(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 101390397 B
(45)授权公告日 2010.09.15

(21)申请号 200780006564.4
(22)申请日 2007.02.21
(30)优先权数据
11/276,336 2006.02.24 US
11/673,423 2007.02.09 US
(85)PCT申请进入国家阶段日
2008.08.25
(86)PCT申请的申请数据
PCT/US2007/004638 2007.02.21
(87)PCT申请的公布数据
WO2007/100616 EN 2007.09.07
(73)专利权人 微软公司
地址 美国华盛顿州
(72)发明人 A·甘纳什 D·J·穆瑟尔
G·J·沙利文 G·F·伊万斯
S·萨德瓦尼 S·J·埃斯特罗普
(74)专利代理机构 上海专利商标事务所有限公司 31100
代理人 陈斌
(51)Int.Cl.
H04N 7/24(2006.01)
G06T 1/00(2006.01)
(56)对比文件
JP 10-322726 A,1998.10.04, 全文.
CN 1503122 ,2004.06.09, 全文.
US 4740073 A,1988.04.26, 全文.
US 2004/070556 A1,2004.04.15, 全文.
审查员 吴琼

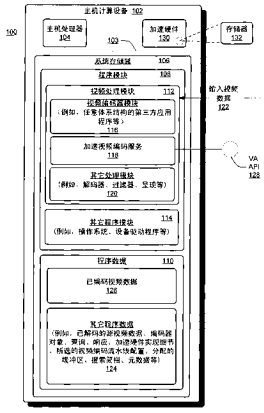
权利要求书 3 页 说明书 43 页 附图 7 页

(54)发明名称

加速视频编码

(57)摘要

描述了一种提高视频编码的速度和质量中的一个或多个的视频编码加速服务。该服务用作任意视频编码器计算机程序应用程序和任意视频加速硬件之间的中介。该服务从视频编码器接收一个或多个查询以标识视频加速硬件的实现细节。该服务与视频加速硬件接口以获得实现细节。该服务将实现细节传送到视频编码器。实现细节使得视频编码器能够:(a)确定与视频编码器相关联的软件编码操作的速度和质量中的一个或多个是否能用一个或多个支持的编码流水线配置和能力的流水线的实现来提高;以及(b)通过与该服务接口来实现该流水线。



CN 101390397 B

1. 一种用于视频编码加速服务的、至少部分地由计算设备实现的方法，所述方法包括：

由所述视频编码加速服务从视频编码器接收一个或多个查询以标识加速硬件的实现细节；

响应于接收所述一个或多个查询，所述视频编码加速服务：

与所述加速硬件接口以获得所述实现细节；

响应于接收所述实现细节，将所述实现细节传送到所述视频编码器；以及

其中，所述实现细节使得所述视频编码器在运行时能够：

(a) 确定是否能用一个或多个支持的编码流水线配置和能力的特定编码流水线的实现来提高与所述视频编码器相关联的软件编码操作的速度和质量中的一个或多个；以及

(b) 通过与所述视频编码加速服务接口来实现所述特定编码流水线。

2. 如权利要求 1 所述的方法，其特征在于，所述软件编码操作包括运动估计、残差计算、运动补偿和变换操作中的一个或多个。

3. 如权利要求 1 所述的方法，其特征在于，所述软件编码操作包括降噪、图像稳定、边缘检测、锐化和帧速率转换操作中的一个或多个。

4. 如权利要求 1 所述的方法，其特征在于，所述一个或多个查询包括获得能力查询，并且其中所接收到的实现细节包括与所述一个或多个支持的编码流水线配置相关联的信息。

5. 如权利要求 1 所述的方法，其特征在于，所述一个或多个查询包括获得距离度量查询，并且其中所接收到的实现细节包括所述视频编码加速硬件支持的用于运动估计操作的一个或多个搜索度量的描述。

6. 如权利要求 1 所述的方法，其特征在于，所述一个或多个查询包括获得搜索简档查询，并且其中所接收到的实现细节包括所述视频编码加速硬件支持的一个或多个搜索简档的描述，所述一个或多个搜索简档允许所述视频编码器评估视频编码处理时间和视频编码质量度量之间的特定于实现的折衷。

7. 如权利要求 1 所述的方法，其特征在于，所述一个或多个查询包括获得运动估计能力查询，并且其中，所接收到的实现细节包括指示最大支持图像大小、最大支持搜索窗大小和加速硬件是否支持可变宏块大小的指示中的一个或多个的数据。

8. 如权利要求 1 所述的方法，其特征在于，还包括：

由所述视频编码加速服务接收包括一组配置参数的请求以创建实现所述特定编码流水线的编码器对象；以及

响应于接收到所述请求，基于所述配置参数来创建所述编码器对象，所述编码器对象用于使用所述特定编码流水线来对已解码的源视频数据进行编码。

9. 如权利要求 8 所述的方法，其特征在于，所述配置参数指定所述特定编码流水线、已编码视频的输出格式、用于与所述特定编码流水线相关联的多个 I/O 数据流的数量、用于内插亮度和色度值的静态配置特性、用于所述 I/O 数据流的建议的数据缓冲区数量、以及基于可用资源的设备驱动程序指定的队列大小中的一个或多个。

10. 如权利要求 1 所述的方法，其特征在于，还包括：

由所述视频编码加速服务从所述视频编码器接收执行请求和一组参数，所述执行请求对应于与所述特定编码流水线相关联的操作以对已解码的源视频数据进行编码；

响应于接收到所述执行请求，所述视频编码加速服务：

将所述执行请求和所述参数传送到所述加速硬件；

从所述加速硬件接收与所传送的执行请求相关联的响应；以及

将所述响应转发给所述视频编码器。

11. 一种用于视频编码加速服务的、至少部分地由计算设备实现的方法，所述方法包括：

由视频编码器程序模块将一个或多个请求传送到视频编码加速服务以标识加速硬件支持的视频编码流水线配置和能力中的一个或多个的能力；

响应于从所述视频编码加速服务接收到所述能力，所述视频编码器：

基于所述能力标识与所述视频编码器相关联的、在由所述加速硬件实现的情况下将在速度和质量中的一个或多个上获益的一个或多个视频编码操作；

由所述视频编码器请求所述视频编码加速服务创建定制视频编码流水线，用于经由所述加速硬件来实现所述一个或多个视频编码操作，使得任何其余的视频编码操作以软件来实现。

12. 如权利要求 11 所述的方法，其特征在于，所述一个或多个视频编码操作包括运动估计、残差计算、运动补偿和变换操作中的一个或多个。

13. 如权利要求 11 所述的方法，其特征在于，所述一个或多个视频编码操作包括降噪、图像稳定、边缘检测、锐化和帧速率转换操作中的一个或多个。

14. 如权利要求 11 所述的方法，其特征在于，所述方法还包括用于指示所述视频编码加速服务创建所述定制视频编码流水线，使得系统存储器和图形设备存储器之间的数据流最小化。

15. 如权利要求 11 所述的方法，其特征在于，还包括：

由所述视频编码器接收已编码或已解码的源视频数据；以及

如果所接收的源视频数据是已编码的，则由所述视频编码器至少部分地解码所述源视频数据以生成已解码的源视频数据以便由所述视频编码加速服务创建的编码对象来编码，所述编码对象实现所述定制视频编码流水线。

16. 如权利要求 11 所述的方法，其特征在于，所述方法还包括用于使用所述定制视频编码流水线来编码已解码的源视频数据。

17. 一种计算设备，包括：

供视频编码加速服务执行以下动作的接口装置：

从视频编码器接收一个或多个查询，所述一个或多个查询请求所述视频编码加速服务标识加速硬件的实现细节，所述实现细节用于使得所述视频编码器能够：(a) 确定是否能用一个或多个支持的编码流水线配置和能力的特定编码流水线的实现来提高与所述视频编码器相关联的软件编码操作的速度和质量中的一个或多个；以及 (b) 经由所述视频编码加速服务实现所述特定编码流水线以对已解码的源视频数据进行编码；

查询所述加速硬件以获得所述实现细节；以及

将从所述加速硬件接收到的实现细节传送到所述视频编码器。

18. 如权利要求 17 所述的计算设备，其特征在于，所述软件编码操作包括运动估计、残差计算、运动补偿和变换操作中的一个或多个。

19. 如权利要求 17 所述的计算设备，其特征在于，所述软件编码操作包括降噪、图像稳定、边缘检测、锐化、和帧速率转换操作中的一个或多个。

20. 如权利要求 17 所述的计算设备，其特征在于，所述接口装置还包括供所述视频编码加速服务执行以下动作的装置：

从所述视频编码器接收创建编码器对象请求以创建实现所述特定编码流水线的编码器对象；

从所述视频编码器接收一个或多个执行请求以在所述加速硬件中实现与所述特定编码流水线相关联的操作；以及

将与所述一个或多个执行请求相关联的信息转发到所述加速硬件以对所述已解码的源视频数据进行编码。

加速视频编码

[0001] 相关申请

[0002] 本申请是 2006 年 2 月 24 日提交的题为“Accelerated Video Encoding(加速视频编码)”的共同待审的美国专利申请第 11/276,336 号的部分延续,并且该申请通过引用结合于此。

[0003] 背景

[0004] 多媒体内容产生和分发操作通常包括视频编码。视频编码过程通常是非常数据和计算密集型的。结果,视频编码过程可能是非常耗时的。例如,软件编码器编码一高质量高清电影可能要花费数十小时。由于视频编码过程的质量和速度对于成功的多媒体内容产生和分发流水线而言是重要因素,因此提高能编码高质量视频内容的速度的系统和技术将是有用的。

[0005] 概述

[0006] 提供本概述是为了用简化的形式介绍将在以下详细描述中进一步描述的一些概念。本概述不旨在标识所要求保护的主题的关键特征或必要特征,也不旨在用于帮助确定所要求保护的主题的范围。

[0007] 鉴于以上原因,描述了一种提高视频编码的速度和质量中的一个或多个的视频编码加速服务。该服务担当任意视频编码器计算机程序应用程序和任意视频加速硬件之间的中介。该服务从视频编码器接收一个或多个查询以标识视频加速硬件的实现细节。该服务与视频加速硬件接口以获得该实现细节。该服务将该实现细节传送到视频编码器。该实现细节使得视频编码器能够:(a) 确定与视频编码器相关联的软件编码操作的速度和质量中的一个或多个是否能用一个或多个支持的编码流水线配置和能力的流水线的实现来提高,以及 (b) 通过与该服务交互来实现该流水线。

[0008] 附图简述

[0009] 在附图中,组件参考标号最左边的数字标识了该组件首次出现的特定附图。

[0010] 图 1 示出根据一个实施例的用于加速视频编码的示例性系统。

[0011] 图 2 示出了视频编码流水线配置的一个示例性实施例,其中某些编码过程在硬件中加速。

[0012] 图 3 示出根据一个实施例的用于加速视频编码的示例性过程。

[0013] 附录中的图 4 示出了根据一个实施例的示例性视频编码器应用程序,其示出了可以利用该视频编码器加速应用程序编程接口的方式。

[0014] 附录中的图 5 示出了根据一个实施例的一个示例性视频编码流水线配置,其中加速硬件加速了运动估计、变换、量化和反过程以产生已编码图像。

[0015] 附录中的图 6 示出了根据一个实施例的示例性视频编码流水线配置,其中硬件仅加速运动估计。

[0016] 附录中的图 7 示出了根据一个实施例的若干示例性运动估计参数。

[0017] 附录中的图 8 示出了根据一个实施例的储存在显示三维 (D3D) 表面中的示例性运动矢量数据。

[0018] 附录中的图 9 示出了根据一个实施例的指示亮度表面的宽度匹配原始 YCbCr 图像的示例性图示。

[0019] 附录中的图 10 示出了根据一个实施例的指示视频的每行残差值的数量是原始视频图像的宽度的 1/2 的示例性图示。

[0020] 附录中的图 11 示出了根据一个实施例的指示残差表面的宽度是原始逐行帧的宽度的 1/4 的示例性图示。

[0021] 详细描述

[0022] 概览

[0023] 用于加速视频编码的系统和方法提供了一种视频编码加速服务。该服务允许任意视频编码器应用程序以设备无关的方式与任意视频加速硬件接口以定义并实现基本上最优的视频编码流水线。为此，该服务展示了视频加速 (VA) 应用程序接口 (API)。这些 API 封装了视频编码过程的模型。为定义编码流水线，该视频编码器应用程序使用 VA API 来查询可用视频（图形）加速硬件的实现细节（例如能力等）。该视频编码器鉴于应用程序的特定视频编码体系结构（软件实现的）来评估这些细节以标识可从在硬件中加速而获益（例如，速度和 / 或质量上获益）的任何编码操作。这些操作包括，例如运动估计、变换和量化操作，以及诸如运动补偿、反变换和反量化等反操作。该 API 还允许视频编码器设计基本上最小化了与主机计算设备和加速硬件相关联的总线和处理器上的数据流转移的编码流水线，并且因此进一步提高了编码速度。该 API 还允许加速硬件影响数据的定位以改善本地高速缓存（例如，视频加速硬件可在对视频硬件本地的存储器上更高效地运作）。

[0024] 基于这些评估，该视频编码器设计了一种在软件中执行部分编码操作并使用加速硬件执行部分编码操作（即，可从硬件加速中获益的操作的至少一个子集）的定制视频编码流水线。该编码器应用程序然后使用该 API 来创建该流水线并编码视频内容。该定制流水线与完全软件实现的流水线相比基本得到优化，因为某些编码操作被加速，并且主机和加速硬件之间的数据转移被最小化。另外，通过加速编码过程的某些方面并最小化数据转移而释放的处理时间允许主机处理器以被释放的处理周期来执行更高质量的编码操作。该 API 还被设计成允许组件并行地操作以使计算资源使用能被最大化。

[0025] 用于加速视频编码的系统和方法的这些和其它方面现将更详细地讨论。

[0026] 示例性系统

[0027] 尽管并非所需，但用于加速视频编码的系统和方法在由诸如个人计算机和图形（视频）编码加速硬件等计算设备执行的计算机可执行指令（程序模块）的一般上下文中描述。程序模块一般包括执行特定任务或实现特定抽象数据类型的例程、程序、对象、组件、数据结构等。

[0028] 图 1 示出了根据一个实施例的用于加速视频编码的示例性系统 100。系统 100 包括主机计算设备 102。主机计算设备 102 表示任何类型的计算设备，诸如个人计算机、膝上型计算机、服务器、手持式或移动计算设备等等。主机计算设备 102 包括通过总线 103 耦合到存储器 106 的一个或多个处理单元 104。系统存储器 106 包括计算机程序模块（“程序模块”）108 和程序数据 110。处理器 104 从程序模块 108 中相应的模块中取出并执行计算机程序指令。程序模块 108 包括用于处理视频内容的视频处理模块 112，以及诸如操作系统、设备驱动程序（例如，用于接口到视频编码加速硬件等）和 / 或其它等的其它程序模块

114。视频处理模块 112 包括，例如视频编码器 116、视频编码加速服务 118、以及其它处理模块 120，例如视频解码器、视频过滤器和视频呈现器等等。

[0029] 在此实现中，视频编码器 116 是任意视频编码器。这意味着由视频编码器 116 实现和 / 或利用的特定体系结构、操作、数据格式等是任意的。例如，视频编码器 116 可以通过第三方、OEM 等来分发。另外，尽管图 1 示出了独立于操作系统的“其它程序模块”114 部分的视频编码加速服务 118，但是在一个实现中，视频编码加速服务 118 是操作系统的一部分。

[0030] 视频处理模块 112 接收压缩的或未压缩的输入视频数据 122。当输入视频数据 122 是压缩（已编码）的时候，视频处理模块 112 解码该输入视频数据 122 以产生已解码源视频数据。这一解码操作由解码器模块来执行。在另一实现中，也可保留部分解码的数据以便进一步协助编码过程。出于示例性说明的目的，这一解码器模块被示为“其它视频处理模块”120 的相应部分。由此，已解码源视频数据或者由在已解码状态下接收到的输入视频数据 122 来表示，或者用对在已编码状态下接收到的输入视频数据 122 进行解码的结果来表示。已解码源视频数据被示为“其它程序数据”124 的相应部分。

[0031] 为设计并实现能用于将已解码源视频数据编码成已编码视频数据 126 的定制视频编码流水线，视频编码器 116 经由视频加速 (VA)API 128 与视频编码加速服务 118 接口。VA API 128 的多种可能实现的一个示例性实现在附录中描述。为定义编码流水线，视频编码器应用程序使用 VA API 128 中相应的几个（例如，参见附录 §3.4，IVideoEncoderService）以获得可用加速硬件 130 的实现细节。这些实现细节包括，例如：

[0032] •标识加速硬件 130 的支持的视频编码流水线配置的枚举阵列（例如，经由在附录 §3.4.1 中描述的 GetCapabilities 接口获得）；

[0033] • 支持的视频格式的指示（例如，MPEG、WMV 等等；参见附录的 GetSupportedFormats，§3.4.2）；

[0034] •用于运动估计 (ME) 操作的支持的搜索度量（参见附录的 GetDistanceMetrics，§3.4.3）；

[0035] • 用于处理时间与质量之间的折衷判定的支持的搜索简档（参见附录的 GetSearchProfiles，§3.4.4）；和 / 或

[0036] • 支持的 ME 能力，例如图像大小信息、最大搜索窗大小、可变宏块支持指示等（参见附录的 GetMECapabilities，§3.4.5）

[0037] 响应于从视频编码器 116 接收到这些请求，视频编码加速服务 118 向视频加速硬件 130 查询所请求的实现细节并从加速硬件 130 向视频编码器 116 返回与对应的响应相关联的信息。视频编码加速服务 118 使用对应的设备驱动程序与视频加速硬件 130 接口。这一设备驱动程序被示为“其它程序模块”114 的相应部分。

[0038] 视频编码器 116 鉴于应用程序的特定视频编码体系结构（软件实现的）来评估加速硬件 130 所支持的实现细节以标识能从在硬件中加速而获益（例如，速度和 / 或质量上获益）的任何编码操作、选择一搜索简档以封装视频编码质量和速度之间的折衷、最小化总线上和处理器之间的数据转移等等。可从硬件加速获益的示例性操作可包括，例如运动估计、变换和量化。例如，在硬件中执行量化的一个原因是最小化流水线各阶段之间的数据流。

[0039] 图 2 示出了视频编码流水线配置的一个示例性实施例，其中某些编码过程在硬件中加速。出于示例性说明和描述的目的，与图 2 相关联的操作和数据流参考图 1 的组件中的特定几个组件来描述。在该描述中，一参考标号最左边的数字指示其中首次引入该组件 / 数据路径 / 引用的项目的特定附图。例如，流水线 200 的最左边的数字是“2”，指示它是在图 2 中首次引入的。在此示例中，编码流水线 200 是由与视频编码服务 118 接口的视频编码器 116（图 1）来配置 / 定制的，使得由主机 102 中的相应几个实现的处理操作在硬件 130 中加速。出于说明的目的，在图 2 中的加粗虚线的右侧所示的处理操作由硬件（例如，图 1 的加速硬件 130）来加速，而该图的左侧所示的处理操作由主机计算设备 102（图 1）来执行。在编码流水线 200 中，以未加粗的虚线示出了可任选地配置的数据访问通路。椭圆 204 和 212 表示相应的原始和已编码图像记忆存储。

[0040] 在该示例实现中，视频编码器 116（图 1）取某一形式的已压缩或未压缩视频数据 202（还请参见图 1 的输入视频数据 122）作为输入。请注意，如果源 202 不是源自主机 102 且如果主机决策制定引擎（例如，视频编码器 116）不使用源视频，则图 2 的示例性流水线配置并不将输入源视频 202（“原始视频源”）复制到主机计算设备 102。例如，如果量化决策不要求主机接触视频数据，则将不传输该数据。在此示例中，流水线 200 被配置成使用块 206、208 和 214 到 218 中的相应操作将输入数据 202 转换成另一压缩形式。

[0041] 这些操作可包括将未压缩 (YUV) 视频数据转换成压缩的 MPEG-2，或者它可包括将视频数据从 MPEG-2 数据格式译码成 WMV 数据格式。出于示例性说明的目的，假设译码操作包括完全或部分解压阶段后跟一编码阶段（存在绕过解压并完全在变换 (DCT) 空间中工作的更高效的模型）。多个视频压缩格式利用了运动估计、变换和量化来实现压缩。在压缩阶段中，运动估计通常是最慢的步骤，包括其中编码器（例如，视频编码器 116）试图为给定图像中的宏块找到最接近的匹配参考宏块的大量搜索操作。

[0042] 一旦对每一宏块确定（例如，经由框 206）了最优运动矢量，编码器 116 就基于先前编码的图像和最优运动矢量来计算差分残差（例如，经由框 208）。运动矢量连同差分残差一起是当前图像的紧凑表示。运动矢量数据被进一步差分地表示。主机编码器可任选地请求视频加速硬件重新评估运动矢量以找出具有更小的组合运动矢量和 / 或残差的宏块。所得的差分运动矢量数据和残差数据例如使用如行程长度编码 (RLE) 和差分编码（例如，哈夫曼和算术编码）等技术来压缩（例如，经由块 218），以生成最终的已编码比特流（已编码视频数据 126）以便传送到目的地（框 218）来向用户呈现。在此示例中，框 206、208 和 214 到 218 的操作（例如，诸如运动估计 (206)、模式判定、运动矢量 (MV) 选择和速率控制 (208)、预测形成 (210)、变换和量化操作 (214)、反量化器和变换及版本 (216) 以及熵编码 (218) 等操作）在本领域中是公知的，因此此处不再描述。

[0043] 再次参考图 1，在一个实现中，视频编码器 116 是为完全利用加速硬件 130 而提供的多线程应用程序。在此实现中，当确定哪些视频编码操作要在硬件中加速时，视频编码器 116 可以结构化特定流水线配置，使得处理器 104 和加速硬件 130 两者都得到完全利用。例如，当对于视频数据的一特定帧，视频编码流水线运动估计操作是由硬件来执行的时候，该流水线可被配置成由主机对该视频数据的一不同帧在软件中执行熵（或算术或哈夫曼）编码操作。表示所选 / 结构化的特定流水线配置的一个示例性单运动矢量流水线以下在附录的 5.1.1 节中描述。其中视频编码器 116 向加速硬件 130 请求多个运动矢量并基于各参

数来选择一个运动矢量流水线的示例性多运动矢量（相对复杂的）流水线在以下附录的5.1.2节中描述。

[0044] 关于选择搜索简档，运动矢量的质量指的是通过使用运动矢量生成的流的比特率。高质量运动矢量与低比特率流相关联。质量由块搜索的完整性、算法的质量、使用的距离度量等来确定。高质量运动矢量应用于执行高质量视频编码操作。为此，视频编码加速服务118提供一称为搜索简档（search profile）的通用构造以封装质量和时间之间的折衷。该搜索简档还包括标识加速硬件130使用的搜索算法等的元数据。视频编码器116基于编码器实现的特定要求选择一特定的搜索简档。

[0045] 关于最小化总线上和处理器之间的数据转移，由视频编码流水线配置实现的编码过程通常包括若干处理阶段，每一处理阶段可以经由或不经由加速硬件130来加速。在其中视频编码器116确定在编码流水线的连续阶段中利用硬件加速的情况下，可能不必将数据从基于加速硬件130的存储器132移至与主机计算设备102相关联的系统存储器106，然后移回基于加速硬件的存储器132以便执行下一阶段，依此类推。

[0046] 更具体地，尽管指向各种类型的视频和运动矢量数据的指针可以在主机计算设备102和加速硬件130之间来回传输，但是在一个实现中，实际数据仅在数据指针（D3D9表面指针）使用例如IDirect3DSurface9::LockRect明确地锁定时才被复制到系统存储器106。用于锁定表面的示例性接口是已知的（例如，公知的IDirect3DSurface9::LockRect接口）。由此，在两个编码流水线阶段彼此紧跟，并且主机计算设备102不需要执行任何中间处理的情况下，主机计算设备102可决定不“锁定”在处理阶段之间已分配的缓冲区。这将防止对数据的冗余的存储器复制，并且因此避免了不必要的数据移动/传输。以此方式，视频编码器116设计了基本上最小化了总线上以及处理器之间的数据传输，且因此进一步提高视频编码速度的视频编码流水线。

[0047] 此时，视频编码器116已经鉴于应用程序的特定视频编码体系结构（软件实现的）评估了加速硬件130所支持的实现细节以标识能从在硬件中加速获益的任何编码操作、选择了搜索简档、最小化了总线上以及处理器之间的数据转移、和/或其它等等。基于这些判定，视频编码器116选择一特定流水线配置来编码已加码的源视频数据，并因此生成已编码视频数据126。接着，视频编码器116与视频编码加速服务118接口以创建一编码器对象来实现所选的流水线（参见附录中的CreateVideoEncoder API，§3.4.6）。在此实现中，编码器对象（例如，常规的COM对象）是通过标识所选流水线配置以及以下的一个或多个来创建的：输出的已编码比特流的格式、与流水线配置相关联的输入和输出数据流的数量、静态配置特性、基于所选流水线配置用于与不同I/O流相关联的建议的缓冲区（表面）数量、以及基于图形驱动程序能够收集的资源的驱动程序指定的分配符队列大小、以及其它参数。（队列大小和数据缓冲区的数量本质上指的是相同的东西：一个是“建议的”，另一个是“实际的”）。

[0048] 接着，视频编码器116使用所创建的编码器对象来与视频编码加速服务118接口以对已解码的源视频数据进行编码。为此，编码器对象向加速硬件130提交执行请求（参见附录中的IVideoEncode:Execute API，§3.2.3）。

[0049] 鉴于以上原因，系统100允许视频编码器应用程序116的任意实现在运行时定义并创建视频编码流水线配置以完全利用可用的视频编码加速硬件来提高编码速度和质量。

作为这些运行时配置操作的一部分，视频编码器 116 可使用 VA API 128 来指定编码流水线要实现交互式有向搜索（细化递增的多遍搜索）、定义并使用一般可选择的搜索策略（例如，基于独立于关于所采用的实际算法的任何细节知识的质量度量来选择搜索算法）、利用格式无关方法（例如，其中视频编码器 116 不知道输入视频数据 122 的特定图像格式，且加速硬件 130 不知道已编码视频数据 126 的压缩的输出格式）来控制搜索、自适应数据大小（例如，其中视频编码器 116 基于搜索算法来选择宏块大小）等等。

[0050] 示例性过程

[0051] 图 3 示出根据一个实施例的用于加速视频编码的示例性过程 300。出于示例性描述的目的，参考图 1 的系统 100 的各组件描述该过程的操作。一组件参考标号最左边的标号指示首次描述该组件的特定附图。

[0052] 在框 302 处，视频编码器 116（图 1）接收输入视频数据 122。如果输入视频数据 122 是未压缩的，则该输入视频数据表示已解码的源视频数据。在框 304 处，如果输入视频数据 122 是压缩的，则视频编码器 116 解压该输入视频数据以生成已解压的源视频数据。在框 306 处，视频编码器 116 与 VA API 128 接口以向加速硬件 130 查询能力以及视频编码流水线配置实现细节。在框 308 处，视频编码器 116 在视频编码器 116 的实现的上下文内评估支持的能力和实现细节，以标识与可从硬件加速获益的视频编码器 116 的特定实现相关联的视频编码操作、作出编码速度和 / 或质量决策、最小化总线上以及处理器之间的数据转移和 / 或其它等等。

[0053] 在框 310 处，视频编码器 116 创建实现被配置成执行所标识的可从在加速硬件 130 中的硬件加速获益的视频编码操作的编码流水线的编码对象、实现速度 / 质量折衷（例如，经由所选的搜索简档）、并最小化数据流转移。在框 312 处，视频编码器使用所创建的编码器对象来根据由在框 310 处生成的定制视频编码流水线描绘的操作序列和编码体系结构对已解码源视频数据进行编码。框 312 的这些编码操作生成已编码视频数据 126（图 1）。

[0054] 结论

[0055] 尽管以对结构特征和 / 或方法操作或动作专用的语言描述了用于加速视频编码的系统和方法，但是可以理解，所附权利要求书中定义的实现不一定要限于所描述的具体特征或动作。

[0056] 例如，尽管图 1 的 API 128 是在编码视频数据的上下文中描述的，但是 API 128 可在编码上下文之外用于诸如边缘检测、基于运动矢量的降噪、图像稳定、锐化、帧速率转换、计算机视觉应用程序的速率计算等其它功能的硬件加速。例如，关于降噪，在一个实现中，视频编码器 116（图 1）对已解码的源图像数据的所有宏块计算运动矢量。然后，视频编码器 116 利用运动幅值、方向和与周围宏块的运动矢量的相关来确定在输入图像中是否有局部对象运动。在此实现中，视频编码器 116 然后利用矢量的幅值来指导特定对象的对象跟踪 / 过滤侵略性或平均移位以减少统计随机噪声。

[0057] 在关于图像稳定的另一示例中，在一个实现中，视频编码器 116 对所有宏块和已解码源数据计算运动矢量。视频编码器 116 然后确定图像中是否有全局运动。这通过将所有运动矢量值相关并确定相关的矢量值是否相似来实现。如果是，则视频编码器 116 得出存在全局运动的结论。或者，视频编码器 116 利用大的宏块大小并确定是否存在大宏块的总体运动。在确定是否存在全局运动之后，如果视频编码器 116 还发现全局运动矢量趋向

于跨帧急动，则视频编码器 116 得出存在照相机急动的结论并在开始噪声过滤和编码操作之前补偿这一急动。

[0058] 因此，系统 100 的具体特征和操作是作为实现所要求保护的主题的示例性形式而公开的。

[0059] 附录 A

[0060] 示例性视频编码加速 API

[0061] 视频编码

[0062] 本附录描述了用于加速视频编码（也称为 VA 编码）的视频编码加速 API128（图 1）的一个示例性实现的各方面。在此实现中，API 128 被设计成使得编码和视频处理应用程序（例如，视频编码器模块 116）能够利用加速硬件 130（例如，GPU）对加速运动估计、残差计算、运动补偿和变换的支持。

[0063] 1 目录

[0144] 2 示例性设计

[0145] 2.1 编码器布局

[0146] 图 5 示出了根据一个实施例的示例性视频编码器应用程序以示出可利用视频编码加速 API 的方式。在此示例中,视频编码器 116 是以 DMO 或 MFT 的形式来实现的。图 5 示出了在若干处理阶段之后的输入数据 (对应于“接收”) 和输出数据。各框表示数据,而圆圈表示由编码器应用程序调用的 API 函数。因此,圆圈表示编码器应用程序看到的 API 的核心。

[0147] 2.2 流水线或模式配置

[0148] 加速硬件被视为流水线,并使用流水线 GUID 来描述该流水线的最基本配置元素。编码加速的目标可被认为是与流水线效率的目标密切相关。

[0149] 该设计允许拆分的 (或多阶段) 流水线,其中在获得最终输出之前数据在主机 PC 和硬件之间来回传递。尚未设计出多阶段流水线配置,以下配置描述了非拆分的、单阶段流水线。

[0150] 2.2.1 VA2_EncodePipe_Full

[0151] //{BFC87EA2-63B6-4378-A619-5B451EDCB940}

[0152] cpp_quote(" DEFINE_GUID(VA2_EncodePipe_Full,0xbfc87ea2,0x63b6,0x4378,0xa6,0x19,0x5b,0x45,

[0153] 0x1e,0xdc,0xb9,0x40) ; ")

[0154] 图 6 示出了根据一个实施例的示例性视频编码流水线配置,其中加速硬件加速运动估计、变换、量化和反过程以产生已解码图像。该硬件产生运动矢量、残差 (亮度和色度) 和已解码图像作为输出。已解码图像不需要被传送到系统存储器,因为其唯一的目的是计算用于下一帧的运动矢量。

[0155] 稍后的文献将谈及称为 NumStreams(流数) 的参数。对于此流水线配置,NumStreams 为 5。实际的 StreamId(流 ID) 在图中的括号中示出。

[0156] 这是单阶段的非拆分流水线,因此 Execute(执行) 的 Stage(阶段) 参数不适用。

[0157] 流描述

[0158] 注意，StreamType_* 是 VA2_Encode_StreamType_* 的缩写。

[0159] 输入图像

[0160] StreamID 是 1，且流类型是 StreamType_Video。该流表示对其寻找运动数据的图像。用于该流的分配符是可协商的 - 当前接口可以提供一个分配符，或者可使用外部分配符。对分配符的选择是在创建时作出的，并且如果选择了外部分配符，则流 ID 1 将被认为是对于 GetBuffer（获得缓冲区）的非法输入值。

[0161] 参考图像

[0162] 流 ID 为 2，并且流类型是 StreamType_Video。该流表示用于计算运动矢量的参考图像列表。当前接口不为该流提供单独的分配符。输入表面从已解码图像流 (ID ＝ 5) 中回收，或者从别处获得。

[0163] 运动矢量

[0164] 流 ID 是 3，且流类型是 StreamType_MV。该流表示包含运动矢量数据的输出参数。用于该流的缓冲区只能经由 GetBuffer 获得。

[0165] 残差

[0166] 流 ID 是 4，且流类型是 StreamType_Residues。该流表示包含用于所有三个平面的残差值的输出参数。用于该流的缓冲区只能经由 GetBuffer 获得。

[0167] 已解码图像

[0168] 流 ID 是 5，并且流类型是 StreamType_Video。该流表示包含从量化的残差和运动矢量值获得的已解码图像的输出参数。用于该流的缓冲区只能经由 GetBuffer 获得。

[0169] 2.2.2 VA2_EncodePipe_MotionEstimation

[0170] //{F18B3D19-CA3E-4a6b-AC10-53F8 6D509E04}

[0171] cpp_quote(" DEFINE_GUID(VA2_EncodePipe_MotionEstimation,0xf18b3d19, 0xca3e,0x4a6b,0xac,0x10,

[0172] 0x53,0xf8,0x6d,0x50,0x9e,0x4) ; ")

[0173] 图 7 示出了根据一个实施例的示例性视频编码流水线配置，其中硬件仅加速运动估计。该流水线配置取一组参考图像作为输入，并转储运动矢量作为输出。在此情况下的已解码图像必须由主机软件来生成和提供。

[0174] 用于此流水线配置的 NumStreams 是 3。用于各种流的 SteamId 在图中的括号中示出。

[0175] 这是单阶段、非拆分的流水线，并且 Excute 的 Stage 参数不适用。

[0176] 3 示例性 API

[0177] 3.1 接口定义

[0178] 3.1.1 IVideoEncoder

[0179] interface IVideoEncoder:IUnknown

[0180] {

[0181] HRESULT GetBuffer(

[0182] [in] UINT8 StreamId,

[0183] [in] UINT32 StreamType,

```
[0184]          [in]  BOOL Blocking,
[0185]          [out]  PVOID pBuffer
[0186]          );
[0187]      HRESULT  ReleaseBuffer(
[0188]          [in]  UINT8 StreamId,
[0189]          [in]  UINT32 StreamType,
[0190]          [in]  PVOID pBuffer
[0191]          );
[0192]      HRESULT Execute(
[0193]          [in]  UINT8 Stage,
[0194]          [in]  UINT32 NumInputDataParameters,
[0195]          [in,size_is(NumInputDataParameters)]
[0196]              VA2_Encode_ExecuteDataParameter * * pInputData,
[0197]          [in]  UINT32 NumOutputDataParameters,
[0198]          [out,size_is(NumOutputDataParameters]
[0199]              VA2_Encode_ExecuteDataParameter * * pOutputData,
[0200]          [in]  UINT32 NumConfigurationParameters,
[0201]          [in, size_is(NumConfigurationParameters]
[0202]              VA2_Encode_ExecuteConfigurationParameter * * pConfiguration,
[0203]          [in]  HANDLE hEvent,
[0204]          [out]  HRESULT * pStatus
[0205]          );
[0206]  }
[0207]  3.1.2  IVideoEncoderService
[0208]  interface IVideoEncoderService:IVideoAccelerationService
[0209]  {
[0210]      HRESULT GetPipelineConfigurations(
[0211]          [out] UINT32 * pCount,
[0212]          [out,unique,size_is( * pCount)]GUID * * pGuids
[0213]          );
[0214]      HRESULT GetFormats(
[0215]          [out] UINT32 * pCount,
[0216]          [out,unique,size_is( * pCount)]GUID * * pGuids
[0217]          );
[0218]      HRESULT GetDistanceMetrics(
[0219]          [out] UINT32 * pCount,
[0220]          [out,unique,size_is( * pCount)]GUID * * pGuids
[0221]          );
[0222]      HRESULT GetSearchProfiles(
```

[0223] [out] UINT32 * pCount,

[0224] [out, unique, size_is(* pCount)]VA2_Encode_SearchProfile * * pSearchProfiles

[0225]) ;

[0226] HRESULT GetMECapabilities(

[0227] [out]VA2_Encode_MECaps * pMECaps

[0228]) ;

[0229] HRESULT CreateVideoEncoder(

[0230] [in] REFGUID PipelineGuid,

[0231] [in] REFGUID FormatGuid,

[0232] [in] UINT32 NumStreams,

[0233] [in] VA2_Encode_StaticConfiguration * pConfiguration,

[0234] [in, size_is(NumStreams)] VA2_Encode_DataDescription * pDataDescription,

[0235] [in, size_is(NumStreams)] VA2_Encode_Allocator * SuggestedAllocatorProperties,

[0236] [out, size_is(NumStreams)] VA2_Encode_Allocator * pActualAllocatorProperties,

[0237] [out] IVideoEncoder * * ppEncode

[0238]) ;

[0239] } ;

[0240] 3.2 方法 :IVideoEncoder

[0241] 3.2.1 GetBuffer

[0242] 该函数返回在 Excute 调用中使用的缓冲区（编码表面）。该缓冲区在使用之后通过调用 ReleaseBuffer（释放缓冲区）来立即释放，以避免使流水线停止。

[0243] HRESULT GetBuffer(

[0244] [in] UINT8 StreamId,

[0245] [in] UINT32 StreamType,

[0246] [in] BOOL Blocking,

[0247] [out]PVOID pBuffer

[0248]) ;

[0249] #define E_NOTAVAILABLE HRESULT_FROM_WIN32(ERROR_INSUFFICIENT_BUFFER)

[0250] #define E_INVALIDPARAMETER HRESULT_FROM_WIN32(ERROR_INVALID_PARAMETER)

[0251] 参数

[0252] StreamId

[0253] 指的是对其需要缓冲区的特定流。取决于特定的流，将返回不同类型的缓冲区，如输入图像缓冲区、运动矢量缓冲区等等。并非对给定配置的所有流 ID 都是对此函数的有效

输入。对于 StreamId 的允许值作为流水线配置的一部分来指定。

[0254] Stream Type

[0255] 指定要返回的缓冲区的类型。通常，流类型由 StreamId 暗示，并且在创建时协商。如果 StreamType 与 StreamId 不一致，则该函数返回出错值。数据缓冲区如由备注一节中的表格所描述的基于 StreamType 的值来解释（强制类型转换）。

[0256] Blocking

[0257] 指定当存在“饥饿”时函数的行为，或用于扼流的某一其它需求。值 True（真）指示该函数应当阻塞，而 False（假）指示该函数应当返回

[0258] E_NOTAVAILABLE。

[0259] pBuffer

[0260] 指向要经由 ReleaseBuffer 释放的数据缓冲区的指针。该指针基于 StreamType 参数来重新强制转换（解释），并且在以下备注一节中的表格中描述。

[0261] 返回值

[0262] S_OK

[0263] 函数成功。

[0264] E_NOTAVAILABLE

[0265] 这是在驱动程序因缺乏缓冲区而饥饿，并且 Blocking 标志被设为假时返回的。

[0266] E_INVALIDPARAMETER

[0267] 输入参数不正确。这可例如在 StreamType 不匹配给定 StreamId 的期望值时使用。

[0268] VFW_E_UNSUPPORTED_STREAM

[0269] StreamId 无效。GetBuffer 不提供用于指定流 ID 的缓冲区。对 StreamId 的允许值作为流水线配置的一部分来描述。对于指定的流 ID，分配符可以是外部的，或者可以完全没有分配符。

[0270] E_FAIL

[0271] 函数失败。

[0272] 备注

[0273] 通常，该函数非常快地返回控制，因为缓冲区已经存在于分配符队列中。该函数应当阻塞（或返回 E_NOTAVAILABLE）的唯一条件是当来自分配符队列的所有缓冲区都被提交给设备，或被应用程序消耗，因此不被释放的时候。

[0274] 流类型和缓冲区格式

[0275]

StreamType_Video	IDirect3DSurface9**pBuffer
StreamType_MV	IDirect3DSurface9**pBuffer
StreamType_Residues	IDirect3DSurface9*pBuffer[3]。即，pBuffer 是指向三个 D3D 表面指针的指针。 pBuffer[0] 是指向亮度表面的指针。

StreamType_Video	IDirect3DSurface9**pBuffer
	pBuffer[1] 是指向 Cb 表面的指针，或者如果不适用则为空。 pBuffer[2] 是指向 Cr 表面的指针，或者如果不适用则为空。

[0276] 3.2.2 ReleaseBuffer

[0277] 该函数用于将表面释放回分配符队列以便经由 GetBuffer 重复使用。

[0278] HRESULT ReleaseBuffer(

[0279] [in] UINT8 StreamId,

[0280] [in] UINT8 StreamType,

[0281] [in] PVOID pBuffer

[0282]);

[0283] 参数

[0284] StreamId

[0285] 与缓冲区相关联的流 ID。

[0286] Stream Type

[0287] 用于缓冲区的流类型。

[0288] pBuffer

[0289] 要释放回分配符队列的缓冲区。

[0290] 返回值

[0291] S_OK

[0292] 函数成功。

[0293] E_FAIL

[0294] 函数失败。

[0295] 3.2.3 Execute

[0296] 该函数用于向硬件提交请求。它包含经由 GetBuffer 获得的输入和输出数据缓冲区以及某些配置信息。该函数是异步的，并且其完成由用信号表示的事件来指示。完成状态使用 pStatus 参数来指示，该参数在堆上分配，并且仅在用信号表示事件之后才检查。

[0297] 作为参数提供给此函数的缓冲区在该函数真正完成之前不被应用程序读取（例如，经由 LockRect）或写入。真正的完成是由函数返回出错值来暗示的，或者如果该函数返回成功，则通过用信号表示 hEvent（该函数的参数）来暗示。当将同一缓冲区输入到 Execute 调用的几个实例时，在所有相关联的 Execute 调用完成之前不访问该缓冲区。指向 Execute 使用的表面的指针仍作为参数提供给与 Execute 类似的 VA 函数，因为这不需要锁定数据。该最后一条规则解释了如何可同时在多个 Execute 调用中使用同一输入图像。

[0298] 提供给该调用的缓冲区遵循在创建时协商的分配符语义。如果在期望使用 GetBuffer 时使用了外部分配符，则该函数将返回 E_FAIL。

[0299] HRESULT Execute(

[0300] [in] UINT8 Stage,

[0301] [in] UINT32 NumInputDataParameters,

[0302] [in,size_is(NumInputDataParameters)]

[0303] VA2_Encode_ExecuteDataParameter * * pInputData,

[0304] [in] UINT32 NumOutputDataParameters,

[0305] [out,size_is(NumOutputDataParameters)]

[0306] VA2_Encode_ExecuteDataParameter * * pOutputData,

[0307] [in] UINT32 NumConfigurationParameters,

[0308] [in,size_is(NumConfigurationParameters]

[0309] VA2_Encode_ExecuteConfigurationParameter * * pConfiguration,

[0310] [in]HANDLE hEvent,

[0311] [out]HRESULT * pStatus

[0312]);

[0313] 参数

[0314] Stage

[0315] 对于拆分的流水线配置,该参数标识拆分的流水线的特定阶段。编号是基于 1 的,并且对于非拆分的流水线该参数被忽略。

[0316] NumInputDataParameters

[0317] 输入数据数组(下一参数)的大小。

[0318] pInputData

[0319] 指向输入数据值的指针数组。个别数据指针基于 StreamId 值来适当地重新强制转换,该值具有在创建时指定的相关联的 StreamDescription(流描述)。数据缓冲区是在创建时分配的,并且在流传送过程期间通过调用 GetBuffer 来获得。

[0320] NumOutputDataParameters

[0321] 输出数据数组(下一参数)的大小。

[0322] pOutputData

[0323] 指向输出数据值的指针数组。个别数据指针基于 StreamId 值来适当地重新强制转换,该值具有在创建时指定的相关联的 StreamDescription。

[0324] 数据缓冲区是在创建时分配的,并且在流传送过程期间通过调用 GetBuffer 来获得。

[0325] NumConfigurationParameters

[0326] 配置数组(下一参数)的大小

[0327] pConfiguration

[0328] 控制流水线的执行的配置参数的数组。总体配置是该结构以及在创建编码器时提供的静态配置参数的并集。

[0329] hEvent

[0330] 用信号标识输出数据就绪的事件句柄。

[0331] pStatus

[0332] 指示所请求的操作是否成功完成的状态。允许的值包括 S_OK(成功完成)、E_

TIMEOUT(如果超过了 TimeLimit(时限))以及 E_SCENECHANGE(如果启用场景改变检测并检测到该改变)。在两种出错的情况下,没有一个输出表面包含有用数据。该参数在堆上分配,并且返回值仅在用信号表示了 hEvent 之后才检查。

[0333] 返回值

[0334] S_OK

[0335] 函数成功。

[0336] E_FAIL

[0337] 函数失败。

[0338] 备注

[0339] 如果用信号表示了事件句柄,则意味着当 LockRect 在任一输出表面上调用时其应立即完成,因为它们已经就绪。特别地,期望 LockRect 调用通过在任何事件句柄上等待而不会在任何时间长度上锁定。也不允许通过忙碌的自旋来浪费 CPU 时间。

[0340] 3.3 数据结构 :Execute

[0341] Execute 调用具有数据参数和配置参数。具体的数据参数可以被认为是从 VA2_Encode_ExecuteDataParameter 基类(或结构)导出的,而具体的配置参数可以被认为是从 VA2_Encode_ExecuteConfigurationParameter 基类(或结构)导出的。

[0342] 3.1.1 VA2_Encode_ExecuteDataParameter

[0343] typedef struct_VA2_Encode_ExecuteDataParameter{

[0344] UINT32 Length ;

[0345] UINT32 StreamId ;

[0346] } VA2_Encode_ExecuteDataParameter ;

[0347] 成员

[0348] Length

[0349] 该结构中的字节数。为可扩展性提供。

[0350] StreamId

[0351] 在流水线配置中定义的数据流的 ID。缓冲区格式在创建时使用 StreamDescription(流描述)参数来协商。

[0352] 3.3.2 VA2_Encode_ExecuteConfigurationParameter

[0353] typedef struct_VA2_Encode_ExecuteConfigurationParameter{

[0354] UINT32 Length ;

[0355] UINT32 StreamId ;

[0356] UINT32 ConfigurationType ;

[0357] } VA2_Encode_ExecuteConfigurationParameter ;

[0358] #define VA2_Encode_ConfigurationType_MotionEstimation 0x1

[0359] #define VA2_Encode_ConfigurationType_Quantization 0x2

[0360] 成员

[0361] Length

[0362] 该结构中的字节数。为可扩展性提供。

[0363] StreamId

[0364] 在流水线配置中定义的数据流的 ID。这可用于推断数据是输入还是输出。

[0365] ConfigurationType

[0366] 该参数描述了配置参数，并用于适当地对当前结构进行类型强制转换。

[0367] 备注

[0368] 该结构用作更专门的配置信息的基础类型。该基础类型基于 ConfigurationType（配置类型）参数进行类型强制转换为更专门的类型。

[0369] ConfigurationType 和专门的结构之间的映射在下表中描述。

[0370] 配置类型

[0371]

ConfigurationType_MotionEstimation	ConfigurationParameter_MotionEstimation
ConfigurationType_Quantization	ConfigurationParameter_Quantization

[0372] 3.3.3 DataParameter_MotionVectors

[0373] typedef struct_VA2_Encode_ExecuteDataParameter_MotionVectors{

[0374] UINT32 Length；

[0375] UINT32 StreamId；

[0376] VA2_Encode_MVSurface * pMVSurface；

[0377] }VA2_Encode_ExecuteDataParameter_MotionVectors；

[0378] 成员

[0379] Length

[0380] 该结构中的字节数。为可扩展性提供。

[0381] StreamId

[0382] 在流水线配置中定义的数据流的 ID。这可用于推断数据是输入还是输出。

[0383] pMVSurface

[0384] 指向包含运动矢量 D3D 表面的结构。

[0385] 3.3.4 DataParameter_Residues

[0386] typedef struct_VA2_Encode_ExecuteDataParameter_Residues{

[0387] UINT32 Length；

[0388] UINT32 StreamId；

[0389] VA2_Encode_ResidueSurface * pResidueSurfaceY；

[0390] VA2_Encode_ResidueSurface * pResidueSurfaceCb；

[0391] VA2_Encode_ResidueSurface * pResidueSurfaceCr；

[0392] } VA2_Encode_ExecuteDataParameter_Residues；

[0393] 成员

[0394] Length

[0395] 该结构中的字节数。为可扩展性提供。

[0396] StreamId

[0397] 在流水线配置中定义的数据流的 ID。这可用于推断数据是输入还是输出。

[0398] pResidueSurfaceY

[0399] 包含亮度值的残差表面。

[0400] pResidueSurfaceCb

[0401] 包含色度 Cb 值的残差表面。

[0402] pResidueSurfaceCr

[0403] 包含色度 Cr 值的残差表面。

[0404] 3.3.5 DataParameter_InputImage

[0405] typedef struct_VA2_Encode_ExecuteDataParameter_InputImage{

[0406] UINT32 Length ;

[0407] UINT32 StreamId ;

[0408] VA2_Encode_ImageInfo * pImageData ;

[0409] } VA2_Encode_ExecuteDataParameter_InputImage ;

[0410] 成员

[0411] Length

[0412] 该结构中的字节数。为可扩展性提供。

[0413] StreamId

[0414] 在流水线配置中定义的数据流的 ID。这可用于推断数据是输入还是输出。

[0415] pImageData

[0416] 指向包含输入图像 D3D 表面的结构的指针。这是对其需要运动矢量的表面。

[0417] 3.3.6 DataParameter_ReferenceImages

[0418] typedef struct_VA2_Encode_ExecuteDataParameter_ReferenceImages{

[0419] UINT32 Length ;

[0420] UINT32 StreamId ;

[0421] UINT32 NumReferenceImages ;

[0422] VA2_Encode_ImageInfo * pReferenceImages

[0423] } VA2_Encode_ExecuteDataParameter_ReferenceImages ;

[0424] 成员

[0425] Length

[0426] 该结构中的字节数。为可扩展性提供。

[0427] StreamId

[0428] 在流水线配置中定义的数据流的 ID。这可用于推断数据是输入还是输出。

[0429] DataType

[0430] NumReferenceImages

[0431] 参考图像数组（下一参数）的大小

[0432] pReferenceImages

[0433] 作为运动矢量的基础的参考图像的数组。对于如 MPEG-2 的简单格式，可以仅使用一个逐行帧（或两个半帧）。另一方面，如 H.264 和 VC-1 等格式支持跨越几个帧的运动矢量。MPEG-2 中的 P 帧仅使用一个参考图像，而 B 帧具有隔行的视频，并且半帧类型的运动可

能使用 4 个图像，其中每一图像可以是一帧或一个半帧。

[0434] 3.3.7 DataParameter_DecodedImage

[0435] typedef struct_VA2_Encode ExecuteDataParameter_DecodedImage{

[0436] UINT32 Length ;

[0437] UINT32 StreamId ;

[0438] VA2_Encode_ImageInfo * pYCbCrImage ;

[0439] } VA2_Encode_ExecuteDataParameter_DecodedImage ;

[0440] 成员

[0441] Length

[0442] 该结构中的字节数。为可扩展性提供。

[0443] StreamId

[0444] 在流水线配置中定义的数据流的 ID。这可用于推断数据是输入还是输出。

[0445] DataType

[0446] pYCbCrImage

[0447] 在反量化、反变换和运动补偿之后获得的输出的已解码图像。对于良好的流水线，相关联的 D3D 表面不应被锁定，也不必将数据传送到系统存储器。表面指针仍可用作参考图像。

[0448] 3.3.8 VA2_Encode_ImageInfo

[0449] typedef struct_VA2_Encode_ImageInfo{

[0450] IDirect3DSurface9 * pSurface ;

[0451] BOOL Field ;

[0452] BOOL Interlaced ;

[0453] RECT Window ;

[0454] } VA2_Encode_ImageInfo ;

[0455] 成员

[0456] pSurface

[0457] 指向包含 YCbCr 格式的图像的 D3D 表面的指针。

[0458] Field

[0459] 值为 1 指示该表面包含视频数据的一个半帧，并且该数据假定为隔行的。0 指示完整的逐行帧。

[0460] Interlaced

[0461] 值为 1 指示该图像数据是隔行的。该标志应仅在 Field（上一个参数）被设为 1 时才使用。如果 Field 被设为 1，则数据被假定为是隔行的。

[0462] Window

[0463] 图像内的矩形。这可用于限制运动估计调用仅返回用于整个图像内的一个矩形的运动矢量。

[0464] 3.3.9 ConfigurationParameter_MotionEstimation

[0465] typedef struct_VA2_Encode_ExecuteConfigurationParameter_MotionEstimation{

[0466] UINT32 Length ;

[0467] UINT32 StreamId ;

[0468] UINT32 ConfigurationType ;

[0469] VA2_Encode_MEParameters * pMEParams ;

[0470] } VA2_Encode_ExecuteConfigurationParameter_MotionEstimation ;

[0471] 成员

[0472] Length

[0473] 该结构中的字节数。为可扩展性提供。

[0474] StreamId

[0475] 在流水线配置中定义的数据流的 ID。这可用于推断数据是输入还是输出。

[0476] ConfigurationType

[0477] pMEParams

[0478] 指向定义支配包括搜索窗的运动搜索等的各种参数的结构的指针。

[0479] 备注

[0480] 图 8 示出了根据一个实施例的几个示例性运动估计参数。这些参数在以下结构中使用。

[0481] 3.3.10 VA2_Encode_SearchResolution

[0482] typedef enum(

[0483] VA2_Encode_SearchResolution_FullPixel,

[0484] VA2_Encode_SearchResolution_HalfPixel,

[0485] VA2_Encode_SearchResolution_QuarterPixel

[0486] } VA2_Encode_SearchResolution ;

[0487] 描述

[0488] FullPixel

[0489] 运动矢量是以完整的像素为单位来计算的。

[0490] HalfPixel

[0491] 运动矢量是以半像素为单位来计算的。因此运动矢量值 (5,5) 表示离开 (2.5, 2.5) 像素的数据宏块。

[0492] QuarterPixel

[0493] 运动矢量是以四分之一像素为单位来计算的。因此运动矢量值 (10,10) 表示离开 (2.5,2.5) 像素的数据宏块。

[0494] 在计算子像素运动矢量值时,编码器使用内插来估计亮度和色度值。具体的内插方案是格式相关的,并且以下 GUID(静态配置的一部分) 控制内插方案。

[0495] //{E9AF78CB-7A8A-4d62-887F-B6A418364C79}

[0496] cpp_quote(" DEFINE_GUID(VA2_Encode_Interpolation_MPEG2Bilinear, 0xe9af78cb,0x7a8a,0x4d62,

[0497] 0x88,0x7f,0xb6,0xa4,0x18,0x36,0x4c,0x79) ; ")

[0498] //{A94BBFCB-1BF1-475c-92DE-67298AF56BB0}

[0499] cpp_quote(" DEFINE_GUID(VA2_Encode_Interpolation_MPEG2Bicubic,

0xa94bbfcb,0x1bf1,0x475c,0x92,

[0500] 0xde,0x67,0x29,0x8a,0xf5,0x6b,0xb0) ; ")

[0501] 3.3.11 VA2_Encode_SearchProfile

[0502] typedef struct_VA2_Encode_SearchProfile{

[0503] UINT8 QualityLevel ;

[0504] UINT8 TimeTaken ;

[0505] GUID SearchTechnique ;

[0506] GUID SubpixelInterpolation ;

[0507] } VA2_Encode_SearchProfile ;

[0508] 成员

[0509] QualityLevel

[0510] 范围在 [0-100] 的数字,指示运动矢量在设备支持的不同简档中的相对质量。

[0511] TimeTaken

[0512] 范围在 [0-100] 的数字,指示对不同搜索简档花费的相对时间。这使得应用程序能够作出合理的时间 - 质量折衷。

[0513] SearchTechnique

[0514] 指示所使用的搜索算法的 GUID。

[0515] SubpixelInterpolation

[0516] 指示所使用的子像素内插方案的 GUID。

[0517] 备注

[0518] 没有统一接受的绝对质量定义,因此同意接受相对度量。针对 TimeTaken 指示的值应当遵循严格的比例规则。如果简档 1 花费 10ms 而简档 2 花费 20ms,则 TimeTaken 值的比例应为 20/10 = 2。

[0519] 3.3.12 VA2_Encode_MBDescription

[0520] typedef struct_VA2_Encode_MBDescription{

[0521] BOOL ConstantMBSize ;

[0522] UINT32 MBWidth ;

[0523] UINT32 MBHeight ;

[0524] UINT32 MBCount ;

[0525] RECT * pMBRectangles ;

[0526] }VA2_Encode_MBDescription ;

[0527] 成员

[0528] ConstantMBSize

[0529] 值为 1 指示当前图像中的所有宏块具有相同的大小。这对于如 H.264 等格式并不如此。

[0530] MBWidth

[0531] 宏块的宽度。仅当 bConstantMBSize 为 1 时才有效。

[0532] MBHeight

[0533] 宏块的高度。仅当 bConstantMBSize 为 1 时才有效。

[0534] MRCount

[0535] 如果 bConstantMBSize 为 0,则使用矩形数组来描述图像中的宏块(或段)。该参数用以下 pMBRectangles 参数的元素来描述大小。

[0536] pMBRectangles

[0537] 描述如何切割图像的矩形数组。

[0538] 3.3.13 VA2_Encode_SearchBounds

[0539] typedef struct_VA2_Encode_SearchBounds{

[0540] BOOL DetectSceneChange;

[0541] UINT32 MaxDistanceInMetric;

[0542] UINT32 TimeLimit;

[0543] UINT32 MaxSearchWindowX;

[0544] UINT32 MaxSearchWindowY;

[0545] } VA2_Encode_SearchBounds;

[0546] 成员

[0547] DetectSceneChange

[0548] 如果该值为 1,则请求场景改变检测。在这一情况下,如果检测到场景改变,则 Execute 调用不计算运动矢量,并且因此不计算残差或已解码图像。这是经由 Execute 调用的 pStatus 参数来指示的,在此情况下该参数应被设为 E_SCENECHANGE。

[0549] MaxDistanceInMetric

[0550] 指的是当使用当前选择的距离度量来进行比较时宏块之间的差别。如果该距离超过 MaxDistanceInMetric 值,则拒绝这一运动矢量。

[0551] TimeLimit

[0552] 允许硬件花费在运动估计阶段上的最大时间。如果花费的时间要长于该时间,则 Execute 调用的 pStatus 参数被设为 E_TIMEOUT。

[0553] Search WindowX

[0554] 返回的运动矢量的 x 分量的最大值。换言之,搜索窗的大小(沿 x 维度)。

[0555] Search Window Y

[0556] 运动矢量的 y 分量的最大值。换言之,搜索窗的大小(沿 y 维度)。

[0557] 备注

[0558] 3.3.14 VA2_Encode_ModeType

[0559] typedef struct_VA2_Encode_ModeType{

[0560] UINT32 SearchProfileIndex;

[0561] GUID DistanceMetric;

[0562] INT16 HintX;

[0563] INT16 HintY;

[0564] } VA2_Encode_ModeType;

[0565] 成员

[0566] SearchProfileIndex

[0567] 由 GetSearchProfiles API 调用返回的搜索简档列表的索引。

[0568] DistanceMetric

[0569] 当比较两个宏块时使用的度量。常用的度量包括 SAD(绝对距离和)和 SSE(均方误差和)。

[0570] HintX

[0571] 关于要引导运动搜索的期望运动方向的提示。这指的是图像中的整体运动,并且不在每一 MB 的基础上适用。

[0572] HintY

[0573] 关于要引导运动搜索的期望运动方向的提示。这指的是图像中的整体运动,并且不在每一 MB 的基础上适用。

[0574] 3.3.15 ConfigurationParameter_Quantization

[0575] typedef struct_VA2_Encode_ExecuteConfigurationParameter_Quantization{

[0576] UINT32 Length;

[0577] UINT32 StreamId;

[0578] UINT32 ConfigurationType;

[0579] UINT32 Stepsize;

[0580] } VA2_Encode_ExecuteConfigurationParameter_Quantization;

[0581] 成员

[0582] Length

[0583] 该结构中的字节数。为可扩展性提供。

[0584] StreamId

[0585] 在流水线配置中定义的数据流的 ID。这可用于推断数据是输入还是输出。

[0586] Configuration Type

[0587] StepSize

[0588] 当执行量化时要使用的步长。该设计允许对在此调用中请求了运动矢量和残差的图像的整个部分仅使用一个步长。

[0589] 3.4 方法:IVideoEncoderService

[0590] 该接口中的方法允许应用程序向硬件查询其能力并用给定配置创建编码器对象

[0591] 3.4.1 GetPipelineConfigurations

[0592] HRESULT GetPipelineConfigurations(

[0593] [out] UINT32 * pCount,

[0594] [out,unique,size_is(* pCount)]GUID * * pGuids

[0595]);

[0596] 参数

[0597] pCount

[0598] 返回值描述了由该函数返回的 pGuids 数组(下一参数)的大小。

[0599] pGuids

[0600] 描述设备支持的各种流水线配置的 GUID 数组。存储器由被调用者分配,并且应当由调用者使用 CoTaskMemFree 来释放。

[0601] 返回值

[0602] S_OK

[0603] 函数成功。

[0604] E_OUTOFMEMORY

[0605] 函数无法分配存储器来返回 GUID 列表

[0606] E_FAIL

[0607] 由于某一设备出错无法确定所支持的流水线配置。

[0608] 3.4.2 GetFormats

[0609] HRESULT GetFormats(

[0610] [out] UINT32 * pCount,

[0611] [out,unique,size_is(* pCount)]GUID * * pGuids

[0612]);

[0613] 参数

[0614] pCount

[0615] 返回值描述了由该函数返回的 pGuids 数组（下一参数）的大小。

[0616] pGuids

[0617] 描述设备支持的各种格式（例如，WMV9、MPEG-2 等）的 GUID 数组。

[0618] 存储器由被调用者分配，并且应当由调用者使用 CoTaskMemFree 来释放。

[0619] 3.4.3 GetDistanceMetrics

[0620] HRESULT GetDistanceMetrics(

[0621] [out] UINT32 * pCount,

[0622] [out,unique,size_is(* pCount)]GUID * * pGuids

[0623]);

[0624] 参数

[0625] pCount

[0626] 返回值描述了由该函数返回的 pGuids 数组（下一参数）的大小。

[0627] pGuids

[0628] 描述设备支持的用于运动估计的各种搜索度量的 GUID 数组。存储器由被调用者分配，并且应当由调用者使用 CoTaskMemFree 来释放。

[0629] 返回值

[0630] S_OK

[0631] 函数成功。

[0632] E_OUTOFMEMORY

[0633] 函数无法分配存储器来返回 GUID 列表

[0634] E_FAIL

[0635] 由于某一设备出错无法确定所支持的度量。

[0636] 3.4.4 GetSearchProfiles

[0637] HRESULT GetSearchProfiles(

[0638] [out] UINT32 * pCount,

[0639] [out,unique,size_is(* pCount)]VA2_Encode_SearchProfile * *

pSearchProfiles

[0640]);

[0641] 参数

[0642] pCount

[0643] 返回值描述了该函数返回的 pGuids 数组（下一参数）的大小。

[0644] pSerachProfiles

[0645] 表示设备支持的搜索简档的 GUID 数组。该搜索简档允许编解码器应用程序更高效地进行时间 - 质量折衷。存储器由被调用者分配，并且由调用者使用 CoTaskMemFree 来释放。

[0646] 返回值

[0647] S_OK

[0648] 函数成功。

[0649] E_OUTOFMEMORY

[0650] 函数无法分配存储器来返回 GUID 列表

[0651] E_FAIL

[0652] 由于某一设备出错无法确定所支持的搜索简档。

[0653] 3.4.5 GetMECapabilities

[0654] HRESULT GetMECapabilities(

[0655] [out] VA2_Encode_MECaps * pMECaps

[0656]);

[0657] 参数

[0658] pMECaps

[0659] 指向设备的运动估计能力的指针。这包括关于设备能够处理的图像的大小、最大搜索窗大小以及设备是否支持可变宏块大小的信息。用于该参数的存储器由调用者分配。

[0660] 返回值

[0661] S_OK

[0662] 函数成功。

[0663] E_FAIL

[0664] 由于某一设备出错函数失败。

[0665] 3.4.6 CreateVideoEncoder

[0666] 该函数创建 IVideoEncoder 的实例。

[0667] HRESULT CreateVideoEncoder(

[0668] [in] REFGUID PipelineGuid,

[0669] [in] REFGUID FormatGuid,

[0670] [in] UINT32 NumStreams,

[0671] [in] VA2_Encode_StaticConfiguration * pConfiguration,

[0672] [in, size_is(NumStreams)] VA2_Encode_StreamDescription * pStreamDescription,

[0673] [in, size_is(NumStreams)] VA2_Encode_Allocator *

SuggestedAllocatorProperties,

[0674] [out, size_is(NumStreams)]VA2_Encode_Allocator * pActualAllocatorProperties,

[0675] [out] IVideoEncoder * * ppEncode

[0676]);

[0677] 参数

[0678] PipelineGuid

[0679] 表示所需的流水线配置的GUID。配置列表经由GetCapabilities获得,并且每一GUID与描述关于该配置的必要细节的公共文档相关联。

[0680] FormatGuid

[0681] 表示最终编码的比特流的格式的GUID。如变换和量化等许多编码操作对其具有格式专用的元素。尽管这些格式专用元素可以由具有足够速度的CPU来处理,但是信息交换将必须使用额外的流水线阶段并且使得更难实现高流水线效率。

[0682] NumStreams

[0683] 与流水线配置相关联的输入和输出数据流的数量。这在许多情况下由流水线GUID来暗示。

[0684] pConfiguration

[0685] 指向静态配置特性的指针。

[0686] pStreamDescription

[0687] 描述流经该流的数据的结构数组,每一个流一个。

[0688] SuggestedAllocatorProperties

[0689] 调用者(编解码器应用程序)基于其流水线设计建议要与不同的流相关联的特定数量的缓冲区(表面)。

[0690] pActualAllocatorProperties

[0691] 驱动程序基于其能够收集的资源和其它考虑事项指定实际分配符队列大小。假设是如果应用程序不能用可用的缓冲(分配符队列大小)来构建高效的流水线则其将放弃对该接口的使用。

[0692] ppEncode

[0693] 输出编码器对象。调用者应当将此认为是要经由IUnknown::Release释放的常规COM对象。

[0694] 返回值

[0695] S_OK

[0696] 函数成功。

[0697] E_FAIL

[0698] 函数失败(可能是缺少资源)。

[0699] 3.5 数据结构:IVideoEncoderService

[0700] 3.5.1 VA2_Encode_MECaps

[0701] typedef struct_VA2_Encode_MECaps{

[0702] BOOL VariableMBSizeSupported;

[0703] BOOL MotionVectorHintSupported ;

[0704] UINT16 MaxSearchWindowX ;

[0705] UINT16 MaxSearchWindowY

[0706] UINT32 MaxImageWidth ;

[0707] UINT32 MaxImageHeight ;

[0708] UINT32 MaxMBSizeX ;

[0709] UINT32 MaxMBSizeY ;

[0710] } VA2_Encode_MECaps ;

[0711] 成员

[0712] VariableMBSizeSupported

[0713] 值为 1 指示当执行运动估计时硬件支持可变宏块大小。特别地，如果支持可变宏块大小，则此 API 的调用者在 VA2_Encode_MBDescription 结构中将 ConstantMBSize 设为 0 并使用 pMBRectangles 参数来描述对图像的划分是合法的。

[0714] MotionVectorHintSupported

[0715] 值为 1 指示硬件能够在其运动搜索算法中使用来自调用者的某些提示。

[0716] 特别地，调用者可设置作为 Execute 配置参数的 VA2_Encode_ModeType 的 HintX 和 HintY 成员。

[0717] MaxSearchWindowX

[0718] 作为 VA2_Encode_SearchBounds 的成员的 SearchWindowX 的最大合法值，VA2_Encode_SearchBounds 是运动估计配置参数。

[0719] MaxSearchWindowY

[0720] 作为 VA2_Encode_SearchBounds 的成员的 SearchWindowY 的最大合法值，VA2_Encode_SearchBounds 是运动估计配置参数。

[0721] MaxImage Width

[0722] 输入图像的最大允许宽度。

[0723] MaxImageHeight

[0724] 输入图像的最大允许高度。

[0725] MaxMBSizeX

[0726] 宏块的最大允许宽度。

[0727] MaxMBSizeY

[0728] 宏块的最大允许高度。

[0729] 3.5.2 VA2_Encode_StaticConfiguration

[0730] typedef struct_VA2_Encode_StaticConfiguration{

[0731] GUID TransformOperator ;

[0732] GUID PixelInterpolation ;

[0733] GUID Quantization ;

[0734] UINT8 NumMotionVectorsPerMB ;

[0735] VA2_Encode_MVLayout MVLayout ;

[0736] VA2_Encode_ResidueLayout ResLayout ;

[0737] } VA2_Encode_StaticConfiguration ;

[0738] 成员

[0739] Transform Operator

[0740] 表示 Transform 算子 -MPEG-2 DCT、WMV9 变换等之一 - 的 GUID。

[0741] PixelInterpolation

[0742] 表示要使用的子像素内插的GUID。双线性和双三次内插系统具有格式专用的多个系数。

[0743] Quantization

[0744] 表示要使用的量化方案的 GUID。

[0745] NumMotion VectorsPerMB

[0746] 每一宏块要计算的运动矢量的数量。该接口的较早的版本所支持的简单流水线配置可能要求该值为 1。

[0747] MVLayout

[0748] 运动矢量表面的布局。

[0749] ResidueLayout

[0750] 残差表面的布局。

[0751] 3.5.3 VA2_Encode_Allocator

[0752] typedef struct_VA2_Encode Allocator{

[0753] BOOL ExternalAllocator ;

[0754] UINT32 NumSurfaces ;

[0755] } VA2_Encode Allocator ;

[0756] 成员

[0757] ExternalAllocator

[0758] False 指示缓冲区是经由 GetBuffer 获得的，而 True 指示缓冲区是经由外部分配符获得的，或者没有与所讨论的流相关联的分配符。流水线配置在许多情况下强迫该字段的值（通常为 0）。一个值得注意的例外是在允许来自外部分配符的输入图像流中。

[0759] NumSurfaces

[0760] 要与分配符队列相关联的表面的数量。

[0761] 3.5.4 VA2_Encode_StreamDescription

[0762] typedef struct_VA2_Encode_StreamDescription{

[0763] UINT32 Length ;

[0764] UINT32 StreamType ;

[0765] } VA2_Encode_StreamDescription ;

[0766] 成员

[0767] Length

[0768] 用于确认类型强制转换并允许可扩展性的整个结构的长度。

[0769] Stream Type

[0770] 描述与该流相关联的数据的类型。用于类型强制转换。

[0771] 备注

[0772] 该基本结构被类型强制转换为 StreamType 字段上的派生类型。类型强制转换在关于 VA2_Encode_StreamType 的文档中有描述。

[0773] 3.5.5 VA2_Encode_StreamType

[0774] #define VA2_Encode_StreamType_Video 0x1

[0775] #define VA2_Encode_StreamType_MV 0x2

[0776] #define VA2_Encode_StreamType_Residues 0x3

[0777] 类型描述

[0778] VA2_Encode_StreamType_Video

[0779] 相关联的流描述结构可被强制转换为 VA2_Encode_StreamDescription_Video。

[0780] VA2_Encode_StreamType_MV

[0781] 相关联的流描述结构可被强制转换为 VA2_Encode_StreamDescription_MV。

[0782] VA2_Encode_StreamType_Residues

[0783] 相关联的流描述结构可被强制转换为 VA2_Encode_StreamDescription_Residues。

[0784] 3.5.6 VA2_Encode_StreamDescription_Video

[0785] typedef struct_VA2_Encode_StreamDescription{

[0786] UINT32 Length ;

[0787] UINT32 StreamType ;

[0788] VA2_VideoDesc VideoDescription ;

[0789] } VA2_Encode_StreamDescription ;

[0790] 成员

[0791] Length

[0792] 用于确认类型强制转换并允许可扩展性的整个结构的长度。

[0793] StreamType

[0794] 描述与该流相关联的数据的类型。用于类型强制转换。

[0795] VideoDescription

[0796] 描述视频流的各种特性,包括维数、帧速率、色原等等。

[0797] 3.5.7 VA2_Encode_StreamDescription_MV

[0798] typedef struct_VA2_Encode_StreamDescription{

[0799] UINT32 Length ;

[0800] UINT32 StreamType ;

[0801] VA2_Encode_MVType MVType ;

[0802] VA2_Encode_MVLayout MVLayout ;

[0803] } VA2_Encode_StreamDescription ;

[0804] 成员

[0805] Length

[0806] 用于确认类型强制转换并允许可扩展性的整个结构的长度。

[0807] StreamType

[0808] 描述与该流相关联的数据的类型。用于类型强制转换。

[0809] MYType

[0810] 描述用于返回运动数据的运动矢量结构的类型。用于解释运动矢量表面的内容。

[0811] MYLayout

[0812] 描述在存储器中如何布局用于给定输入图像的运动矢量结构。

[0813] 3.5.8 VA2_Encode_StreamDescriptoin_Residues

[0814] typedef struct_VA2_Encode_StreamDescription{

[0815] UINT32 Length ;

[0816] UINT32 StreamType ;

[0817] VA2_Encode_SamplingType SamplingType ;

[0818] UINT32 LumaWidth ;

[0819] UINT32 LumaHeight ;

[0820] UINT32 ChromaCbWidth ;

[0821] UINT32 ChromaCbHeight ;

[0822] UINT32 ChromaCrWidth ;

[0823] UINT32 ChromaCrHeight ;

[0824] } VA2_Encode_StreamDescription ;

[0825] 成员

[0826] Length

[0827] 用于确认类型强制转换并允许可扩展性的整个结构的长度。

[0828] Stream Type

[0829] 描述与该流相关联的数据的类型。用于类型强制转换。

[0830] SamplingType

[0831] 指定残差数据是否为 4:4:4、4:2:2 等等。

[0832] LumaWidth

[0833] 亮度表面的宽度。

[0834] LumaHeight

[0835] 亮度表面的高度。

[0836] ChromaCbWidth

[0837] 包含 Cb 残差值的表面的宽度。

[0838] ChromaCbHeight

[0839] 包含 Cb 残差值的表面的高度。

[0840] ChromaCrWidth

[0841] 包含 Cr 残差值的表面的宽度。

[0842] ChromaCrHeight

[0843] 包含 Cr 残差值的表面的高度。

[0844] 3.6 数据结构 :运动矢量

[0845] 3.6.1 运动矢量布局

[0846] 图 9 示出了根据一个实施例的储存在 D3D 结构中的示例性运动矢量数据。被描述为“MV”的每一单元是运动矢量结构。取决于 VA2_Encode_MVType 和 VA2_Encode_MVLayout

的值使用不同的表示。实际的结构和布局描述如下。

[0847] 3.6.2 新的 D3D 格式

[0848] typedef enum_D3DFORMAT

[0849] {

[0850] D3DFMT_MOTIONVECTOR16 = 105,

[0851] D3DFMT_MOTIONVECTOR32 = 106,

[0852] D3DFMT_RESIDUE16 = 107,

[0853] } D3DFORMAT;

[0854] 运动矢量表面和残差表面与以上新的 D3D 格式类型相关联,该类型指示了个别运动矢量和残差的大小。该大小信息由驱动程序在应用程序使用创建 API 提供的表面或资源之一来创建表面时使用。与编码表面相关联的资源标志是 VA2_EncodeBuffer。

[0855] //缓冲区类型

[0856] enum

[0857] {

[0858] VA2_EncodeBuffer = 7,

[0859] };

[0860] typedef struct_D3DDDI_RESOURCEFLAGS

[0861] {

[0862] union

[0863] {

[0864] struct

[0865] {

[0866] UINT TextApi :1;//0x20000000

[0867] UINT EncodeBuffer :1;//0x40000000

[0868] UINT Reserved :1;//0x80000000

[0869] };

[0870] UINT Value;

[0871] };

[0872] } D3DDDI_RESOURCEFLAGS;

[0873] 3.6.3 VA2_Encode_MVSurface

[0874] 该结构从 IDirect3DSurface9 中高效地派生,并且携带了允许解释嵌入的 D3D 结构的内容的状态信息。

[0875] typedef struct_VA2_Encode_MVSurface{

[0876] IDirect3DSurface9 * pMVSurface;

[0877] VA2_Encode_MVType MVType;

[0878] VA2_Encode_MVLayout MVLayout;

[0879] GUID DistanceMetric;

[0880] } VA2_Encode_MVSurface;

[0881] 成员

[0882] pMVSurface

[0883] 指向包含运动矢量的 D3D 表面的指针。

[0884] MYType

[0885] 该值用于标识要用于解释个别运动矢量的结构 (VA2_Encode_MotionVector8 等)。

[0886] MYLayout

[0887] 该值标识如何在 D3D 表面中布局个别运动矢量结构。

[0888] DistanceMetric

[0889] 表示用于测量两个宏块之间的距离的距离度量的GUID。距离度量用于标识最接近的宏块,且因此是最优运动矢量。

[0890] 3.6.4 VA2_Encode_MVType

[0891] 该枚举值用于解码运动矢量 D3D9 表面的内容。取决于运动矢量的类型,使用几个不同的运动矢量结构之一来解释该表面的内容。

[0892] typedef enum{

[0893] VA2_Encode_MVType_Simple8,

[0894] VA2_Encode_MVType_Simple16,

[0895] VA2_Encode_MVType_Extended8,

[0896] VA2_Encode_MVType_Extended16

[0897] } VA2_Encode_MVType ;

[0898] 描述

[0899] VA2_Encode_MVType_Simple8

[0900] 运动矢量结构是 VA2_Encode_MotionVector8。

[0901] VA2_Encode_MVType_Simple16

[0902] 运动矢量结构是 VA2_Encode_MotionVector16。

[0903] VA2_Encode_MVType_Extended8

[0904] 运动矢量结构是 VA2_Encode_MotionVectorEx8。

[0905] VA2_Encode_MVType_Extended16

[0906] 运动矢量结构是 VA2_Encode_MotionVectorEx16。

[0907] 3.6.5 VA2_Encode_MYLayout

[0908] typedef enum{

[0909] VA2_Encode_MVLayout_A,

[0910] VA2_Encode_MVLayout_B,

[0911] VA2_Encode_MVLayout_C

[0912] } VA2_Encode_MVLayout ;

[0913] 描述

[0914] 类型 A

[0915] 实际的 D3D 表面是由宏块索引和行索引来索引的运动矢量结构数组。

[0916] 类型 B

[0917] 这是其中每一宏块的运动矢量的数量不是常量的压缩布局。细节待定。

[0918] 类型 C

[0919] 3.6.6 VA2_Encode_MotionVector8

[0920] typedef struct_VA2_Encode_MotionVector8{

[0921] INT8 x ;

[0922] INT8 y ;

[0923] } VA2_Encode_MotionVector8 ;

[0924] 成员

[0925] x

[0926] 运动矢量的 x 坐标。

[0927] y

[0928] 运动矢量的 y 坐标。

[0929] 3.6.7 VA2_Encode_MotionVector16

[0930] typedef struct_VA2_Encode_MotionVector16{

[0931] INT16 x ;

[0932] INT16 y ;

[0933] } VA2_Encode_MotionVector16 ;

[0934] 成员

[0935] x

[0936] 运动矢量的 x 坐标。

[0937] y

[0938] 运动矢量的 y 坐标。

[0939] 3.6.8 VA2_Encode_MotionVectorEx8

[0940] typedef struct_VA2_Encode_MotionVectorEx8{

[0941] INT8 x ;

[0942] INT8 y ;

[0943] UINT8 ImageIndex ;

[0944] UINT8 Distance ;

[0945] } VA2_Encode_MotionVectorEx8 ;

[0946] 成员

[0947] x

[0948] 运动矢量的 x 坐标。

[0949] y

[0950] 运动矢量的 y 坐标。

[0951] ImageIndex

[0952] 对在对 ComputeMotionVectors 的调用中提供的参考图像列表的基于 0 的索引。

[0953] Distance

[0954] 测量单位由 VA_Encode_MVSurface 的 DistanceMetric 字段指定。它测量当前宏块与实际运动矢量 (x,y) 所涉及的参考宏块的距离。

[0955] 3.6.9 VA2_Encode_MotionVectorEx16

[0956] typedef struct_VA2_Encode_MotionVectorEx16{

[0957] INT16 x ;

[0958] INT16 y ;

[0959] UINT16 ImageIndex ;

[0960] UINT16 Distance ;

[0961] } VA2_Encode_MotionVectorEx16 ;

[0962] 成员

[0963] x

[0964] 运动矢量的 x 坐标。

[0965] y

[0966] 运动矢量的 y 坐标。

[0967] ImageIndex

[0968] 对在对 ComputeMotionVectors 的调用中提供的参考图像列表的基于 0 的索引。

[0969] Distance

[0970] 测量单位由 VA_Encode_MVSurface 的 DistanceMetric 字段指定。它测量当前宏块与实际运动矢量 (x, y) 所涉及的参考宏块的距离。

[0971] 3.7 数据结构 :残差

[0972] 残差表面是两字节长的有符号整数值的数组 - 换言之,其类型为 INT16。该方案看似在所有重要的情况下都是恰当的。例如,MPEG-2 处理 9 位残差值,而 H.264 处理 12 位残差。并且,如果原始数据是 YUV2,则每一亮度值占据一字节,且因此残差使用 9 位 (0-255 = -255)。此外,应用 DCT 类型的变换将数据要求增加到每一残差值 11 位。所有这些情况都通过使用 2 字节的长整型有符号残差值来适当地处理。

[0973] 残差表面的宽度是一行中的残差值的数量。例如,具有 4:2:2 的采样的 640x480 逐行图像每行具有 640 个亮度值和 320 个色度值。相关联的亮度表面的大小是 640x480x2,而色度表面的大小是 320x480x2 字节。

[0974] 残差表面是使用 D3DFMT_RESIDUE16 格式标志和 VA2_EncodeBuffer 资源类型来创建的。

[0975] 3.7.1 亮度平面

[0976] 图 10 示出了指示亮度表面的宽度匹配原始的 YCbCr 图像的示例性图示。例如,一个 640x480 的图像每行具有 480 个亮度值,因此亮度表面的宽度是 480。因此,亮度表面的大小是 640x480x2 字节。

[0977] 平面 = VA2_Encode_Residue_Y

[0978] 采样 = VA2_Encode_SamplingType_*

[0979] 3.7.2 色度 4:2:2

[0980] 图 11 示出了根据一个实施例的指示视频的每行的残差值的数量是原始视频图像的宽度的一半的示例性图示。因此,对于 640x480 的图像,每行的残差值的数量以及因此的表面宽度是 320。

[0981] 平面 = VA2_Encode_Residue_U 或 VA2_Encode_Residue_V

[0982] 采样 = VA2_Encode_SamplingType_422

[0983] 3.7.3 色度 4:2:0

[0984] 在此情形中，残差表面的宽度是原始逐行帧的宽度的一半，并且高度也是一半。因此，对于 640x480 的图像，色度表面本身将是 320 宽 240 长。

[0985] 平面＝ VA2_Encode_Residue_U 或 VA2_Encode_Residue_V

[0986] 采样＝ VA2_Encode_SamplingType_420

[0987] 4 示例性 DDI 文档

[0988] 扩展设备是由 VA 接口提供的通过 (pass-through) 机制，以添加除了现有的视频解码器和视频处理器功能之外的新功能。例如，它们将用于支持新的视频编码器功能。

[0989] 扩展设备类似于应用程序可用于向驱动程序发送数据 / 从驱动程序接收数据的非类型化漏斗来工作。数据的含义对于 VA 栈是未知的，并且由驱动程序基于 CreateExtensionDevice 调用的 pGuid 参数和 ExtensionExecute 的 Function 参数来解释。

[0990] VA 编码器使用以下 GUID 值（与 IVideoEncoder 的 uuid 相同）：

[0991] {7AC3D93D-41FC-4c6c-A1CB-A875E4F57CA4}

[0992] DEFINE_GUID(VA_Encoder_Extension,0x7ac3d93d,0x41fc,0x4c6c,0xa1,0xcb,0xa8,0x75,0xe4,0xf5,

[0993] 0x7c,0xa4)；

[0994] 4.1 枚举和能力

[0995] 扩展设备使用其类型参数被设为 GETEXTENSIONGUIDCOUNT 或 GETEXTENSIONGUIDS 的 FND3DDDI_GETCAPS 来枚举。编解码器应用程序在由 GETEXTENSIONGUIDS 返回的扩展 guid 列表中查找 VA_Encoder_Extension 以确定 VA 编码支持是否可用。

[0996] 4.1.1 FND3DDDI_GETCAPS

[0997] typedef HRESULT

[0998] (APIENTRY * PFND3DDDI_GETCAPS)

[0999] (

[1000] HANDLE hAdapter,

[1001] CONST D3DDDIARG_GETCAPS *

[1002])；

[1003] 当查询扩展设备（编码器设备）的能力时，使用具有以下结构的 GETEXTENSIONCAPS 作为 D3DDDIARG_GETCAPS 结构中的 pInfo。

[1004] 4.1.2 VADDI_QUERYEXTENSIONCAPSINPUT

[1005] typedef struct_VADDI_QUERYEXTENSIONCAPSINPUT

[1006] {

[1007] CONST GUID * pGuid；

[1008] UINT CapType；

[1009] VADDI_PRIVATEDATA * pPrivate；

[1010] } VADDI_QUERYEXTENSIONCAPSINPUT；

[1011] VADDI_QUERYEXTENSIONCAPSINPUT 的 pGuid 参数被设为 VA_Encoder_Extension。

[1012] #define VADDI_Encode_Captype_Guids VADDI_EXTENSION_CAPTYPE_MIN

[1013] #define VADDI_Encode_Captype_DistanceMetrics VADDI_EXTENSION_CAPTYPE_MIN+1

[1014] #define VADDI_Encode_Captype_SearchProfiles VADDI_EXTENSION_CAPTYPE_MIN+2

[1015] #define VADDI_Encode_Captype_MECaps VADDI_EXTENSION_CAPTYPE_MIN+3

[1016] GETEXTENSIONCAPS 的输出在 D3DDDIARG_GETCAPS 的 pData 参数中封装。pData 参数如下解释：

[1017] • Captype_Guids :Type = (GUID*).DataSize = sizeof(GUID)*

[1018] guid_count

[1019] • Captype_DistanceMetrics :Type = (GUID*).DataSize = sizeof(GUID)*

[1020] guid_count.

[1021] • Captype_SearchProfiles :Type = (VADDI_Encode_SearchProfile*).

[1022] DataSize = sizeof(VADDI_Encode_SearchProfile).

[1023] • Captype_MECaps :Type = (VADDI_Encode_MECaps).DataSize =

[1024] sizeof(VADDI_Encode_MECaps).

[1025] 4.1.3 D3DDDIARG_CREATEEXTENSIONDEVICE

[1026] 实际创建经由 D3DDDI_CREATEEXTENSIONDEVICE 调用发生，其主自变量如下示出：

[1027] typedef struct_D3DDDIARG_CREATEEXTENSIONDEVICE

[1028] {

[1029] CONST GUID * pGuid ;

[1030] VADDI_PRIVATEDATA * pPrivate ;

[1031] HANDLE hExtension ;

[1032] } D3DDDIARG_CREATEEXTENSIONDEVICE ;

[1033] 4.2 编码功能

[1034] 实际扩展单元功能经由 D3DDDI_EXTENSIONEXECUTE 调用来调用。扩展单元的实例已经与一 GUID 相关联，因此扩展单元的类型在作出执行调用时已经是已知的。唯一的附加参数是指示要执行的特定操作的 Function。例如，类型为 Encoder（编码器）的扩展设备可支持 MotionEstimation（运动估计）作为其功能之一。通常，扩展设备具有其自己的枚举扩展设备的能力的 GetCaps 函数。

[1035] typedef struct_D3DDDIARG_EXTENSIONEXECUTE

[1036] {

[1037] HANDLE hExtension ;

[1038] UINT Function ;

[1039] VADDI_PRIVATEDATA * pPrivateInput ;

[1040] VADDI_PRIVATEDATA * pPrivateOutput ;

[1041] UINT NumBuffers ;

[1042] VADDI_PRIVATEBUFFER * pBuffers ;

[1043] } D3DDDIARG_EXTENSIONEXECUTE ;

[1044] pBuffers 参数不被 VA 编码器使用，并且应被认为是一保留参数。Function 参数对于 VA 编码器取以下值：

[1045] #define VADDI_Encode_Function_Execute 1

[1046] D3DDDIARG_EXTENSIONEXECUTE 的 pPrivateInput 和 pPrivateOutput 参数用于封装 Execute API 调用的参数。嵌入在以下输入和输出参数中的编码专用参数尚未从 API 领域映射到 DDI 领域 - 但这只是重命名的事情，并且有可能能够应付单个定义。

[1047] 4.2.1 VADDI_Encode_Function_Execute_Input

[1048] 该参数包含对 Execute API 调用的输入参数。

[1049] typedef struct_VADDI_Encode_Function_Execute_Input

[1050] {

[1051] UINT32 NumDataParameters；

[1052] VA2_Encode_ExecuteDataParameter * * pData；

[1053] UINT32 NumConfigurationParameters；

[1054] VA2_Encode_ExecuteconfigurationParameter * * pConfiguration；

[1055] } VADDI_Encode_Function_Execute_Input；

[1056] 4.2.2 VADDI_Encode_Function_Execute_Output

[1057] 该结构封装了来自 Execute 调用的输出数据。

[1058] typedef struct_VADDI_Encode_Function_Execute_Output

[1059] {

[1060] UINT32 NumDataParameters；

[1061] VA2_Encode_ExecuteDataParameter * * pData；

[1062] } VADDI_Encode_Function_Execute_Output；

[1063] 4.3 扩展设备结构

[1064] 下节描述了与 VA 扩展机制相关联的各种结构和函数回调。

[1065] 4.3.1 VADDI_PRIVATEBUFFER

[1066] typedef struct_VADDI_PRIVATEBUFFER

[1067] {

[1068] HANDLE hResource；

[1069] UINT SubResourceIndex；

[1070] UINT DataOffset；

[1071] UINT DataSize；

[1072] } VADDI_PRIVATEBUFFER；

[1073] typedef struct_VADDI_PRIVATEDATA

[1074] {

[1075] VOID * pData；

[1076] UINT DataSize；

[1077] } VADDI_PRIVATEDATA；

[1078] 4.3.2 D3DDDIARG_EXTENSIONEXECUTE

[1079] typedef struct_D3DDDIARG_EXTENSIONEXECUTE

[1080] {

[1081] HANDLE hExtension ;

[1082] UINT Function ;

[1083] VADDI_PRIVATEDATA * pPrivateInput ;

[1084] VADDI_PRIVATEDATA * pPrivateOutput ;

[1085] UINT NumBuffers ;

[1086] VADDI_PRIVATEBUFFER * pBuffers ;

[1087] } D3DDDIARG_EXTENSIONEXECUTE ;

[1088] typedef HRESULT

[1089] (APIENTRY * PFND3DDDI_CREATEEXTENSIONDEVICE)

[1090] (

[1091] HANDLE hDevice,

[1092] D3DDDIARG_CREATEEXTENSIONDEVICE *

[1093]) ;

[1094] hDeVice 参数涉及 D3D9 设备，并且是使用对 D3DDDI_CREATEDEVICE 的调用来创建的。

[1095] 4.3.3 FND3DDDI_DESTROYEXTENSIONDEVICE

[1096] typedef HRESULT

[1097] (APIENTRY * PFND3DDDI_DESTROYEXTENSIONDEVICE)

[1098] (

[1099] HANDLE hDevice,

[1100] HANDLE hExtension

[1101]) ;

[1102] 4.3.4 FND3DDDI_EXTENSIONEXECUTE

[1103] typedef HRESULT

[1104] (APIENTRY * PFND3DDDI_EXTENSIONEXECUTE)

[1105] (

[1106] HANDLE hDevice,

[1107] CONST D3DDDIARG_EXTENSIONEXECUTE *

[1108]) ;

[1109] 4.3.5 D3DDDI_DEVICEFUNCS

[1110] typedef struct_D3DDDI_DEVICEFUNCS

[1111] {

[1112] PFND3DDDI_CREATEEXTENSIONDEVICE pfnCreateExtensionDevice ;

[1113] PFND3DDDI_DESTROYEXTENSIONDEVICE pfnDestroyExtensionDevice ;

[1114] PFND3DDDI_EXTENSIONEXECUTE pfnExtensionExecute ;

[1115] } D3DDDI_DEVICEFUNCS ;

[1116] 4.4 D3D9 结构和函数

[1117] 以下 D3D 结构和回调表示了获得扩展设备的能力的通用 D3D 机制。

[1118] typedef enum_D3DDDICAPS_TYPE

[1119] {

[1120] D3DDDICAPS_GETEXTENSIONGUIDCOUNT = 31,

[1121] D3DDDICAPS_GETEXTENSIONGUIDS = 32,

[1122] D3DDDICAPS_GETEXTENSIONCAPS = 33,

[1123] } D3DDDICAPS_TYPE ;

[1124] typedef struct_D3DDDIARG_GETCAPS

[1125] {

[1126] D3DDDICAPS_TYPE Type ;

[1127] VOID * pInfo ;

[1128] VOID * pData ;

[1129] UINT DataSize ;

[1130] } D3DDDIARG_GETCAPS ;

[1131] 5 示例性编程模型

[1132] 5.1 流水线效率

[1133] 为了实现最大效率,编码器应用程序以使得 CPU 以及图形硬件两者都被完全利用的方式来结构化。由此,尽管运动估计对于某一帧是正在进行中,但是在一不同的帧上运行量化步骤可能是有益的。

[1134] 获得完全硬件利用是用多线程编码器来促进的。

[1135] 5.1.1 示例 :单个运动矢量 (流水线满)

[1136] 以下 2 线程应用程序 (伪代码) 示出了编码器实现 2 阶段软件流水线的一种方式,并提供了关于如何高效地使用 VA 编码接口的某些方针。特别地,如在软件线程中所见到的,它实施了 k = AllocatorSize 的缓冲。这解决了在提交硬件请求时存在异步性的事实 :硬件线程提交请求,同时软件线程稍后拾取该结果并处理它们。

[1137] HardwareThread( )

[1138] {

[1139] while(Streaming)

[1140] {

[1141] LoadFrame(ppInputBuffer[n]) ;

[1142] codec->ProcessInput(ppInputBuffer[n]) ;// 阻 塞 GetBuffer+Execute

[1143] }

[1144] }

[1145] SoftwareThread()

[1146] {

[1147] k = AllocatorSize() ;

[1148] while(Streaming)

[1149] {

[1150] //k 表示流水线阶段之间的缓冲区

```
[1151]        Codec->ProcessOutput(ppOutputBuffer[n-k]) ;// 等待,ReleaseBuffer
[1152]        VLE() ;
[1153]        Bitstream() ;
[1154]      }
[1155]    }
```

[1156] 以上的 ProcessInput（进程输入）可被认为是 Execute 和 GetBuffer 外部的包装，而 ProcessOutput（进程输出）可被认为是执行事件上的 Wait（等待）以及之后的适当 ReleaseBuffer 调用外部的包装。

[1157] 并不清楚如何确定表示流水线阶段之间的缓冲区的参数 k。它表示分配符大小，并且作为一个起始点，可以使用在编解码器和 VA 编码器对象之间的分配符协商中使用的同一值（队列长度）。如果 k 大于分配符大小，则 ProcessInput 调用很可能甚至在使用 k 个缓冲区之前无论如何都阻塞。

[1158] 应用程序的目标应是最大化花费在 SoftwareThread（软件线程）中的时间而在 ProcessOutput 上没有阻塞。换言之，应用程序大多数时间应在 VLE() 和 Bitstream() 函数上工作。如果硬件非常慢，则尽管分配符大小为“k”，ProcessOutput() 也将阻塞。软件将始终“在前面”。以上流水线仅在硬件处理缓冲区花费的时间与软件运行 VLE 和 Bitstream 花费的时间相同的意义上才是高效的。“k”个缓冲所实现的全部是填充抖动。

[1159] 以下代码段示出了 GetBuffer 和 ReleaseBuffer 的粗略实现。

```
[1160]  IVideoEncoder::GetBuffer(Type,ppBuffer,Blocking)
[1161]  {
[1162]    if(Empty)
[1163]    {
[1164]      if(Blocking)Wait(NotEmptyEvent) ;
[1165]      else return STATUS_EMPTY ;
[1166]    }
[1167]    * ppBuffer = pQueue[Type][Head] ;
[1168]    Head++ ;
[1169]    if(Head == Tail)
[1170]    (
[1171]      Empty = 1 ;
[1172]      ResetEvent(NotEmptyEvent) ;
[1173]    }
[1174]    return S_OK ;
[1175]  }
[1176]  IVideoEncoder::ReleaseBuffer(Type,pBuffer)
[1177]  {
[1178]    if((Tail == Head)&& ! Empty)return STATUS_FULL ;
[1179]    pQueue[Type][Tail] = pBuffer ;
[1180]    Tail++ ;
```

```
[1181]      if(Empty)
[1182]      {
[1183]        Empty = false;
[1184]        SetEvent(NotEmptyEvent);
[1185]      }
[1186]      return S_OK;
[1187]    }
```

[1188] 以下略述了 ProcessInput 和 ProcessOutput 的编解码器实现：

```
[1189]    // 本实现不像普通语义,是阻塞的
[1190]    Codec::ProcessInput(IMediaBuffer pInput)
[1191]    {
[1192]      GetBuffer(TypeUncompressed,pYUVBuffer,true);
[1193]      GetBuffer(TypeMotionVector,pMVBuffer,true);
[1194]      GetBuffer(TypeResidues,pResidueBuffer,true);
[1195]      memcpy(pYUVBuffer,pInput.Image);
[1196]      Execute(pYUVBuffer,pMVBuffer,pResidueBuffer,pEvent);
[1197]      CodecQueue.Enqueue(pYUVBuffer,pMVBuffer,pResidueBuffer,pEvent);
[1198]    }
[1199]    Codec::ProcessOutput(IMediaBuffer pOutput)
[1200]    {
[1201]      if(CodecQueue.Empty())
[1202]      {
[1203]        pOutput.dwFlags = DMO_OUTPUT_DATABUFFERF_INCOMPLETE;
[1204]        return S_FALSE;
[1205]      }
[1206]      CodecQueue.Dequeue(pYUVBuffer,pMVBuffer,pResidueBuffer,pEvent);
[1207]      Wait(pEvent);
[1208]      memcpy(pOutput.MVBuffer,pMVBuffer);
[1209]      memcpy(pOutput.ResidueBuffer,pResidueBuffer);
[1210]      ReleaseBuffer(TypeUncompressed,pYUVBuffer);
[1211]      ReleaseBuffer(TypeMotionVector,pMVBuffer);
[1212]      ReleaseBuffer(TypeResidues,pResidueBuffer);
[1213]      return S_OK;
[1214]    }
```

[1215] 此处是 Codec::ProcessInput 的一个替换实现，它如普通语义一样是非阻塞的。

```
[1216]    Codec::ProcessInput(IMediaBuffer pInput)
[1217]    {
[1218]      if(GetBuffer(TypeUncompressed,pYUVBuffer,false) == STATUS_EMPTY)
[1219]      {
```

```
[1220]        return DMO_E_NOTACCEPTING;
[1221]      }
[1222]      if(GetBuffer(TypeMotionVector,pMVBuffer,false) == STATUS_EMPTY)
[1223]      {
[1224]        return DMO_E_NOTACCEPTING;
[1225]      }
[1226]      if(GetBuffer(TypeResidues,pResidueBuffer,false) == STATUS_EMPTY)
[1227]       {
[1228]           return DMO_E_NOTACCEPTING;
[1229]       }
[1230]       memcpy(pYUVBuffer,pInput.Image);
[1231]       Execute(pYUVBuffer,pMVBuffer,pResidueBuffer,pEvent);
[1232]       CodecQueue.Enqueue(pYUVBuffer,pMVBuffer,pResidueBuffer,pEvent);
[1233]   }
```

[1234] 5.2.1 示例:多个运动矢量

[1235] 在本节中,观察其中编码器向硬件请求多个运动矢量并且基于各种参数选择其中一个并重新提交它们以便处理的更复杂的流水线。以下代码单纯地如上一样继续使用 2 阶段流水线,请求多个运动矢量并重新提交最佳的那一个。在此实现中涉及固有串行化。

```
[1236]   HardwareThread()
[1237]   {
[1238]      while(Streaming)
[1239]      {
[1240]         LoadFrame(ppInputBuffer[n]);
[1241]         ProcessInput(ppInputBuffer[n]);
[1242]         ProcessOutput(ppOutputBuffer[n]);
[1243]         SelectMV(ppOutputBuffer[n],ppOutputBuffer2[n]);
[1244]         ProcessInput2(ppOutputBuffer2[n]);
[1245]         n++;
[1246]      }
[1247]   }
[1248]   SoftwareThread()
[1249]   {
[1250]      while(Streaming)
[1251]      {
[1252]         ProcessOutput2(ppOutputBuffer2[n-k]);
[1253]         VLE(ppOutputBuffer2[n-k]);
[1254]         Bitstream(ppOutputBuffer2[n-k]);
[1255]      }
[1256]   }
```

[1257] 在以上示例中，软件将有一半时间在 ProcessOutput 和 ProcessOutput2 上阻塞，这很明显是不利于流水线效率的。另一方面，CPU 利用是相当低的，并且总吞吐量仍要高于非加速编码。基于 3 个线程的 3 阶段流水线将如下解决串行化问题：

[1258]
```
HardwareThread1()
```
[1259]
```
{
```
[1260]
```
   while(Streaming)
```
[1261]
```
   {
```
[1262]
```
      LoadFrame(ppInputBuffer[n]);
```
[1263]
```
      ProcessInput(ppInputBuffer[n]);
```
[1264]
```
   }
```
[1265]
```
}
```
[1266]
```
HardwareThread2()
```
[1267]
```
{
```
[1268]
```
   while(Streaming)
```
[1269]
```
   {
```
[1270]
```
      ProcessOutput(ppOutputBuffer[n-k1]);
```
[1271]
```
      SelectMV(ppOutputBuffer[n-k1],ppOutputBuffer2[n-k1]);
```
[1272]
```
      ProcessInput2(ppOutputBuffer2[n-k1]);
```
[1273]
```
    }
```
[1274]
```
}
```
[1275]
```
SoftwareThread()
```
[1276]
```
{
```
[1277]
```
   while(Streaming)
```
[1278]
```
   {
```
[1279]
```
      ProcessOutput2(ppOutputBuffer2[n-k1-k2]);
```
[1280]
```
      VLE(ppOutputBuffer2[n-k1-k2]);
```
[1281]
```
      Bitstream(ppOutputBuffer2[n- k1-k2]);
```
[1282]
```
    }
```
[1283]
```
}
```

[1284] 由于有 3 个流水线阶段，因此添加了附加的缓冲区以便在两个硬件阶段之间填充。因此有两个值 k1 和 k2。

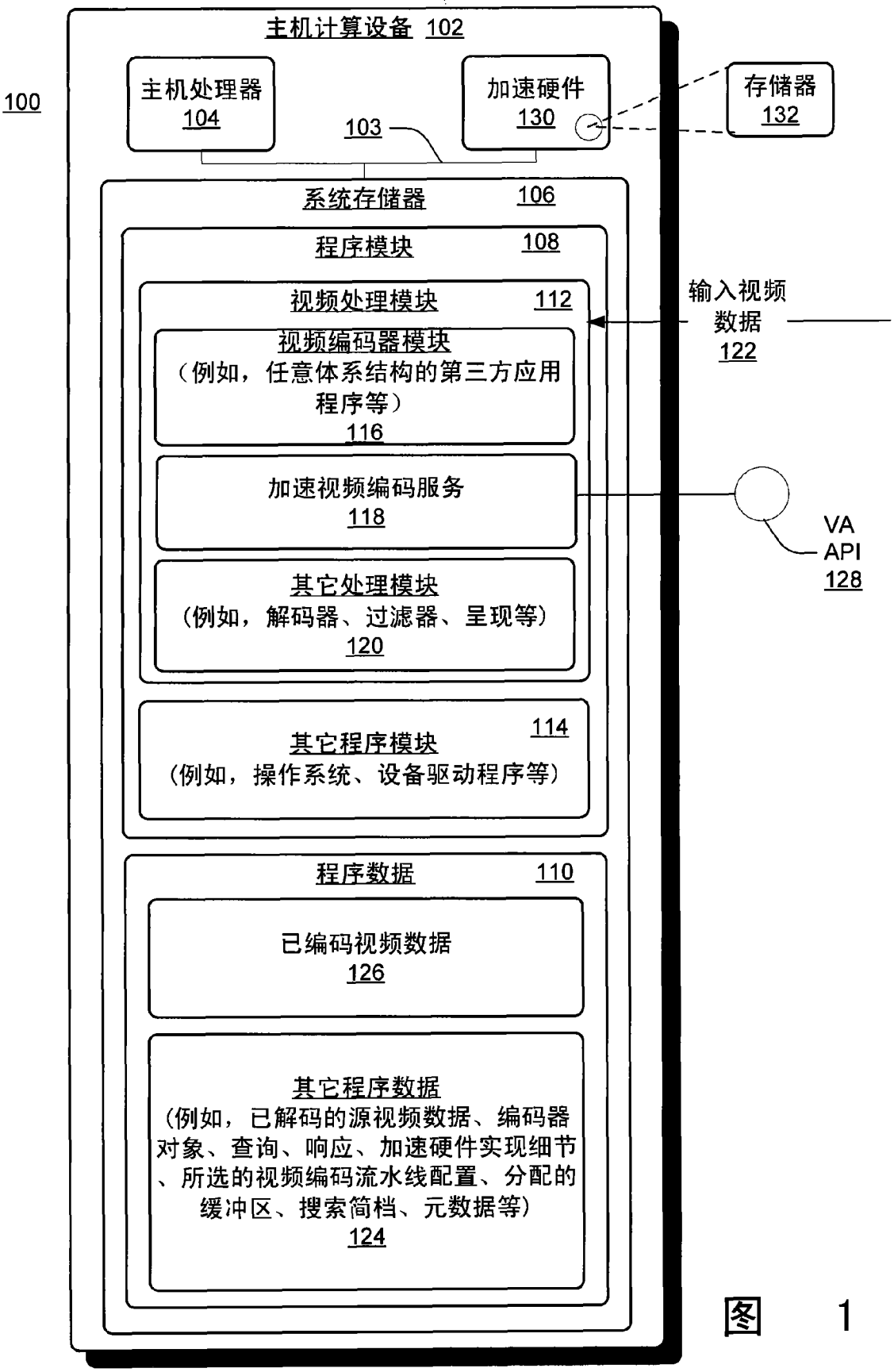

100
主机计算设备 102
主机处理器 104
103
加速硬件 130
存储器 132
系统存储器 106
程序模块 108
视频处理模块 112
视频编码器模块
（例如，任意体系结构的第三方应用程序等）
116
加速视频编码服务
118
其它处理模块
（例如，解码器、过滤器、呈现等）
120
其它程序模块 114
（例如，操作系统、设备驱动程序等）
程序数据 110
已编码视频数据
126
其它程序数据
（例如，已解码的源视频数据、编码器对象、查询、响应、加速硬件实现细节、所选的视频编码流水线配置、分配的缓冲区、搜索简档、元数据等）
124
输入视频数据
122
VA API
128

图 1

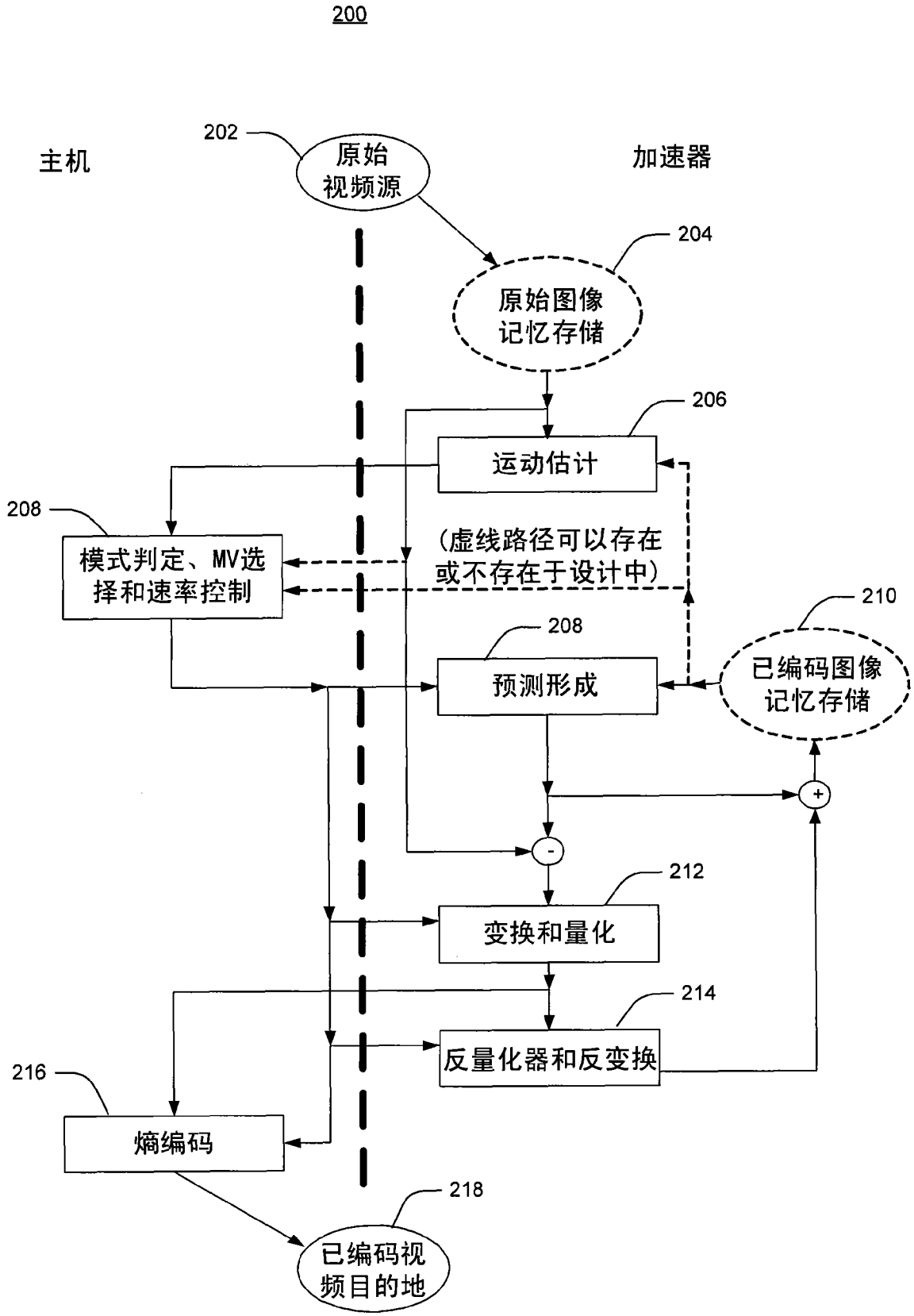

200
主机
加速器
202
原始
视频源
204
原始图像
记忆存储
206
运动估计
208
模式判定、MV选
择和速率控制
(虚线路径可以存在
或不存在于设计中)
210
已编码图像
记忆存储
208
预测形成
+
-
212
变换和量化
214
反量化器和反变换
216
熵编码
218
已编码视
频目的地

图 2

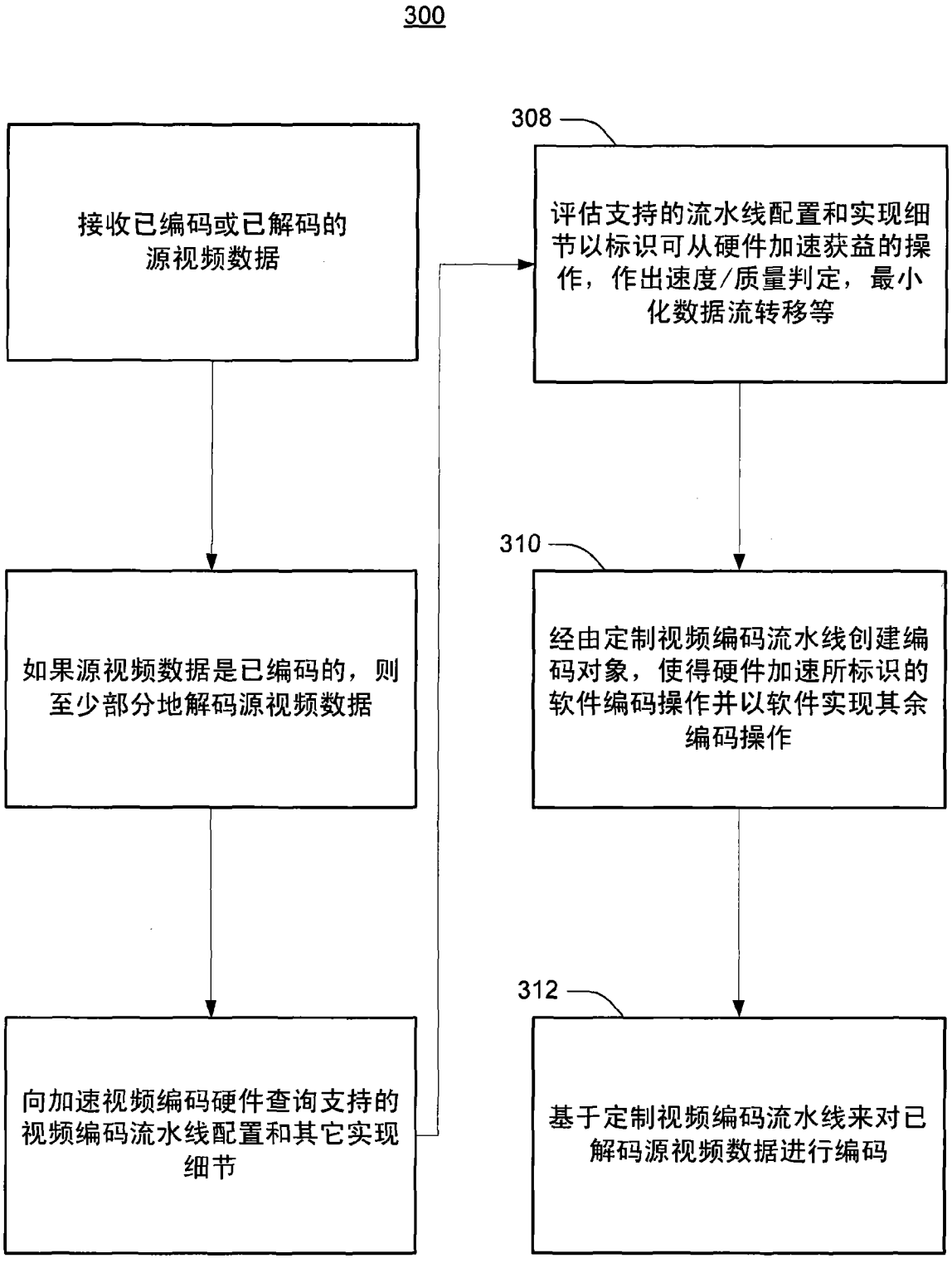

300
接收已编码或已解码的源视频数据
如果源视频数据是已编码的，则至少部分地解码源视频数据
向加速视频编码硬件查询支持的视频编码流水线配置和其它实现细节
308
评估支持的流水线配置和实现细节以标识可从硬件加速获益的操作，作出速度/质量判定，最小化数据流转移等
310
经由定制视频编码流水线创建编码对象，使得硬件加速所标识的软件编码操作并以软件实现其余编码操作
312
基于定制视频编码流水线来对已解码源视频数据进行编码

图 3

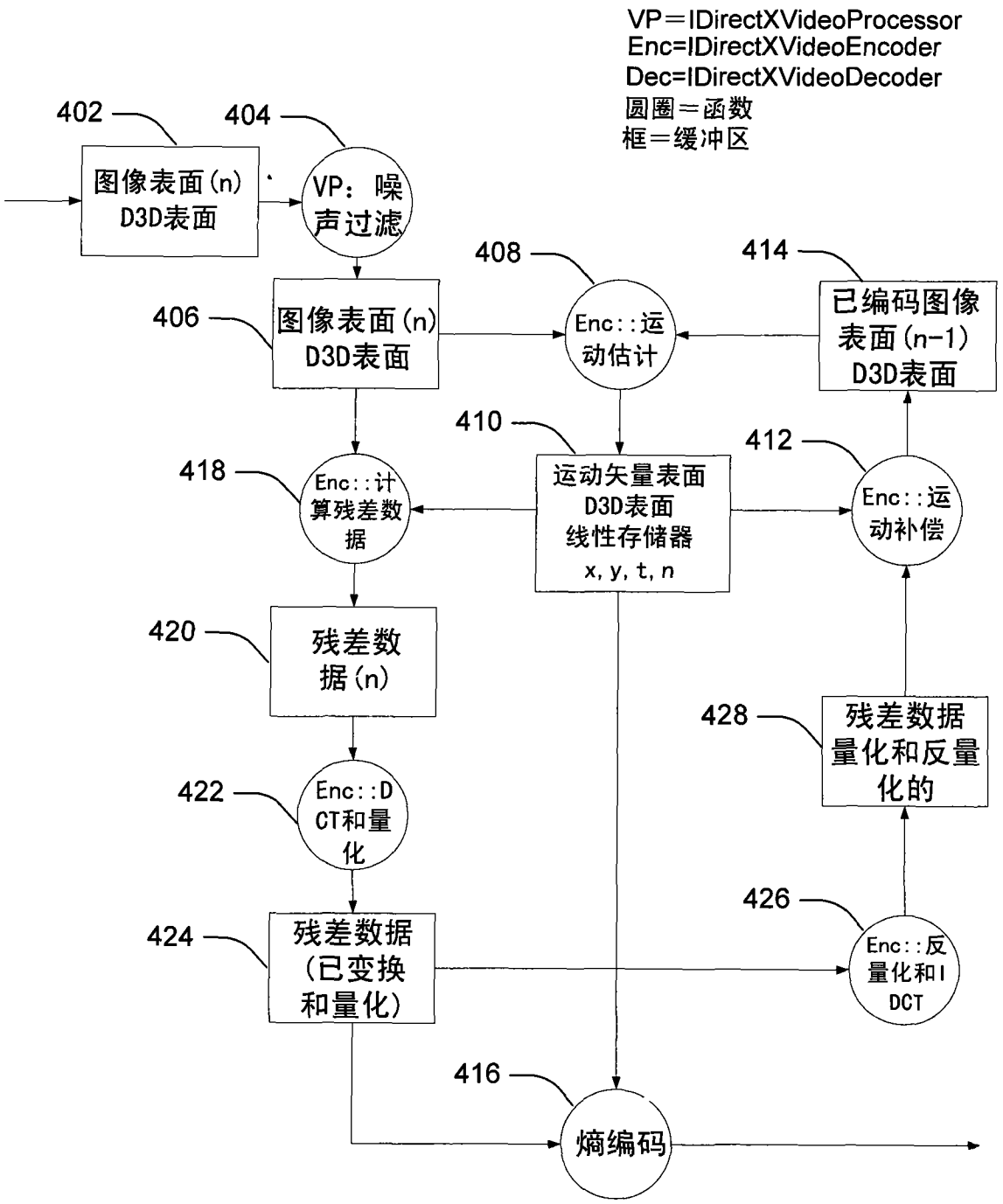

VP＝IDirectXVideoProcessor
Enc=IDirectXVideoEncoder
Dec=IDirectXVideoDecoder
圆圈＝函数
框＝缓冲区
402
404
图像表面(n)
D3D表面
VP：噪声过滤
408
414
406
图像表面(n)
D3D表面
Enc::运动估计
已编码图像表面(n-1)
D3D表面
410
412
418
Enc::计算残差数据
运动矢量表面
D3D表面
线性存储器
x, y, t, n
Enc::运动补偿
420
残差数据(n)
428
残差数据量化和反量化的
422
Enc::DCT和量化
426
424
残差数据(已变换和量化)
Enc::反量化和IDCT
416
熵编码

图 4

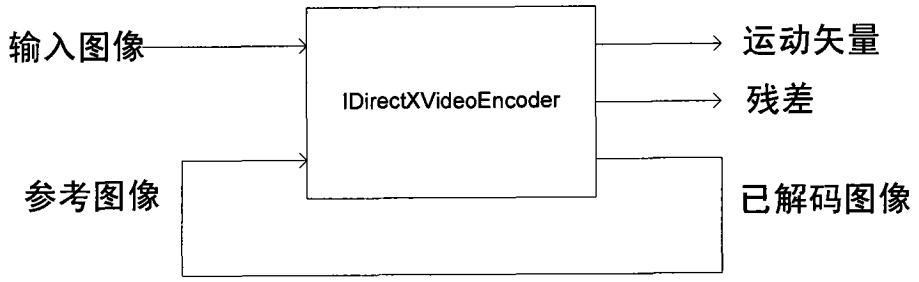

输入图像
参考图像
IDirectXVideoEncoder
运动矢量
残差
已解码图像

图 5

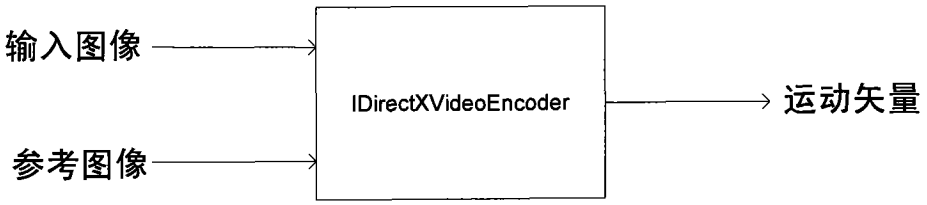

输入图像
参考图像
IDirectXVideoEncoder
运动矢量

图 6

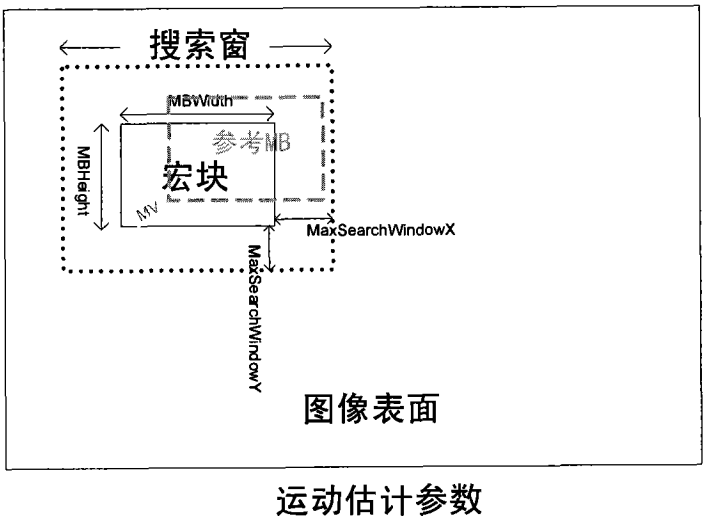

搜索窗
MBWidth
参考MB
宏块
MBHeight
MV
MaxSearchWindowX
MaxSearchWindowY
图像表面
运动估计参数

图 7

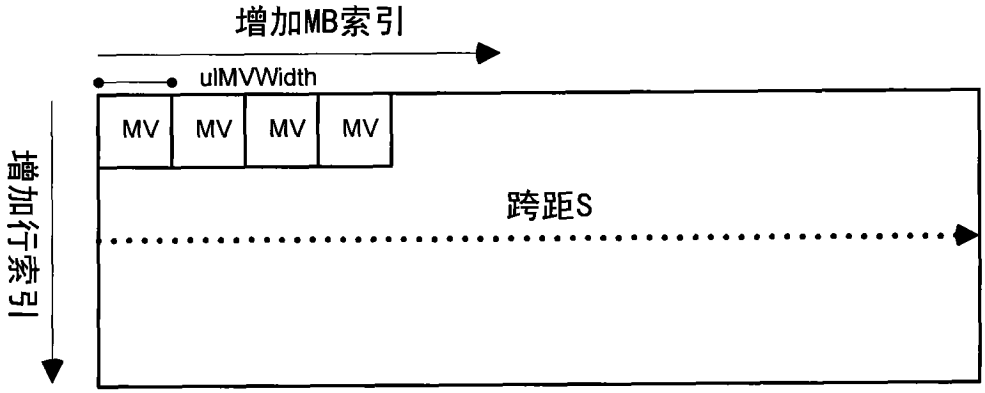

增加MB索引
ulMVWidth
MV
MV
MV
MV
跨距S
增加行索引

图 8

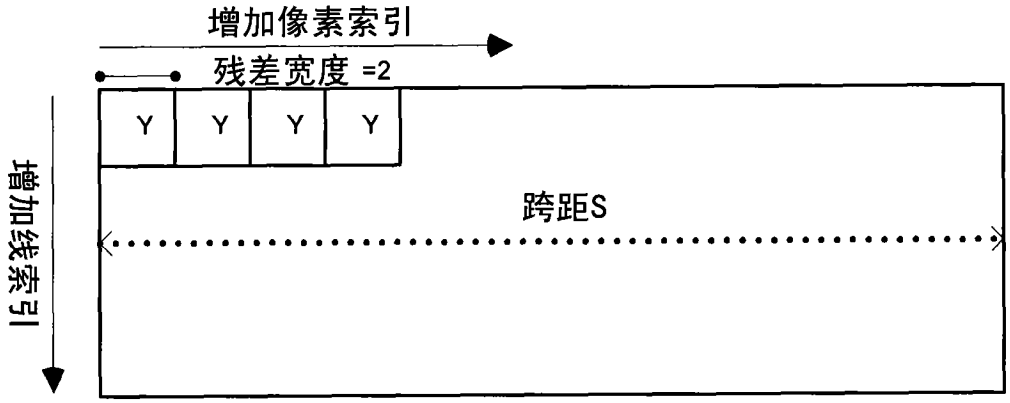

增加像素索引
残差宽度 =2
Y
Y
Y
Y
跨距S
增加线索引

图 9

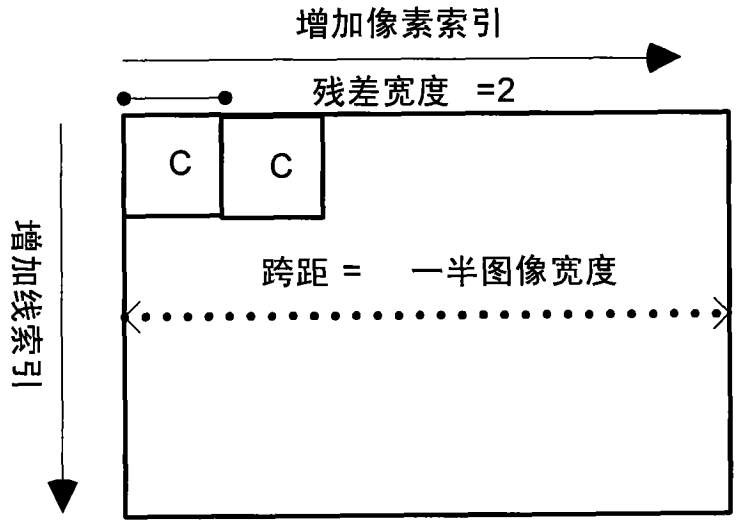

增加像素索引
残差宽度 =2
C
C
增加线索引
跨距 = 一半图像宽度

图 10

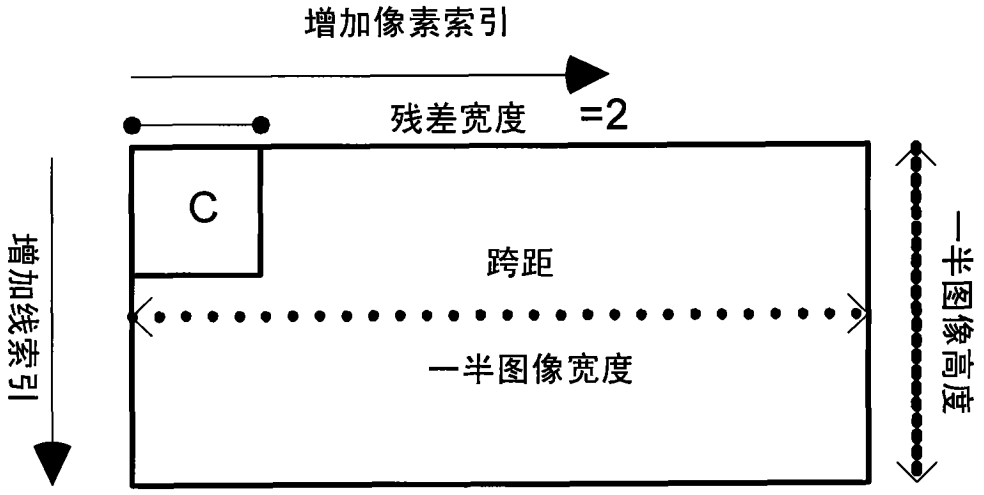

增加像素索引
残差宽度 =2
C
增加线索引
跨距
一半图像宽度
一半图像高度

图 11