveis, uma ou mais solicitações para um serviço de aceleração de codificação de vídeo para identificar capacidades de uma ou mais das configurações de canalização de codificação de vídeo e capacidades suportadas pelo hardware de aceleração; e

responsivo à recepção das capacidades do serviço de aceleração de codificação de vídeo, o codificador de vídeo:

identifica, com base nas capacidades, uma ou mais operações de codificação de vídeo associadas com o codificador de vídeo que se beneficiará de uma ou mais de velocidade e qualidade se implementadas pelo hardware de aceleração;

solicita, pelo codificador de vídeo, o serviço de aceleração de codificação de vídeo para criar uma canalização de codificação de vídeo personalizada para implementar as uma ou mais operações de codificação de vídeo via o hardware de aceleração, tal que quaisquer operações de codificação de vídeo restantes são implementadas em software; e

direciona o serviço de aceleração de codificação de vídeo para criar a canalização de codificação de vídeo personalizada tal que o fluxo de dados entre a memória do sistema e a memória do dispositivo gráfico é minimizado.

11. Método, de acordo com a reivindicação 10, **caracterizado pelo fato de que** as uma ou mais operações de codificação de vídeo compreendem uma ou mais de estimativa do movimento, computação de resíduo, compensação de movimento e operações de

transformação.

12. Método, de acordo com a reivindicação 10, **caracterizado pelo fato de que** as uma ou mais operações de codificação de vídeo compreendem uma ou mais de redução de ruído, estabilização da imagem, detecção de borda, nitidez e operações de conversão de taxa de quadros.

13. Método, de acordo com a reivindicação 10, **caracterizado pelo fato de que** ainda compreende:

receber, pelo codificador de vídeo, dados de vídeo de origem codificados ou decodificados; e

se os dados de vídeo de origem recebidos estão codificados, pelo menos parcialmente decodificar, pelo codificador de vídeo, os dados de vídeo de origem para gerar dados de vídeo de origem decodificados para a codificação por um objeto de codificação criado pelo serviço de aceleração de codificação de vídeo, o objeto de codificação implementando a canalização de codificação de vídeo personalizada.

14. Método, de acordo com a reivindicação 10, **caracterizado pelo fato de que** ainda compreende codificar dados de vídeo de origem decodificados usando a canalização de codificação de vídeo personalizada.

15. Dispositivo de computação, **caracterizado pelo fato de que** compreende:

um processador acoplado a uma memória; e

um módulo mantido na memória e executado no processador para implementar um serviço de aceleração de codificação de vídeo para:

receber uma ou mais consultas de um codificador de vídeo, as uma ou mais consultas solicitando que o serviço de aceleração de codificação de vídeo identifique condições específicas de implementa-

ção do hardware de aceleração, as condições específicas de implementação para possibilitar que o codificador de vídeo:

(a) determine se uma ou mais dentre velocidade e qualidade das operações de codificação de software associadas com o codificador de vídeo podem ser aumentadas com implementação de uma canalização de codificação particular de uma ou mais configurações e capacidades de canalização de codificação suportadas, e

(b) implemente a canalização de codificação particular via o serviço de aceleração de codificação de vídeo para codificar dados de vídeo de origem decodificados;

consultar o hardware de aceleração para obter as condições específicas de implementação;

comunicar as condições específicas de implementação recebidas do hardware de aceleração para o codificador de vídeo;

receber uma solicitação de criar objeto de codificador do codificador de vídeo para criar um objeto de codificador que implementa a canalização de codificação particular;

receber uma ou mais solicitações de execução do codificador de vídeo para implementar operações associadas com a canalização de codificação particular no hardware de aceleração; e

enviar a informação associada com as uma ou mais solicitações de execução para o hardware de aceleração para codificar os dados de vídeo de origem decodificados.

16. Dispositivo de computação, de acordo com a reivindicação 15, **caracterizado pelo fato de que** as operações de codificação de software compreendem uma ou mais de estimativa de movimento, computação de resíduo, compensação de movimento e operações de transformação.

17. Dispositivo de computação, de acordo com a reivindicação 15, **caracterizado pelo fato de que** as operações de codificação

de software compreendem uma ou mais de redução de ruído, estabilização de imagem, detecção de borda, nitidez e operações de conversão de taxa de quadros.

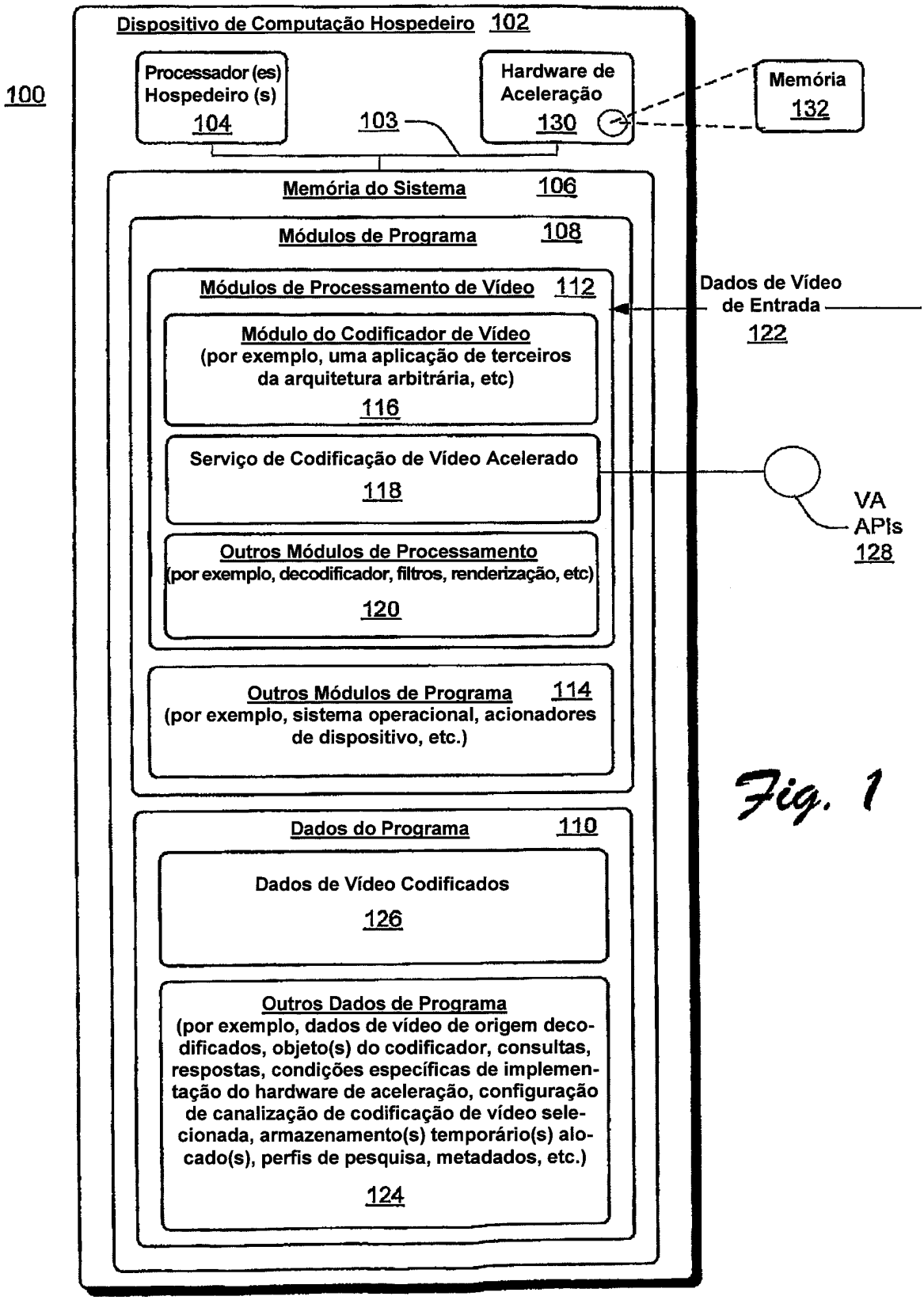
100
Dispositivo de Computação Hospedeiro 102
Processador (es) Hospedeiro (s) 104
103
Hardware de Aceleração 130
Memória 132
Memória do Sistema 106
Módulos de Programa 108
Módulos de Processamento de Vídeo 112
Módulo do Codificador de Vídeo (por exemplo, uma aplicação de terceiros da arquitetura arbitrária, etc) 116
Serviço de Codificação de Vídeo Acelerado 118
Outros Módulos de Processamento (por exemplo, decodificador, filtros, renderização, etc) 120
Outros Módulos de Programa 114 (por exemplo, sistema operacional, acionadores de dispositivo, etc.)
Dados de Vídeo de Entrada 122
VA APIs 128
Dados do Programa 110
Dados de Vídeo Codificados 126
Outros Dados de Programa (por exemplo, dados de vídeo de origem decodificados, objeto(s) do codificador, consultas, respostas, condições específicas de implementação do hardware de aceleração, configuração de canalização de codificação de vídeo selecionada, armazenamento(s) temporário(s) alocado(s), perfis de pesquisa, metadados, etc.) 124

Fig. 1

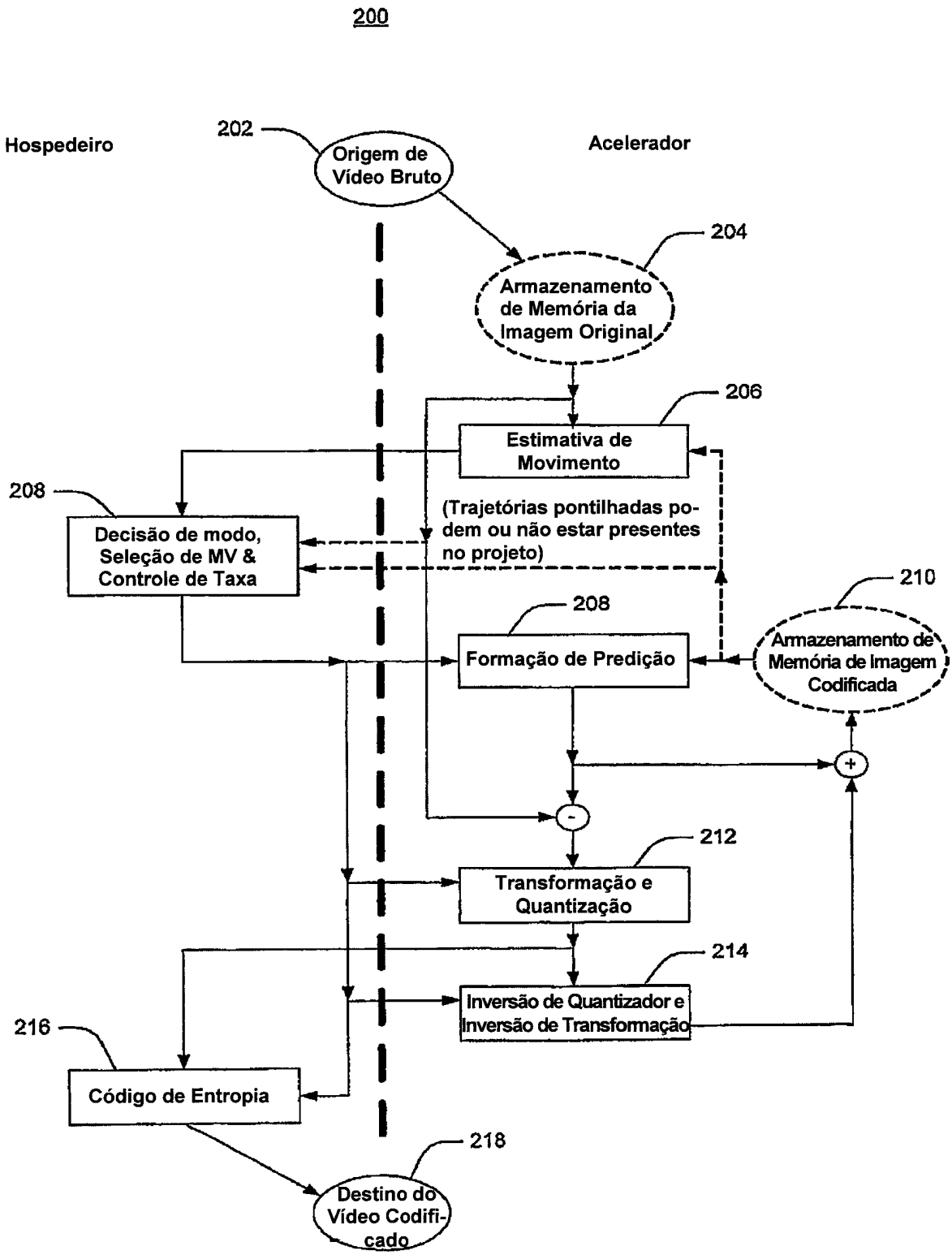
200
Hospedeiro
Acelerador
202
Origem de Vídeo Bruto
204
Armazenamento de Memória da Imagem Original
206
Estimativa de Movimento
208
Decisão de modo, Seleção de MV & Controle de Taxa
(Trajetórias pontilhadas podem ou não estar presentes no projeto)
210
Armazenamento de Memória de Imagem Codificada
208
Formação de Predição
212
Transformação e Quantização
214
Inversão de Quantizador e Inversão de Transformação
216
Código de Entropia
218
Destino do Vídeo Codificado

Fig. 2

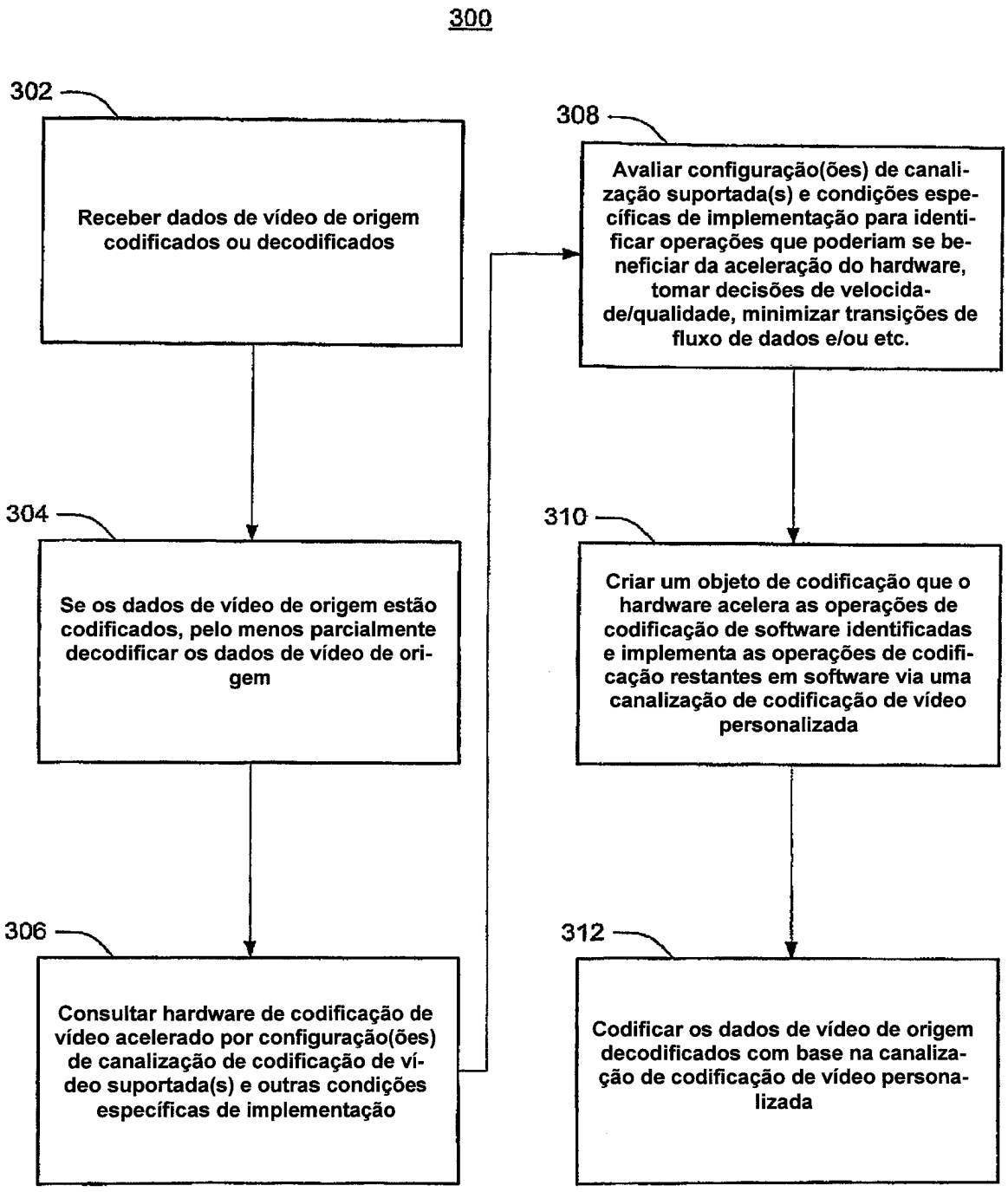
300
302
Receber dados de vídeo de origem codificados ou decodificados
304
Se os dados de vídeo de origem estão codificados, pelo menos parcialmente decodificar os dados de vídeo de origem
306
Consultar hardware de codificação de vídeo acelerado por configuração(ões) de canalização de codificação de vídeo suportada(s) e outras condições específicas de implementação
308
Avaliar configuração(ões) de canalização suportada(s) e condições específicas de implementação para identificar operações que poderiam se beneficiar da aceleração do hardware, tomar decisões de velocidade/qualidade, minimizar transições de fluxo de dados e/ou etc.
310
Criar um objeto de codificação que o hardware acelera as operações de codificação de software identificadas e implementa as operações de codificação restantes em software via uma canalização de codificação de vídeo personalizada
312
Codificar os dados de vídeo de origem decodificados com base na canalização de codificação de vídeo personalizada

Fig. 3

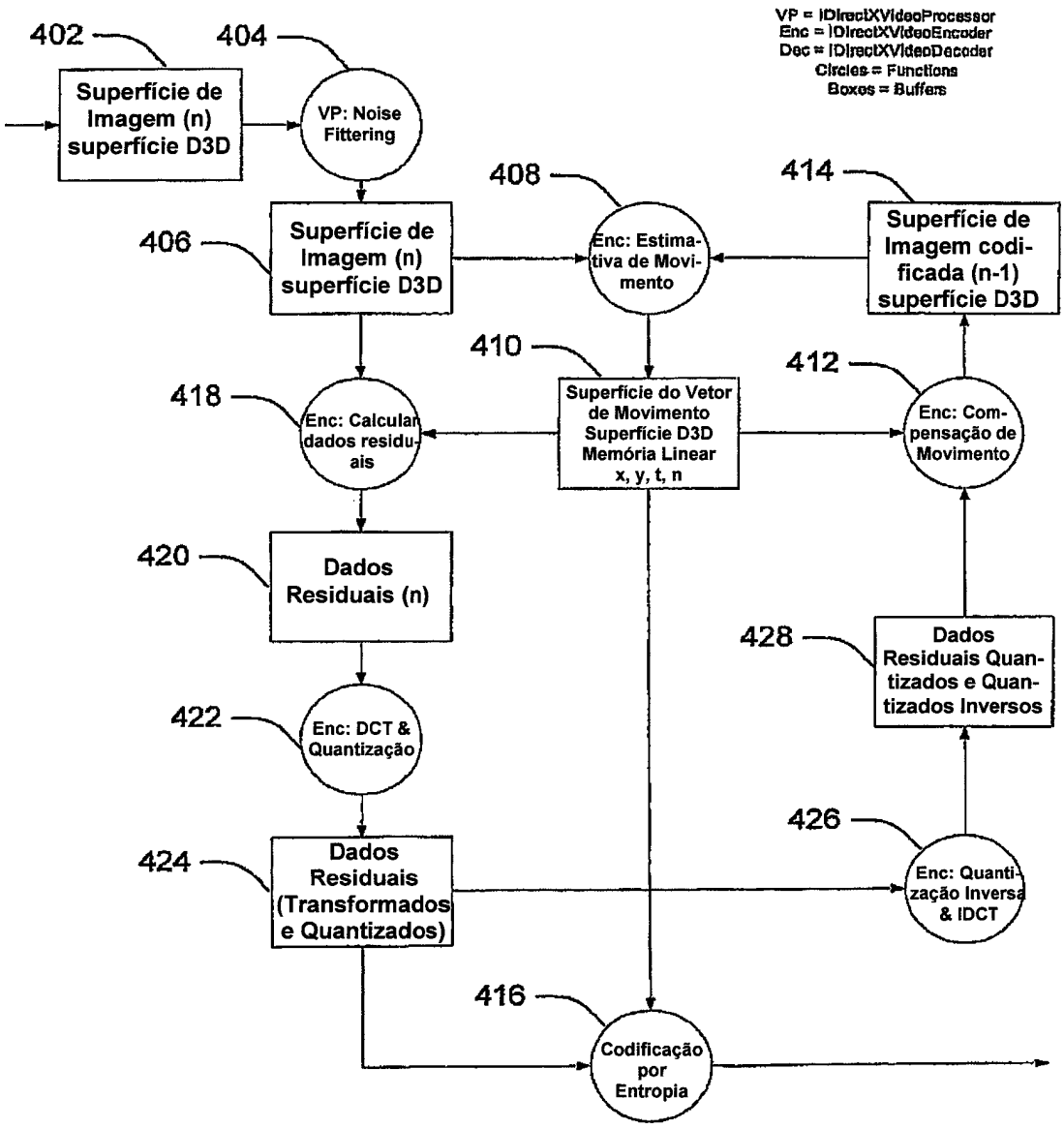
VP = IDirectXVideoProcessor
Enc = IDirectXVideoEncoder
Dec = IDirectXVideoDecoder
Circles = Functions
Boxes = Buffers
402
Superfície de Imagem (n) superfície D3D
404
VP: Noise Fittering
406
Superfície de Imagem (n) superfície D3D
408
Enc: Estimativa de Movimento
414
Superfície de Imagem codificada (n-1) superfície D3D
410
Superfície do Vetor de Movimento Superfície D3D Memória Linear x, y, t, n
412
Enc: Compensação de Movimento
418
Enc: Calcular dados residuais
420
Dados Residuais (n)
422
Enc: DCT & Quantização
424
Dados Residuais (Transformados e Quantizados)
428
Dados Residuais Quantizados e Quantizados Inversos
426
Enc: Quantização Inversa & IDCT
416
Codificação por Entropia

Fig. 4

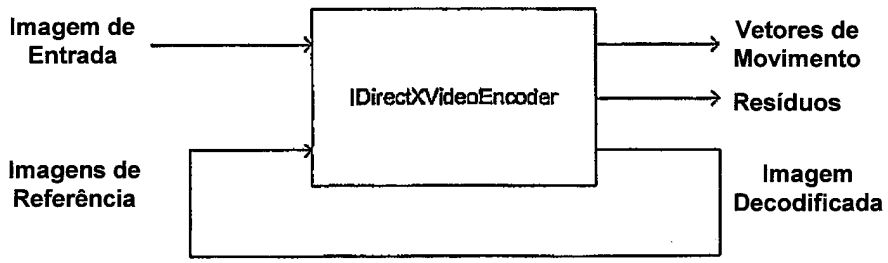

Fig. 5

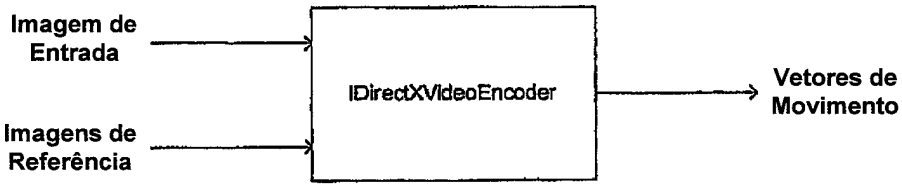

Fig. 6

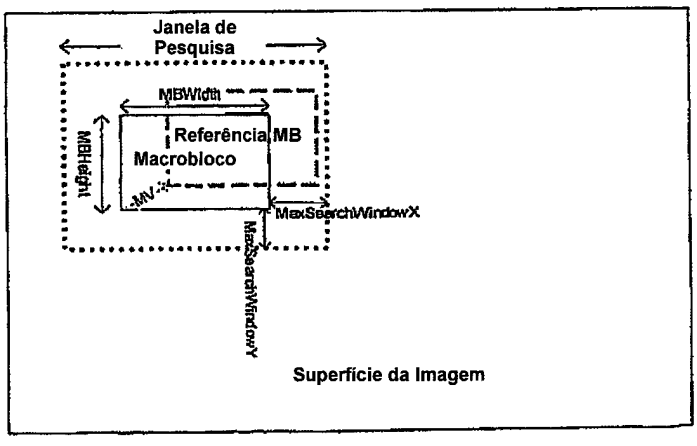

Fig. 7

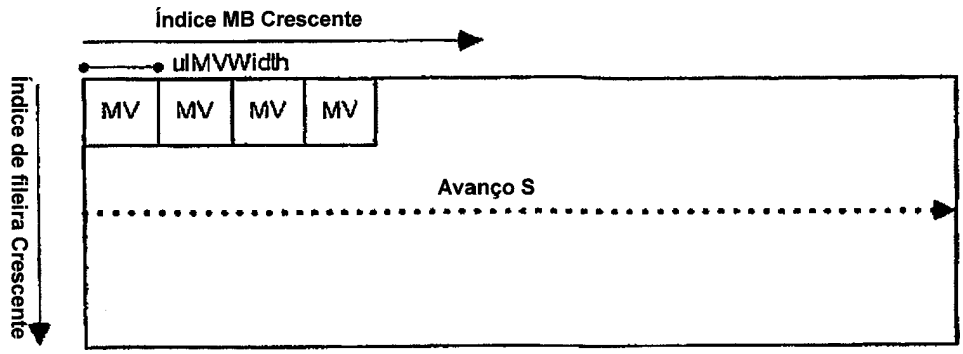

Fig. 8

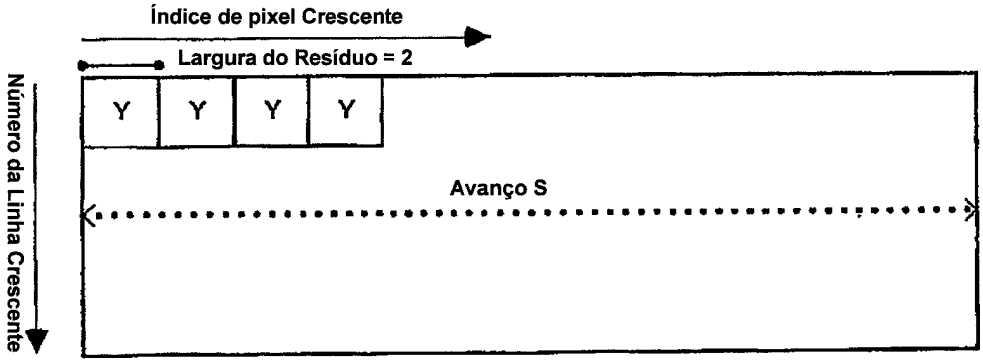

Fig. 9

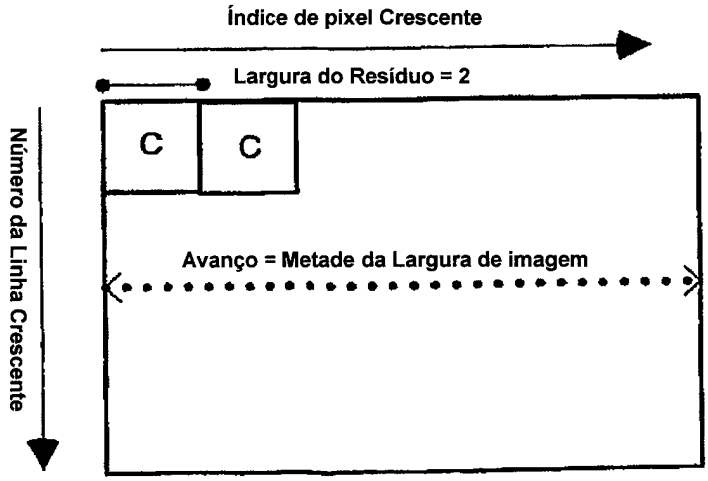

Fig. 10

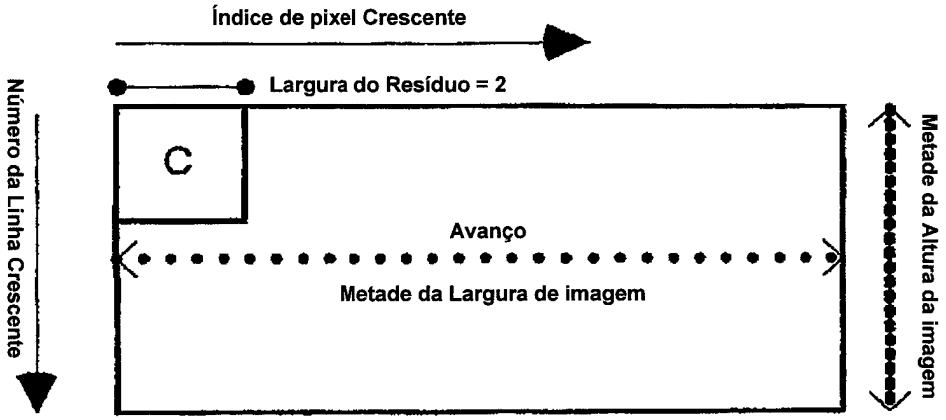
Índice de pixel Crescente
Largura do Resíduo = 2
Número da Linha Crescente
C
Avanço
Metade da Largura de imagem
Metade da Altura da imagem

Fig. 11