(19)国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 117499679 A
(43)申请公布日 2024.02.02

(21)申请号 202311450652.0
(22)申请日 2020.04.16
(66)本国优先权数据
PCT/CN2019/082855 2019.04.16 CN
(62)分案原申请数据
202080028533.4 2020.04.16
(71)申请人 北京字节跳动网络技术有限公司
地址 100041 北京市石景山区实兴大街30号院3号楼2层B-0035房间
申请人 字节跳动有限公司
(72)发明人 刘鸿彬 张莉 张凯 庄孝强 邓智玭
(74)专利代理机构 北京市柳沈律师事务所 11105
专利代理师 张亮 刘文洁
(51)Int.Cl.
H04N 19/82(2014.01)
H04N 19/117(2014.01)
H04N 19/70(2014.01)
H04N 19/176(2014.01)
H04N 19/186(2014.01)
H04N 19/14(2014.01)

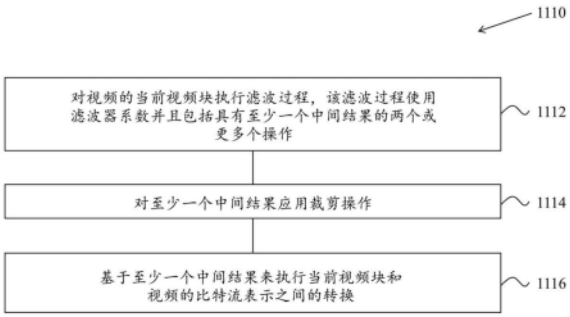
权利要求书3页 说明书48页 附图20页

(54)发明名称

用于视频编解码的自适应环路滤波

(57)摘要

描述了用于自适应环路滤波的设备、系统和方法。在示例性方面,一种用于视频处理的方法包括:对视频的当前视频块执行滤波过程,其中该滤波过程使用滤波器系数并且包括具有至少一个中间结果的两个或更多个操作;对至少一个中间结果应用裁剪操作;以及基于至少一个中间结果来执行当前视频块和视频的比特流表示之间的转换,其中,至少一个中间结果基于滤波器系数和当前视频块的当前样点与当前样点的临近样点之间的差的加权和。



CN 117499679 A

1.一种处理视频数据的方法,包括:

为视频的第一视频区域和视频的比特流之间的转换,确定比特流中存在对应于所述第一视频区域的第一语法元素,其中所述第一语法元素指示所述第一视频区域所参考的第一自适应参数集的索引;以及

基于所述确定执行所述转换;

其中,响应于所述确定,包括在所述第一自适应参数集中的第二语法元素的值是1,并且其中所述第二语法元素指示在所述第一自适应参数集中是否包括亮度滤波器集,以及

其中,对于所述第一视频区域的样点,基于不同方向上的多个样点差来导出滤波器索引,并且基于所述滤波器索引从所述亮度滤波器集中选择特定的亮度滤波器。

2.根据权利要求1所述的方法,其中,所述第一视频区域是视频图片,并且所述第一语法元素存在于图片头语法元素集中。

3.根据权利要求1所述的方法,其中,所述第一视频区域是视频条带,并且所述第一语法元素存在于条带头语法元素集中。

4.根据权利要求1所述的方法,其中,所述视频区域包括多个视频编解码树块,并且所述视频编解码树块被划分成多个M*M视频块,M等于2或4,并且

其中,对同一M*M视频块中的样点应用相同的滤波器索引。

5.根据权利要求4所述的方法,其中,基于1:2的子采样率来针对每个M*M视频区域导出所述不同方向上的多个样点差。

6.根据权利要求1所述的方法,其中,所述第一自适应参数集包括自适应环路滤波自适应参数集。

7.根据权利要求1所述的方法,其中,所述滤波器索引的最大值由N表示,并且所述亮度滤波器集中的亮度滤波器的数量小于或等于N+1。

8.根据权利要求7所述的方法,其中,响应于所述亮度滤波器集中的亮度滤波器的数量小于N+1,具有不同的滤波器索引的至少两个亮度滤波器具有相同的滤波器系数。

9.根据权利要求1所述的方法,其中,所述滤波器索引用于导出没有包括在所述亮度滤波器集中的另一特定亮度滤波器。

10.根据权利要求1所述的方法,其中,所述转换包括将所述第一视频区域编码成所述比特流。

11.根据权利要求1所述的方法,其中,所述转换包括从所述比特流解码所述第一视频区域。

12.一种用于处理视频数据的装置,包括处理器和其上具有指令的非暂时性存储器,其中所述指令在由所述处理器执行时使得所述处理器:

为视频的第一视频区域和视频的比特流之间的转换,确定比特流中存在对应于所述第一视频区域的第一语法元素,其中所述第一语法元素指示所述第一视频区域所参考的第一自适应参数集的索引;以及

基于所述确定执行所述转换;

其中,响应于所述确定,包括在所述第一自适应参数集中的第二语法元素的值是1,并且其中所述第二语法元素指示在所述第一自适应参数集中是否包括亮度滤波器集,以及

其中,对于所述第一视频区域的样点,基于不同方向上的多个样点差来导出滤波器索

引,并且基于所述滤波器索引从所述亮度滤波器集中选择特定的亮度滤波器。

13.根据权利要求12所述的装置,其中,所述第一视频区域是视频图片,并且所述第一语法元素存在于图片头语法元素集中。

14.根据权利要求12所述的装置,其中,所述第一视频区域是视频条带,并且所述第一语法元素存在于条带头语法元素集中。

15.根据权利要求12所述的装置,其中,所述视频区域包括多个视频编解码树块,并且所述视频编解码树块被划分成多个M*M视频块,M等于2或4,并且

其中对同一M*M视频块中的样点应用相同的滤波器索引。

16.根据权利要求12所述的装置,其中,所述第一自适应参数集包括自适应环路滤波自适应参数集。

17.根据权利要求12所述的装置,其中,所述滤波器索引的最大值由N表示,并且所述亮度滤波器集中的亮度滤波器的数量小于或等于N+1。

18.根据权利要求17所述的装置,其中,响应于所述亮度滤波器集中的亮度滤波器的数量小于N+1,具有不同的滤波器索引的至少两个亮度滤波器具有相同的滤波器系数。

19.一种存储指令的非暂时性计算机可读存储介质,所述指令使得处理器:

为视频的第一视频区域和视频的比特流之间的转换,确定比特流中存在对应于所述第一视频区域的第一语法元素,其中所述第一语法元素指示所述第一视频区域所参考的第一自适应参数集的索引;以及

基于所述确定执行所述转换;

其中,响应于所述确定,包括在所述第一自适应参数集中的第二语法元素的值是1,并且其中所述第二语法元素指示在所述第一自适应参数集中是否包括亮度滤波器集,以及

其中,对于所述第一视频区域的样点,基于不同方向上的多个样点差来导出滤波器索引,并且基于所述滤波器索引从所述亮度滤波器集中选择特定的亮度滤波器。

20.一种存储由视频处理装置执行的方法生成的视频的比特流的非暂时性计算机可读记录介质,其中所述方法包括:

为视频的第一视频区域,确定比特流中存在对应于所述第一视频区域的第一语法元素,其中所述第一语法元素指示所述第一视频区域所参考的第一自适应参数集的索引;以及

基于所述确定生成所述比特流;

其中,响应于所述确定,包括在所述第一自适应参数集中的第二语法元素的值是1,并且其中所述第二语法元素指示在所述第一自适应参数集中是否包括亮度滤波器集,以及

其中,对于所述第一视频区域的样点,基于不同方向上的多个样点差来导出滤波器索引,并且基于所述滤波器索引从所述亮度滤波器集中选择特定的亮度滤波器。

21.一种存储视频的比特流的方法,包括:

为视频的第一视频区域,确定比特流中存在对应于所述第一视频区域的第一语法元素,其中所述第一语法元素指示所述第一视频区域所参考的第一自适应参数集的索引;

基于所述确定生成所述比特流;以及

将所生成的比特流存储到非暂时性计算机可读记录介质上,

其中,响应于所述确定,包括在所述第一自适应参数集中的第二语法元素的值是1,并

且其中所述第二语法元素指示在所述第一自适应参数集中是否包括亮度滤波器集,以及

其中,对于所述第一视频区域的样点,基于不同方向上的多个样点差来导出滤波器索引,并且基于所述滤波器索引从所述亮度滤波器集中选择特定的亮度滤波器。

用于视频编解码的自适应环路滤波

[0001] 本申请是于2021年10月13日提交的中国发明专利申请第202080028533.4号的分案申请,该中国发明专利申请是基于2020年4月16日提交的国际专利申请PCT/CN2020/085068的中国国家阶段,其要求于2019年4月16日提交的国际专利申请No.PCT/CN2019/082855的优先权和权益。前述申请的全部公开通过引用而并入作为本申请的公开的一部分。

技术领域

[0002] 本专利文档涉及视频编解码技术、设备和系统。

背景技术

[0003] 尽管视频压缩技术有所进步,但数字视频仍然在互联网和其它数字通信网络上占据最大的带宽使用。随着能够接收和显示视频的联网用户设备数量的增加,预计对数字视频使用的带宽需求将继续增长。

发明内容

[0004] 描述了与数字视频编解码相关,并且更具体地,与用于视频编解码的自适应环路滤波相关的设备、系统和方法。所描述的方法可以被应用于现有的视频编解码标准(例如,高效视频编解码(High Efficiency Video Coding,HEVC))和未来的视频编解码标准(例如,通用视频编解码(Versatile Video Coding,VVC))或编解码器。

[0005] 视频编解码标准主要是通过众所周知的ITU-T和ISO/IEC标准的发展而演变的。ITU-T产生了H.261和H.263,ISO/IEC产生了MPEG-1和MPEG-4Visual,并且这两个组织联合产生了H.262/MPEG-2视频和264/MPEG-4高级视频编解码(Advanced Video Coding,AVC)标准和H.265/HEVC标准。自H.262以来,视频编解码标准基于混合视频编解码结构,其中利用了时域预测加变换编解码。为了探索HEVC以外的未来的视频编解码技术,VCEG和MPEG于2015年联合成立了联合视频探索小组(Joint Video Exploration Team,JVET)。此后,JVET采用了许多新方法,并将其放入到命名为联合探索模型(Joint Exploration Model,JEM)的参考软件中。2018年4月,建立了VCEG(Q6/16)和ISO/IEC JTC1 SC29/WG11(MPEG)之间的联合视频专家小组(Joint Video Expert Team,JVET)以致力于VVC标准,目标是与HEVC相比的50%比特率降低。

[0006] 在一个代表性方面,所公开的技术可以用于提供一种用于视频处理的方法。该方法包括:对视频的当前视频块执行滤波过程,其中该滤波过程使用滤波器系数并且包括具有至少一个中间结果的两个或更多个操作;对至少一个中间结果应用裁剪操作;以及基于至少一个中间结果来执行当前视频块和视频的比特流表示之间的转换,其中,至少一个中间结果基于滤波器系数和当前视频块的当前样点与当前样点的临近样点之间的差的加权和。

[0007] 在另一个代表性方面,所公开的技术可以用于提供一种用于视频处理的方法。该

方法包括:将视频的当前视频块编码为视频的比特流表示,其中,当前视频块是用自适应环路滤波器(ALF)编解码的;以及基于一个或多个时域自适应滤波器集合的可用性或使用,在比特流表示中选择性地包括一个或多个时域自适应滤波器集合内的时域自适应滤波器集合的指示。

[0008] 在又一个代表性方面,所公开的技术可以用于提供一种用于视频处理的方法。该方法包括:基于视频的比特流表示中的时域自适应滤波器集合的指示来确定一个或多个时域自适应滤波器集合的可用性或使用,其中该一个或多个时域自适应滤波器集合包括适用于用自适应环路滤波器(ALF)编解码的视频的当前视频块的时域自适应滤波器集合;以及通过基于该确定来选择性地应用时域自适应滤波器集合,从比特流表示生成解码的当前视频块。

[0009] 在又一个代表性方面,所公开的技术可以用于提供一种用于视频处理的方法。该方法包括:基于可用的时域自适应环路滤波(ALF)系数集,为用自适应环路滤波器编解码的当前视频块确定多个时域ALF系数集,其中,可用的时域ALF系数集已经在该确定之前被编码或解码,并且其中,多个ALF系数集用于包括当前视频块的片组、片、条带、图片、编解码树块(CTB)或视频单元;以及基于多个时域ALF系数集,执行当前视频块和当前视频块的比特流表示之间的转换。

[0010] 在又一个代表性方面,所公开的技术可以用于提供一种用于视频处理的方法。该方法包括:为视频的当前视频块和视频的比特流表示之间的转换确定视频的视频区域的头中的自适应环路滤波(ALF)的指示等于与比特流表示相关联的自适应参数集(APS)网络抽象层(NAL)单元中的ALF的指示;以及执行转换。

[0011] 在又一个代表性方面,所公开的技术可以用于提供一种用于视频处理的方法。该方法包括:基于视频的视频区域使用的自适应环路滤波器的类型,对视频的当前视频块和视频的比特流表示之间的转换选择性地启用非线性自适应环路滤波(ALF)操作;以及在该选择性地启用之后执行转换。

[0012] 在又一个代表性方面,上述方法以处理器可执行代码的形式被体现并且被存储在计算机可读程序介质中。

[0013] 在又一个代表性方面,公开了一种被配置为或可操作来执行上述方法的设备。该设备可以包括被编程来实施该方法的处理器。

[0014] 在再一个代表性方面,视频解码器装置可以实施如本文所描述的方法。

[0015] 在附图、说明书和权利要求书中更详细地描述了所公开的技术的以上以及其它方面和特征。

附图说明

[0016] 图1示出了用于视频编解码的编码器框图的示例。

[0017] 图2A、图2B和图2C示出了基于几何变换的自适应环路滤波器(GALF)滤波器形状的示例。

[0018] 图3示出了GALF编码器决定的流程图的示例。

[0019] 图4A-图4D示出了用于自适应环路滤波器(ALF)分类的示例子采样拉普拉斯算子计算。

[0020] 图5示出了亮度滤波器形状的示例。

[0021] 图6示出了宽视频图形阵列(WVGA)序列的区域划分的示例。

[0022] 图7示出了具有重整形(reshape)的解码流程的示例性流程图。

[0023] 图8示出了双向光流(BIO)算法使用的光流轨迹的示例。

[0024] 图9A和图9B示出了在没有块扩展的情况下使用双向光流(BIO)算法的示例快照。

[0025] 图10示出了利用光流的预测细化(PROF)的示例。

[0026] 图11A-图11F示出了根据所公开的技术的用于自适应环路滤波的示例方法的流程图。

[0027] 图12是用于实施在本文档中描述的可视媒体解码或可视媒体编码技术的硬件平台的示例的框图。

[0028] 图13是其中可以实施所公开的技术的示例视频处理系统的框图。

具体实施方式

[0029] 由于对更高分辨率视频的增长的需求,视频编解码方法和技术在现代技术中无处不在。视频编解码器通常包括压缩或解压缩数字视频的电子电路或软件,并且被不断改进以提供更高的编解码效率。视频编解码器将未压缩的视频转换为压缩格式,反之亦然。视频质量、用于表示视频的数据量(由比特率确定)、编解码和解码算法的复杂性、对数据丢失和错误的敏感性、编辑的便利性、随机访问和端到端延迟(时延)之间存在复杂的关系。压缩格式通常符合标准视频压缩规范,例如高效视频编解码(HEVC)标准(也称为H.265或MPEG-H第2部分)、待最终确定的通用视频编解码(VVC)标准、或其他当前和/或未来的视频编解码标准。

[0030] 在一些实施例中,使用称为联合探索模型(JEM)的参考软件来探索未来的视频编解码技术。在JEM中,在几个编解码工具(诸如仿射预测、可选时域运动矢量预测(ATMVP)、空时运动矢量预测(STMVP)、双向光流(BIO)、帧速率上转换(FRUC)、局部自适应运动矢量分辨率(LAMVR)、重叠块运动补偿(OBMC)、局部照明补偿(LIC)和解码器侧运动矢量细化(DMVR))中采用基于子块的预测。

[0031] 所公开的技术的实施例可以被应用于现有的视频编解码标准(例如,HEVC、H.265)和未来的标准,以提高运行时性能。在本文档中使用章节标题以提高描述的可读性,并且不以任何方式将讨论或实施例(和/或实施方式)仅限制于相应的章节。

[0032] 1色彩空间和色度子采样的示例

[0033] 色彩空间,也称为色彩模型(或色彩系统),是抽象的数学模型,其简单地将色彩的范围描述为数字的元组,通常为3或4个值或色彩分量(例如,RGB)。从根本上讲,色彩空间是坐标系和子空间的详化。

[0034] 对于视频压缩,最常用的色彩空间是YCbCr和RGB。

[0035] YCbCr、Y′CbCr、或Y Pb/Cb Pr/Cr,也写为YCBCR或Y'CBCR,是色彩空间家族,在视频和数字摄影系统中用作色彩图像流水线的一部分。Y′是亮度分量,CB和CR是蓝差和红差色度分量。Y′(有基色)不同于作为亮度的Y,意味着光强度是基于伽马校正的RGB基色非线性编解码的。

[0036] 色度子采样是通过对色度信息实施比亮度信息更低的分辨率来编码图像的实践,

利用了人类视觉系统对色差比亮度更低的敏锐度。

[0037] 1.1 4:4:4色彩格式

[0038] 三个Y'CbCr分量中的每一个都具有相同的采样率,因此没有色度子采样。该方案有时用于高端电影扫描仪和电影后期制作中。

[0039] 1.2 4:2:2色彩格式

[0040] 两个色度分量以亮度采样率的一半进行采样,例如水平色度分辨率被减半。这将未压缩视频信号的带宽减少了三分之一,几乎没有视觉差异。

[0041] 1.3 4:2:0色彩格式

[0042] 在4:2:0下,水平采样与4:1:1相比被加倍,但由于Cb和Cr通道在该方案中仅在每条交替线上被采样,因此垂直分辨率被减半。数据速率因此相同。Cb和Cr每个水平地和垂直地以因子2进行子采样。有4:2:0方案的三种变体,具有不同的水平和垂直选点(siting)。

[0043] o在MPEG-2中,Cb和Cr是水平共置的。Cb和Cr位于垂直方向上的像素之间(位于中间)。

[0044] o在JPEG/JFIF、H.261和MPEG-1中,Cb和Cr位于中间,在交替亮度样点的中间。

[0045] o在4:2:0DV中,Cb和Cr在水平方向上共置。在垂直方向上,它们在交替线上共置。

[0046] 2典型视频编解码器的编解码流程示例

[0047] 图1示出了VVC的编码器框图的示例,其包含三个环路滤波块:去方块滤波器(DF)、样点自适应偏移(SAO)和ALF。与使用预定义的滤波器的DF不同,SAO和ALF利用当前图片的原始样点,用信令通知偏移和滤波器系数的编解码边信息,分别通过添加偏移和通过应用有限脉冲响应(FIR)滤波器来减小原始样点和重构样点之间的均方误差。ALF位于每个图片的最后一个处理阶段,并且可以被看作试图捕捉和修复前一阶段创建的伪像(artifact)的工具。

[0048] 3 JEM中的基于几何变换的自适应环路滤波器的示例

[0049] 在JEM中,利用基于块的滤波器自适应[3]的基于几何变换的自适应环路滤波器(GALF)被应用。对于亮度分量,基于局部梯度的方向和活动,为每个2×2块选择25个滤波器当中的一个。

[0050] 3.1滤波器形状的示例

[0051] 在JEM中,可以为亮度分量选择至多三个菱形滤波器形状(如图2A、图2B和图2C所示,分别为5×5菱形、7×7菱形和9×9菱形)。在图片级别信令通知索引,以指示用于亮度分量的滤波器形状。对于图片中的色度分量,总是使用5×5菱形形状。

[0052] 3.1.1块分类

[0053] 每个2×2块被归类为25个类别中的一个。分类索引C基于其方向性D和活动$\hat{A}$的量化值进行推导,如下:

[0054] $$C = 5D + \hat{A} \quad (1)$$

[0055] 为了计算D和$\hat{A}$,首先使用1-D拉普拉斯算子计算水平、垂直和两个对角线方向的梯度:

[0056] $$g_v = \sum_{k=i-2}^{i+3} \sum_{l=j-2}^{j+3} V_{k,l}, \; V_{k,l} = |2R(k,l) - R(k,l-1) - R(k,l+1)|, \quad (2)$$

[0057] $$g_h = \sum_{k=i-2}^{i+3}\sum_{l=j-2}^{j+3} H_{k,l},\ H_{k,l} = |2R(k,l) - R(k-1,l) - R(k+1,l)|, \quad (3)$$

[0058] $$g_{d1} = \sum_{k=i-2}^{i+3}\sum_{l=j-3}^{j+3} D1_{k,l}, D1_{k,l} = |2R(k,l) - R(k-1,l-1) - R(k+1,l+1)| \quad (4)$$

[0059] $$g_{d2} = \sum_{k=i-2}^{i+3}\sum_{j=j-2}^{j+3} D2_{k,l},\ D2_{k,l} = |2R(k,l) - R(k-1,l+1) - R(k+1,l-1)| \quad (5)$$

[0060] 索引i和j表示2×2块中的左上方样点的坐标,并且R(i,j)表示坐标(i,j)处的重构样点。

[0061] 然后水平和垂直方向的梯度的D的最大值和最小值被设置为:

[0062] $$g_{h,v}^{max} = max(g_h, g_v),\ g_{h,v}^{min} = min(g_h, g_v), \quad (6)$$

[0063] 并且两个对角线方向的梯度的最大值和最小值被设置为:

[0064] $$g_{d0,d1}^{max} = \max(g_{d0}, g_{d1}),\ g_{d0,d1}^{min} = min(g_{d0}, g_{d1}) \quad (7)$$

[0065] 为了推导方向性D的值,将这些值相互比较,并且与两个阈值r_1和t_2进行比较:

[0066] 步骤1.如果$g_{h,v}^{max} \le t_1 \cdot g_{h,v}^{min}$和$g_{d0,d1}^{max} \le t_1 \cdot g_{d0,d1}^{min}$都为真,则S被设置为0。

[0067] 步骤2.如果$g_{h,v}^{max}/g_{h,v}^{min} > g_{d0,d1}^{max}/g_{d0,d1}^{min}$,从步骤3继续;否则,从步骤4继续。

[0068] 步骤3.如果$g_{h,v}^{max} > t_2 \cdot g_{h,v}^{min}$,则S被设置为2;否则,D被设置为1。

[0069] 步骤4.如果$g_{d0,d1}^{max} > t_2 \cdot g_{d0,d1}^{min}$,则D被设置为4;否则,D被设置为3。

[0070] 活动值A被计算为:

[0071] $$A = \sum_{k=i-2}^{i+3}\sum_{l=j-2}^{j+3}(V_{k,l} + H_{k,l}). \quad (8)$$

[0072] A被进一步量化到0至4的范围(包括0和4),并且量化值被表示为$\hat{A}$。

[0073] 对于图片中的两个色度分量,不应用分类方法,即对每个色度分量应用单个ALF系数集合。

[0074] 3.1.2滤波器系数的几何变换

[0075] 在对每个2×2块进行滤波之前,根据为该块计算的梯度值,对滤波器系数f(k,l)应用几何变换,诸如旋转或对角线和垂直翻转。这相当于将这些变换应用于滤波器支持区域中的样点。这个想法通过对齐应用ALF的不同块的方向性来使该不同块更加类似。

[0076] 引入了三种几何变换,包括对角线、垂直翻转和旋转:

[0077] $$\text{对角线:}\ f_D(k,l) = f(l,k),$$
$$\text{垂直翻转:}\ f_V(k,l) = f(k,K-l-1), \quad (9)$$
$$\text{旋转:}\ f_R(k,l) = f(K-l-1,k).$$

[0078] 在本文中,K是滤波器的尺寸,并且0≤k,l≤K-1是系数坐标,使得位置(0,0)在左上角,并且位置(K-1,K-1)在右下角。根据为该块计算的梯度值,将变换应用于滤波器系数f(k,l)。表1总结了变换和四个方向的四个梯度之间的关系。

[0079] 表1:为一个块计算的梯度和变换的映射

[0080]

梯度值	变换
$g_{d2}<g_{d1}$并且$g_h<g_v$	无变换

$g_{d2}<g_{d1}$并且$g_v<g_h$	对角线
$g_{d1}<g_{d2}$并且$g_h<g_v$	垂直翻转
$g_{d1}<g_{d2}$并且$g_v<g_h$	旋转

[0081] 3.1.3滤波器参数的信令通知

[0082] 在JEM中,为第一CTU信令通知GALF滤波器参数,即,在第一CTU的条带头之后并且在SAO参数之前。可以信令通知至多25个亮度滤波器系数集合。为了减少比特开销,可以合并不同分类的滤波器系数。此外,参考图片的GALF系数被存储并允许被重新用作当前图片的GALF系数。当前图片可以选择使用为参考图片存储的GALF系数,并且避开GALF系数信令。在这种情况下,仅信令通知参考图片中的一个的索引,并且为当前图片继承所指示的参考图片的存储的GALF系数。

[0083] 为了支持GALF时域预测,维护GALF滤波器集的候选列表。在解码新序列的开始,候选列表是空的。在解码一个图片之后,可以将对应的滤波器集合添加到候选列表。一旦候选列表的尺寸达到最大允许值(即,在当前JEM中为6),新的滤波器集合按解码顺序覆盖最旧的集合,即应用先进先出(FIFO)规则来更新候选列表。为了避免重复,只有当对应的图片不使用GALF时域预测时,才能将集合添加到列表。为了支持时域可伸缩性,有多个滤波器集的候选列表,并且每个候选列表与时域层相关联。更具体地,通过时域层索引(TempIdx)分配的每个阵列可以由具有等于更低TempIdx的先前解码的图片的滤波器集组成。例如,第k个阵列被分配为与等于k的TempIdx相关联,并且它仅包含来自TempIdx小于或等于k的图片的滤波器集。在对某个图片进行编解码之后,与图片相关联的滤波器集将用于更新与等于或高于TempIdx相关联的那些阵列。

[0084] GALF系数的时域预测用于帧间编解码的帧,以最小化信令开销。对于帧内帧,时域预测不可用,并且16个固定滤波器的集合被分配给每个类别。为了指示固定滤波器的使用,信令通知每个类别的标志,并且如果需要,还信令通知所选择的固定滤波器的索引。即使当为给定类别选择固定滤波器时,自适应滤波器的系数f(k,l)仍然可以为该类别进行传送,在这种情况下,将被应用于重构图像的滤波器的系数是两个系数集合的和。

[0085] 亮度分量的滤波过程可以在CU级别进行控制。信令通知标志来指示GALF是否被应用于CU的亮度分量。对于色度分量,仅在图片级别指示是否应用GALF。

[0086] 3.1.4滤波过程

[0087] 在解码器侧,当对块启用GALF时,对块内的每个样点R(i,j)进行滤波,产生如下所示的样点值R′(i,j),其中L表示滤波器长度,$f_{m,n}$表示滤波器系数,并且f(k,l)表示解码的滤波器系数。

[0088] $$R'(i,j)=\sum_{k=-L/2}^{L/2}\sum_{l=-L/2}^{L/2}f(k,l)\times R(i+k,j+l) \qquad (10)$$

[0089] 3.1.5编码器侧滤波器参数的确定过程

[0090] 在图3中示出了GALF的整个编码器决定过程。对于每个CU的亮度样点,编码器决定是否应用GALF,并且适当的信令标志被包括在条带头中。对于色度样点,应用滤波器的决定是基于图片级别而不是CU级别做出的。此外,仅当为图片启用亮度GALF时,才检查图片的色度GALF。

[0091] 4 VVC中的基于几何变换的自适应环路滤波器的示例

[0092] 与JEM中的相比,VVC中的GALF的当前设计具有以下主要改变:

[0093] 1)自适应滤波器形状被移除。针对亮度分量仅允许7×7滤波器形状,并且针对色度分量允许5×5滤波器形状。

[0094] 2)ALF参数的时域预测和来自固定滤波器的预测都被移除。

[0095] 3)对于每个CTU,一个比特标志被信令通知是启用还是禁用ALF。

[0096] 4)类别索引的计算是在4x4级别而不是2x2级别执行的。此外,如JVET-L0147中所建议的,利用了用于ALF分类的子采样拉普拉斯算子计算方法。更具体地,不需要计算一个块内的每个样点的水平/垂直/45对角线/135度梯度。相反,利用了1:2子采样。

[0097] 5 AVS2中的基于区域的自适应环路滤波器的示例

[0098] ALF是环路滤波的最后一个阶段。在该过程中有两个阶段。第一阶段是滤波器系数推导。为了训练滤波器系数,编码器将亮度分量的重构像素分类为16个区域,并且使用wiener-hopf等式为每个类别训练一个滤波器系数集合,以最小化原始帧和重构帧之间的均方误差。为了减少这16个滤波器系数集合之间的冗余,编码器将基于率失真性能来自适应地合并它们。最多可以为亮度分量分配16个不同的滤波器集,并且为色度分量仅分配一个滤波器集。第二阶段是滤波决定,包括帧级别和LCU级别两者。首先,编码器决定是否执行帧级别自适应环路滤波。如果帧级别ALF开启,则编码器进一步决定是否执行LCU级别ALF。

[0099] 5.1滤波器形状

[0100] 对于亮度和色度分量,在AVS-2中采用的滤波器形状是7×7十字形,叠加了3×3正方形,正如图5所示。图5中的每个正方形对应于样点。因此,总共17个样点用于推导位置C8的样点的滤波值。考虑到发送系数的开销,利用点对称滤波器,其中只剩下九个系数{C0,C1,...,C8},这将滤波器系数的数量以及滤波中的乘法次数减少为一半。点对称滤波器还可以减少一个滤波样点的计算的一半,例如一个滤波样点仅9次乘法和14次加法运算。

[0101] 5.2基于区域的自适应合并

[0102] 为了适应不同的编解码误差,AVS-2对亮度分量采用基于区域的多重自适应环路滤波器。亮度分量被划分为16个尺寸大致相等的基本区域,其中每个基本区域与最大编解码单元(LCU)边界对齐,如图6所示,并且为每个区域推导一个维纳(Wiener)滤波器。使用的滤波器越多,减少的失真越多,但是用于编码这些系数的比特随着滤波器的数量而增加。为了实现最佳的率失真性能,可以将这些区域合并为更少的更大区域,这些区域共享相同的滤波器系数。为了简化合并过程,根据基于图像先验相关性的修改的希尔伯特顺序,向每个区域分配索引。具有连续索引的两个区域可以基于率失真代价而合并。

[0103] 区域之间的映射信息应该被信令通知给解码器。在AVS-2中,基本区域的数量用于表示合并结果,并且滤波器系数根据其区域顺序而顺序地压缩。例如,当{0,1}、{2,3,4}、{5,6,7,8,9}分别和左侧基本区域合并为一个区域时,仅三个整数被编解码来表示该合并映射,即2、3、5。

[0104] 5.3边信息的信令通知

[0105] 还使用了多个切换标志。序列切换标志adaptive_loop_filter_enable用于控制自适应环路滤波器是否被应用于整个序列。图像切换标志picture_alf_enble[i]控制ALF是否被应用于对应的第i个图像分量。仅当启用picture_alf_enble[i]时,才会发送该色彩分量的对应LCU级别标志和滤波器系数。LCU级别标志lcu_alf_enable[k],控制是否对对应

的第k个LCU启用ALF,并且被交织到条带数据中。不同级别调整标志的决定都基于率失真代价。高度的灵活性进一步使得ALF更加显著地提高编解码效率。

[0106] 在一些实施例中,对于亮度分量,可以有至多16个滤波器系数集合。

[0107] 在一些实施例中,对于每个色度分量(Cb和Cr),可以发送一个滤波器系数集合。

[0108] 6VTM-4中的GALF

[0109] 在VTM4.0中,自适应环路滤波器的滤波过程被执行如下:

[0110] $O(x,y)=\sum_{(i,j)}w(i,j).I(x+i,y+j)$ (11)

[0111] 其中样点I(x+i,y+j)是输入样点,O(x,y)是滤波后的输出样点(即,滤波器结果),并且w(i,j)表示滤波器系数。实际上,在VTM4.0中,它是使用定点精度计算的整数算法实施的:

[0112] $$O(x,y)=\left(\sum_{i=-\frac{L}{2}}^{\frac{L}{2}}\sum_{j=-\frac{L}{2}}^{\frac{L}{2}}w(i,j).I(x+i,y+j)+64\right)\gg 7 \quad (12)$$

[0113] 其中,L表示滤波器长度,并且其中w(i,j)是定点精度的滤波器系数。

[0114] 7非线性自适应环路滤波(ALF)

[0115] 7.1滤波再形成(reformulation)

[0116] 等式(11)可以在不影响编解码效率的情况下再形成为以下表达式:

[0117] $O(x,y)=I(x,y)+\sum_{(i,j)\neq(0,0)}w(i,j).(I(x+i,y+j)-I(x,y))$ (13)

[0118] 在本文中,w(i,j)是与等式(11)中的滤波器系数相同的滤波器系数[除了在等式(13)中等于1而在等式(11)中等于$1-\sum_{(i,j)\neq(0,0)}w(i,j)$的w(0,0)]。

[0119] 7.2修改的滤波器

[0120] 使用上面的滤波器公式(13),我们可以很容易地引入非线性,以通过使用简单的裁剪函数降低临近样点值I(x+i,y+j))的影响(当它们与正被滤波的当前样点值I(x,y)太不同时)来使得ALF更有效。

[0121] 在该建议中,ALF滤波器被修改如下:

[0122] $O'(x,y)=I(x,y)+\sum_{(i,j)\neq(0,0)}w(i,j).K(I(x+i,y+j)-I(x,y),k(i,j))$ (14)

[0123] 在本文中,K(d,b)=min(b,max(-b,d))是裁剪函数,并且k(i,j)是取决于(i,j)滤波器系数的裁剪参数。编码器执行优化以找到最佳k(i,j)。

[0124] 在JVET-N0242实施方式中,为每个ALF滤波器指定裁剪参数k(i,j),每个滤波器系数信令通知一个裁剪值。这意味着对于每个亮度滤波器至多可以在比特流中信令通知12个裁剪值,并且对于色度滤波器至多可以信令通知6个裁剪值。

[0125] 为了限制信令开销和编码器复杂度,我们将裁剪值的估计限制在小的可能值集合内。在建议中,我们仅使用了4个固定值,其对于帧间和帧内片组是相同的。

[0126] 因为亮度的局部差的方差通常高于色度,所以我们对亮度和色度滤波器使用两个不同的集合。我们还在每个集合中包括最大样点值(这里10比特的比特深度为1024),因此如果不需要,可以禁用裁剪。

[0127] 在JVET-N0242测试中使用的裁剪值集合在表2中提供。这4个值已经通过在对数域中大致相等地划分亮度的样点值的整个范围(在10比特上编解码)和色度的范围(从4到1024)而选择。

[0128] 更准确地说,裁剪值的亮度表是通过以下公式获得的:

[0129] $\mathrm{AlfClip_L}=\left\{\mathrm{round}\left(\left((M)^{\frac{1}{N}}\right)^{N-n+1}\right)\ 其中\ n\in\ 1..N\right]\right\}$,其中M=$2^{10}$并且N=4。

[0130] 类似地,裁剪值的色度表是根据以下公式获得的:

[0131] $\mathrm{AlfClip_C}=\left\{\mathrm{round}\left(A.\left(\left(\frac{M}{A}\right)^{\frac{1}{N-1}}\right)^{N-n}\right)\ 其中\ n\in\ 1..N\right]\right\}$,其中M=$2^{10}$,N=4并且A=4。

[0132] 表2:授权裁剪值

[0133]

	帧内/帧间片组
亮度	{1024,181,32,6}
色度	{1024,161,25,4}

[0134] 通过使用对应于上表2中的裁剪值的索引的Golomb编码方案,在“alf_data”语法元素中对所选择的裁剪值进行编解码。该编码方案与滤波器索引的编码方案相同。

[0135] 8 JVET-N0415中基于CTU的ALF

[0136] 条带级别时域滤波器。在VTM4中采用自适应参数集(APS)。每个APS包含一个信令通知的ALF滤波器的集合,至多支持32个APS。在该建议中,对条带级别时域滤波器进行测试。片组可以重新使用来自APS的ALF信息,以减少开销。APS作为先进先出(FIFO)缓冲区来更新。

[0137] 基于CTB的ALF。对于亮度分量,当ALF被应用于亮度CTB时,16个固定的、5个时域的或1个信令通知的滤波器集当中的选择被指示。仅滤波器集索引被信令通知。对于一个条带,仅可以信令通知一个新的25个滤波器的集合。如果为条带信令通知新的集合,则相同条带中的所有亮度CTB共享该集合。固定滤波器集可以用于预测新的条带级别滤波器集,并且也可以用作亮度CTB的候选滤波器集。滤波器的数量总共为64。

[0138] 对于色度分量,当ALF被应用于色度CTB时,如果为条带信令通知新的滤波器,则CTB使用新的滤波器,否则,应用满足时域可伸缩性约束的最近的时域色度滤波器。

[0139] 作为条带级别时域滤波器,APS作为先进先出(FIFO)缓冲区来更新。

[0140] 规范。

[0141] 以下文本基于JVET-K1001-v6使用{{fixed filter}}({{固定滤波器}})、[[temporal filters]]([[时域滤波器]])和((CTB-based filter index))(((基于CTB的滤波器索引)))(即,使用双花括号、双方括号和双圆括号)修改

[0142] 7.3.3.2自适应环路滤波器数据语法

[0143]

alf_data( ) {	描述符
alf_chroma_idc	tu(v)
((alf_signal_new_filter_luma	u(1)
if(alf_signal_new_filter_luma > 0) { **))**	
{{ alf_luma_use_fixed_filter_flag	u(1)
if(alf_luma_use_fixed_filter_flag){	
alf_luma_fixed_filter_set_index	tb(v)
alf_luma_fixed_filter_usage_pattern	u(1)
if (alf_luma_fixed_filter_usage_pattern > 0)	
for (i = 0; i < NumAlfFilters; i++)	
alf_luma_fixed_filter_usage[i]	u(1) **}}**
((alf_num_available_temporal_filter_sets_luma	tb(1) **))**
alf_luma_num_filters_signalled_minus1	tb(v)
alf_luma_type_flag	u(1)
if(alf_luma_num_filters_signalled_minus1 > 0) {	
for(filtIdx = 0; filtIdx < NumAlfFilters; filtIdx++)	
alf_luma_coeff_delta_idx[filtIdx]	tu(v)
}	
alf_luma_coeff_delta_flag	u(1)
if (!alf_luma_coeff_delta_flag && alf_luma_num_filters_signalled_minus1 > 0)	
alf_luma_coeff_delta_prediction_flag	u(1)
alf_luma_min_eg_order_minus1	tu(v)
for(i = 0; i < (alf_luma_type_flag == 1) ? 2 : 3; i++)	
alf_luma_eg_order_increase_flag[i]	u(1)

[0144]

if (alf_luma_coeff_delta_flag) {	
for(sigFiltIdx = 0; sigFiltIdx <= alf_luma_num_filters_signalled_minus1; sigFiltIdx++)	
alf_luma_coeff_flag[sigFiltIdx]	u(1)
}	
for(sigFiltIdx = 0; sigFiltIdx <= alf_luma_num_filters_signalled_minus1; sigFiltIdx++){	
if (alf_luma_coeff_flag[sigFiltIdx]) {	
for (j = 0; j < (alf_luma_type_flag == 1) ? 6 : 12; j++) {	
alf_luma_coeff_delta_abs[sigFiltIdx][j]	uek(v)
if(alf_luma_coeff_delta_abs[sigFiltIdx][j])	
alf_luma_coeff_delta_sign[sigFiltIdx][j]	u(1)
}	
}	
}	
((}))	
if (alf_chroma_idc > 0) {	
((**alf_signal_new_filter_chroma**	u(1)
if (alf_signal_new_filter_chroma){))	
alf_chroma_min_eg_order_minus1	tu(v)
for(i = 0; i < 2; i++)	
alf_chroma_eg_order_increase_flag[i]	u(1)
for(j = 0; j < 6; j++) {	
alf_chroma_coeff_abs[j]	uek(v)
if(alf_chroma_coeff_abs[j] > 0)	
alf_chroma_coeff_sign[j]	u(1)
}	
((}))	
}	
}	

[0145] 7.3.4.2编解码树单元语法

[0146]

coding_tree_unit( ) {	**描述符**
xCtb = (CtbAddrInRs % PicWidthInCtbsY) << CtbLog2SizeY	
yCtb = (CtbAddrInRs / PicWidthInCtbsY) << CtbLog2SizeY	
if(slice_alf_enable_flag){	
alf_ctb_flag[0][xCtb >> Log2CtbSize][yCtb >> Log2CtbSize]	ae(v)
if (alf_ctb_flag[0][xCtb >> Log2CtbSize][yCtb >> Log2CtbSize] {	
((coding_alf_ctu_filter_set_index(CtbAddrInRs)))	
}	
if(alf_chroma_idc == 1 || alf_chroma_idc == 3)	
alf_ctb_flag[1][xCtb >> Log2CtbSize][yCtb >> Log2CtbSize]	ae(v)
if(alf_chroma_idc == 2 || alf_chroma_idc == 3)	
alf_ctb_flag[2][xCtb >> Log2CtbSize][yCtb >> Log2CtbSize]	ae(v)
}	
if(slice_type = = I && qtbtt_dual_tree_intra_flag) {	
dual_tree_implicit_qt_split (xCtb, yCtb, CtbLog2SizeY, 0)	
else	
coding_quadtree(xCtb, yCtb, CtbLog2SizeY, 0, SINGLE_TREE)	
}	

[0147]

((coding_alf_ctu_filter_set_index(CtbAddrInRs, slice_type){))	描述符
xCtb = (CtbAddrInRs % PicWidthInCtbsY) << CtbLog2SizeY	
yCtb = (CtbAddrInRs / PicWidthInCtbsY) << CtbLog2SizeY	
alf_use_new_filter	ae(1)
if (alf_use_new_filter == 0){	
alf_use_fixed_filter	ae(1)
}	
if (alf_use_new_filter){	
alf_luma_ctb_filter_set_index[xCtb >> Log2CtbSize][yCtb >> Log2CtbSize] = 16	
}	
else if (alf_use_fixed_filter){	
alf_fixed_filter_index	tb(v)
alf_luma_ctb_filter_set_index[xCtb >> Log2CtbSize][yCtb >> Log2CtbSize] = alf_fixed_filter_index	
}	
else{	
alf_temporal_index	tb(v)
alf_luma_ctb_filter_set_index[xCtb >> Log2CtbSize][yCtb >> Log2CtbSize] = 16 + alf_temporal_index	
}	

[0148] 7.4.4.2自适应环路滤波器数据语义

[0149] 等于1的((alf_signal_new_filter_luma))指定信令通知新的亮度滤波器集。等于0的alf_signal_new_filter_luma指定不信令通知新的亮度滤波器集。当不存在时为0。

[0150] 等于1的{{alf_luma_use_fixed_filter_flag}}指定固定滤波器集用于信令通知自适应环路滤波器。等于0的alf_luma_use_fixed_filter_flag指定固定滤波器集不用于信令通知自适应环路滤波器。

[0151] {{alf_luma_fixed_filter_set_index}}指定固定的滤波器集索引。它可以为0…15。

[0152] 等于0的{{alf_luma_fixed_filter_usage_pattern}}指定所有新的滤波器都使用固定滤波器。等于1的alf_luma_fixed_filter_usage_pattern指定新的滤波器中的一些使用固定滤波器,而其他不使用。

[0153] 等于1的{{alf_luma_fixed_filter_usage[i]}}指定第i个滤波器使用固定滤波器。等于0的alf_luma_fixed_filter_usage[i]指定第i个滤波器不使用固定滤波器。当它不存在时,其被推断为1。

[0154] 等于1的((alf_signal_new_filter_chroma))指定信令通知新的色度滤波器。等于0的alf_signal_new_filter_chroma指定不信令通知新的色度滤波器。

[0155] ((alf_num_available_temporal_filter_sets_luma))指定可以用于当前条带的可用时域滤波器集的数量,它可以来自0..5。当不存在时为0。

[0156] 变量alf_num_available_filter_sets被推导为16+alf_signal_new_filter_luma+alf_num_available_temporal_filter_sets_luma。

[0157] ((如果alf_signal_new_filter_luma为1,则进行以下过程))

[0158] 变量filterCoefficients[sigFiltIdx][j](其中sigFiltIdx=0..alf_luma_num_filters_signalled_minus1,j=0..11)被初始化如下:

[0159] filterCoefficients[sigFiltIdx][j]＝alf_luma_coeff_delta_abs[sigFiltIdx][j]*(7-50)

[0160] (1-2*alf_luma_coeff_delta_sign[sigFiltIdx][j])

[0161] 当alf_luma_coeff_delta_prediction_flag等于1时,filterCoefficients[sigFiltIdx][j](其中sigFiltIdx＝1..alf_luma_num_filters_signalled_minus1并且j＝0..11)被修改如下:

[0162] filterCoefficients[sigFiltIdx][j]+＝filterCoefficients[sigFiltIdx-1][j] (7-51)

[0163] 亮度滤波器系数$AlfCoeff_L$(元素为$AlfCoeff_L$[filtIdx][j],其中filtIdx＝0..NumAlfFilters-1并且j＝0..11)被推导如下

[0164] $AlfCoeff_L$[filtIdx][j]＝filterCoefficients[alf_luma_coeff_delta_idx[filtIdx]][j] (7-52)

[0165] {{如果alf_luma_use_fixed_filter_flag为1,并且alf_luma_fixed_filter_usage[filtidx]为1,则以下适用:

[0166] $AlfCoeff_L$[filtIdx][j]＝$AlfCoeff_L$[filtIdx][j]+AlfFixedFilterCoeff[AlfClassToFilterMapping[alf_luma_fixed_filter_index][filtidx]][j]}}

[0167] 最后的滤波器系数$AlfCoeff_L$[filtIdx][12],其中filtIdx＝0..NumAlfFilters-1被推导如下:$AlfCoeff_L$[filtIdx][12]＝128-Σ_k($AlfCoeff_L$[filtIdx][k]<<1),其中k＝0..11(7-53)

[0168] 比特流一致性的要求是$AlfCoeff_L$[filtIdx][j](其中filtIdx＝0..NumAlfFilters-1,j＝0..11)的值应当在-2^7至2^7-1(包括-2^7和2^7-1)的范围内,并且$AlfCoeff_L$[filtIdx][12]的值应当在0至2^8-1(包括0 to 2^8-1)的范围内。

[0169] ((亮度滤波器系数))$AlfCoeff_{LumaAll}$(元素为$AlfCoeff_{LumaAll}$[filtSetIdx][filtIdx][j],其中filtSetIdx＝0..15,filtSetIdx＝0..NumAlfFilters-1并且j＝0..12)被推导如下

[0170] $AlfCoeff_{LumaAll}$[filtSetIdx][filtIdx][j]＝{{AlfFixedFilterCoeff[AlfClassToFilterMapping[}}filtSetIdx{{][filtidx]][j]}}

[0171] ((亮度滤波器系数))$AlfCoeff_{LumaAll}$(元素为$AlfCoeff_{LumaAll}$[filtSetIdx][filtIdx][j],其中filtSetIdx＝16,filtSetIdx＝0..NumAlfFilters-1并且j＝0..12)被推导如下

[0172] 变量closest_temporal_index被初始化为-1。Tid是当前条带的时域层索引。((如果alf_signal_new_filter_luma为1))

[0173] $AlfCoeff_{LumaAll}$[16][filtIdx][j]＝$AlfCoeff_L$[filtIdx][j]

[0174] ((否则,调用以下过程))

```
for (i = Tid; i >= 0; i--)
{
    for (k = 0; k < temp_size_L; k++)
    {
        if (temp_Tid_L[ k ] == i)
        {
            closest_temporal_index被设置为k;
            break;
        }
        }
}
```

[0175]

[0176] $AlfCoeff_{LumaAll}$[16][filtIdx][j]＝$Temp_L$[closest_temporal_index][filtIdx][j]((亮度滤波器系数))$AlfCoeff_{LumaAll}$(元素为$AlfCoeff_{LumaAll}$[filtSetIdx][filtIdx][j],其中filtSetIdx＝17..alf_num_available_filter_sets-1,filtSetIdx＝0..NumAlfFilters-1并且j＝0..12)被推导如下

```
i = 17;
  for (k = 0; k < temp_size_L并且i < alf_num_available_filter_sets; j++)
  {
    if (temp_Tid_L[ k ] <= Tid并且k不等于closest_temporal_index)
    {
      AlfCoeff_LumaAll[ i ][ filtIdx ][ j ] = Temp_L[ k ][ filtIdx ][ j ];
          i++;
    }
    }
```

[0177]

```
{{ AlfFixedFilterCoeff }}[64][13] =
{
  { 0,   0,   2,  -3,   1,  -4,   1,   7,  -1,   1,  -1,   5, 112 },
  { 0,   0,   0,   0,   0,  -1,   0,   1,   0,   0,  -1,   2, 126 },
  { 0,   0,   0,   0,   0,   0,   0,   1,   0,   0,   0,   0, 126 },
  { 0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  -1,   1, 128 },
  { 2,   2,  -7,  -3,   0,  -5,  13,  22,  12,  -3,  -3,  17,  34 },
  {-1,   0,   6,  -8,   1,  -5,   1,  23,   0,   2,  -5,  10,  80 },
  { 0,   0,  -1,  -1,   0,  -1,   2,   1,   0,   0,  -1,   4, 122 },
  { 0,   0,   3, -11,   1,   0,  -1,  35,   5,   2,  -9,   9,  60 },
  { 0,   0,   8,  -8,  -2,  -7,   4,   4,   2,   1,  -1,  25,  76 },
  { 0,   0,   1,  -1,   0,  -3,   1,   3,  -1,   1,  -1,   3, 122 },
  { 0,   0,   3,  -3,   0,  -6,   5,  -1,   2,   1,  -4,  21,  92 },
  {-7,   1,   5,   4,  -3,   5,  11,  13,  12,  -8,  11,  12,  16 },
  {-5,  -3,   6,  -2,  -3,   8,  14,  15,   2,  -7,  11,  16,  24 },
  { 2,  -1,  -6,  -5,  -2,  -2,  20,  14,  -4,   0,  -3,  25,  52 },
  { 3,   1,  -8,  -4,   0,  -8,  22,   5,  -3,   2, -10,  29,  70 },
  { 2,   1,  -7,  -1,   2, -11,  23,  -5,   0,   2, -10,  29,  78 },
  {-6,  -3,   8,   9,  -4,   8,   9,   7,  14,  -2,   8,   9,  14 },
  { 2,   1,  -4,  -7,   0,  -8,  17,  22,   1,  -1,  -4,  23,  44 },
  { 3,   0,  -5,  -7,   0,  -7,  15,  18,  -5,   0,  -5,  27,  60 },
  { 2,   0,   0,  -7,   1, -10,  13,  13,  -4,   2,  -7,  24,  74 },
  { 3,   3, -13,   4,  -2,  -5,   9,  21,  25,  -2,  -3,  12,  24 },
  {-5,  -2,   7,  -3,  -7,   9,   8,   9,  16,  -2,  15,  12,  14 },
  { 0,  -1,   0,  -7,  -5,   4,  11,  11,   8,  -6,  12,  21,  32 },
  { 3,  -2,  -3,  -8,  -4,  -1,  16,  15,  -2,  -3,   3,  26,  48 },
  { 2,   1,  -5,  -4,  -1,  -8,  16,   4,  -2,   1,  -7,  33,  68 },
  { 2,   1,  -4,  -2,   1, -10,  17,  -2,   0,   2, -11,  33,  74 },
  { 1,  -2,   7, -15, -16,  10,   8,   8,  20,  11,  14,  11,  14 },
  { 2,   2,   3, -13, -13,   4,   8,  12,   2,  -3,  16,  24,  40 },
  { 1,   4,   0,  -7,  -8,  -4,   9,   9,  -2,  -2,   8,  29,  54 },
  { 1,   1,   2,  -4,  -1,  -6,   6,   3,  -1,  -1,  -3,  30,  74 },
  {-7,   3,   2,  10,  -2,   3,   7,  11,  19,  -7,   8,  10,  14 },
  { 0,  -2,  -5,  -3,  -2,   4,  20,  15,  -1,  -3,  -1,  22,  40 },
  { 3,  -1,  -8,  -4,  -1,  -4,  22,   8,  -4,   2,  -8,  28,  62 },
  { 0,   3, -14,   3,   0,   1,  19,  17,   8,  -3,  -7,  20,  34 },
```

[0178]

```
    { 0,   2,  -1,  -8,   3,  -6,   5,  21,   1,   1,  -9,  13,  84 },
    {-4,  -2,   8,  20,  -2,   2,   3,   5,  21,   4,   6,   1,   4 },
    { 2,  -2,  -3,  -9,  -4,   2,  14,  16,   3,  -6,   8,  24,  38 },
    { 2,   1,   5, -16,  -7,   2,   3,  11,  15,  -3,  11,  22,  36 },
    { 1,   2,   3, -11,  -2,  -5,   4,   8,   9,  -3,  -2,  26,  68 },
    { 0,  -1,  10,  -9,  -1,  -8,   2,   3,   4,   0,   0,  29,  70 },
    { 1,   2,   0,  -5,   1,  -9,   9,   3,   0,   1,  -7,  20,  96 },
    {-2,   8,  -6,  -4,   3,  -9,  -8,  45,  14,   2, -13,   7,  54 },
    { 1,  -1,  16, -19,  -8,  -4,  -3,   2,  19,   0,   4,  30,  54 },
    { 1,   1,  -3,   0,   2, -11,  15,  -5,   1,   2,  -9,  24,  92 },
    { 0,   1,  -2,   0,   1,  -4,   4,   0,   0,   1,  -4,   7, 120 },
    { 0,   1,   2,  -5,   1,  -6,   4,  10,  -2,   1,  -4,  10, 104 },
    { 3,   0,  -3,  -6,  -2,  -6,  14,   8,  -1,  -1,  -3,  31,  60 },
    { 0,   1,   0,  -2,   1,  -6,   5,   1,   0,   1,  -5,  13, 110 },
    { 3,   1,   9, -19, -21,   9,   7,   6,  13,   5,  15,  21,  30 },
    { 2,   4,   3, -12, -13,   1,   7,   8,   3,   0,  12,  26,  46 },
    { 3,   1,  -8,  -2,   0,  -6,  18,   2,  -2,   3, -10,  23,  84 },
    { 1,   1,  -4,  -1,   1,  -5,   8,   1,  -1,   2,  -5,  10, 112 },
    { 0,   1,  -1,   0,   0,  -2,   2,   0,   0,   1,  -2,   3, 124 },
    { 1,   1,  -2,  -7,   1,  -7,  14,  18,   0,   0,  -7,  21,  62 },
    { 0,   1,   0,  -2,   0,  -7,   8,   1,  -2,   0,  -3,  24,  88 },
    { 0,   1,   1,  -2,   2, -10,  10,   0,  -2,   1,  -7,  23,  94 },
    { 0,   2,   2, -11,   2,  -4,  -3,  39,   7,   1, -10,   9,  60 },
    { 1,   0,  13, -16,  -5,  -6,  -1,   8,   6,   0,   6,  29,  58 },
    { 1,   3,   1,  -6,  -4,  -7,   9,   6,  -3,  -2,   3,  33,  60 },
    { 4,   0, -17,  -1,  -1,   5,  26,   8,  -2,   3, -15,  30,  48 },
    { 0,   1,  -2,   0,   2,  -8,  12,  -6,   1,   1,  -6,  16, 106 },
    { 0,   0,   0,  -1,   1,  -4,   4,   0,   0,   0,  -3,  11, 112 },
    { 0,   1,   2,  -8,   2,  -6,   5,  15,   0,   2,  -7,   9,  98 },
    { 1,  -1,  12, -15,  -7,  -2,   3,   6,   6,  -1,   7,  30,  50 },
};

{{ AlfClassToFilterMapping }}[ 16 ][ 25 ] =
{
    { 8,   2,   2,   2,   3,   4,  53,   9,   9,  52,   4,   4,   5,   9,   2,   8,  10,
9,   1,   3,  39,  39,  10,   9,  52 },
    { 11,  12,  13,  14,  15,  30,  11,  17,  18,  19,  16,  20,  20,   4,  53,  21,
22,  23,  14,  25,  26,  26,  27,  28,  10 },
    { 16,  12,  31,  32,  14,  16,  30,  33,  53,  34,  35,  16,  20,   4,   7,  16,
21,  36,  18,  19,  21,  26,  37,  38,  39 },
    { 35,  11,  13,  14,  43,  35,  16,   4,  34,  62,  35,  35,  30,  56,   7,  35,
21,  38,  24,  40,  16,  21,  48,  57,  39 },
    { 11,  31,  32,  43,  44,  16,   4,  17,  34,  45,  30,  20,  20,   7,   5,  21,
22,  46,  40,  47,  26,  48,  63,  58,  10 },
    { 12,  13,  50,  51,  52,  11,  17,  53,  45,   9,  30,   4,  53,  19,   0,  22,
23,  25,  43,  44,  37,  27,  28,  10,  55 },
    { 30,  33,  62,  51,  44,  20,  41,  56,  34,  45,  20,  41,  41,  56,   5,  30,
56,  38,  40,  47,  11,  37,  42,  57,   8 },
    { 35,  11,  23,  32,  14,  35,  20,   4,  17,  18,  21,  20,  20,  20,   4,  16,
21,  36,  46,  25,  41,  26,  48,  49,  58 },
    { 12,  31,  59,  59,   3,  33,  33,  59,  59,  52,   4,  33,  17,  59,  55,  22,
36,  59,  59,  60,  22,  36,  59,  25,  55 },
```

[0179]
```
{ 31, 25, 15, 60, 60, 22, 17, 19, 55, 55, 20, 20, 53, 19, 55, 22,
46, 25, 43, 60, 37, 28, 10, 55, 52 },
{ 12, 31, 32, 50, 51, 11, 33, 53, 19, 45, 16, 4, 4, 53, 5, 22,
36, 18, 25, 43, 26, 27, 27, 28, 10 },
{ 5, 2, 44, 52, 3, 4, 53, 45, 9, 3, 4, 56, 5, 0, 2, 5, 10,
47, 52, 3, 63, 39, 10, 9, 52 },
{ 12, 34, 44, 44, 3, 56, 56, 62, 45, 9, 56, 56, 7, 5, 0, 22,
38, 40, 47, 52, 48, 57, 39, 10, 9 },
{ 35, 11, 23, 14, 51, 35, 20, 41, 56, 62, 16, 20, 41, 56, 7, 16,
21, 38, 24, 40, 26, 26, 42, 57, 39 },
{ 33, 34, 51, 51, 52, 41, 41, 34, 62, 0, 41, 41, 56, 7, 5, 56,
38, 38, 40, 44, 37, 42, 57, 39, 10 },
{ 16, 31, 32, 15, 60, 30, 4, 17, 19, 25, 22, 20, 4, 53, 19, 21,
22, 46, 25, 55, 26, 48, 63, 58, 55 }, };
```

[0180] ((如果alf_signal_new_filter_chroma为1,则进行以下过程))

[0181] 色度滤波器系数$AlfCoeff_C[j]$(其中j=0..5)被推导如下:

[0182] $AlfCoeff_C[j]=alf_chroma_coeff_abs[j]*(1-2*alf_chroma_coeff_sign[j])$ (7-57)

[0183] 对于j=6的最后一个滤波器系数被推导如下:

[0184] $AlfCoeff_C[6]=128-\sum_k(AlfCoeff_C[k]<<1)$,其中k=0..5 (7-58)

[0185] 比特流一致性的要求是$AlfCoeff_C[j]$(其中j=0..5)的值应当在-2^7-1至2^7-1(包括-2^7-1和2^7-1)的范围内,并且$AlfCoeff_C[6]$的值应当在0至2^8-1(包括0和2^8-1)的范围内。

[0186] 否则(((alf_signal_new_filter_chroma为0))),以下被调用

[0187]
```
for (i = Tid; i >= 0; i--)
{
    for (k = 0; k < temp_size_C; k++)
    {
        if (temp_Tid_C[ k ] == i)
        {
            closest_temporal_index被设置为k;
            break;
        }
        }
}
```

[0188] 色度滤波器系数$AlfCoeff_C[j]$(其中j=0..6)被推导如下:

[0189] $AlfCoeff_C[j]=Temp_C[closest_temporal_index][j]$

[0190] 7.4.5.2编解码树单元语义

[0191] ((alf_luma_ctb_filter_set_index[xCtb>>Log2CtbSize][yCtb>>Log2CtbSize]))指定位置(xCtb,yCtb)处的亮度CTB的滤波器集索引。

[0192] ((alf_use_new_filter))等于1指定alf_luma_ctb_filter_set_index[xCtb>>Log2CtbSize][yCtb>>Log2CtbSize]为16。alf_use_new_filter等于0指定alf_luma_ctb_filter_set_index[xCtb>>Log2CtbSize][yCtb>>Log2CtbSize]不为16。

[0193] ((alf_use_fixed_filter))等于1指定使用固定滤波器集中的一个。alf_use_fixed_filter等于0指定当前亮度CTB不使用任何固定滤波器集。

[0194] ((alf_fixed_filter_index))指定固定滤波器集索引,其可以从0到15。

[0195] ((alf_temporal_index))指定时域滤波器集索引,其可以从0到alf_num_available_temporal_filter_sets_luma-1。

[0196] [[8.5.1通用]]

[0197] 1.当sps_alf_enabled_flag等于1时,以下适用:

[0198] -[[调用在第8.5.4.5节中指定的时域滤波器刷新过程。]]

[0199] -以重构图片样点阵列S_L、S_{Cb}和S_{Cr}作为输入调用在第8.5.4.1节中指定的自适应环路滤波器过程,以样点自适应偏移之后的修改的重构图片样点阵列S'_L、S'_{Cb}和S'_{Cr}作为输出。

[0200] -阵列S'_L、S'_{Cb}和S'_{Cr}分别被分配给阵列S_L、S_{Cb}和S_{Cr}(表示解码的图片)。

[0201] -[[调用在条款8.5.4.6中指定的时域滤波器更新过程。]]((8.5.4.2亮度样点的编解码树块滤波过程))

[0202] -对应于由filtIdx[x][y]指定的滤波器的亮度滤波器系数f[j]的阵列被推导如下,其中j=0..12:

[0203] f[j]=((AlfCoeff$_{LumaAll}$))[alf_luma_ctb_filter_set_index[xCtb>>Log2CtbSize][yCtb>>L$_{og}$2CtbSize]]][filtIdx[x][y]][j] (8-732)

[0204] [[8.5.4.5时域滤波器刷新]]

[0205] 如果以下条件中的任何一个为真,

[0206] -当前图片是IDR图片

[0207] -当前图片是BLA图片

[0208] -按照解码顺序,当前图片是POC大于最后一个解码的IRAP图片的POC的第一个图片,即在前导图片(leading picture)之后并在拖尾图片(trailing picture)之前。

[0209] 然后temp_size_L和temp_size_C被设置为0。

[0210] [[8.5.4.6时域滤波器更新]]

[0211] 如果slice_alf_enabled_flag为1,并且alf_signal_new_filter_luma为1,则以下适用。

[0212] 如果亮度时域滤波器缓冲区尺寸temp_size_L<5,则temp_size_L=temp_size_L+1。

[0213] $Temp_L$[i][j][k](其中i=temp_size_L-1…1,j=0…NumAlfFilters-1并且k=0…12)被更新为:

[0214] $Temp_L$[i][j][k]=$Temp_L$[i-1][j][k]

[0215] $Temp_L$[0][j][k](其中j=0…NumAlfFilters-1并且k=0..12)被更新为

[0216] $Temp_L$[0][j][k]=$AlfCoeff_L$[j][k]

[0217] $Temp_{Tid_L}$[i](其中i=temp_size_L-1…1)被更新为

[0218] $Temp_{Tid_L}$[i]=$Temp_{Tid_L}$[i-1]

[0219] $Temp_{Tid_L}$[0]被设置为当前条带的时域层索引Tid

[0220] 如果alf_chroma_idx不为0,并且alf_signal_new_filter_chroma为1,则以下适用

[0221] $Temp_c$[i][j](其中i=temp_size_c-1…1并且j=0…6)被更新为:

[0222] $Temp_c$[i][j]=$Temp_c$[i-1][j]

[0223] $Temp_C$[0][j](其中j＝0…6)被更新为

[0224] $Temp_C$[0][j]＝$AlfCoeff_C$[j]

[0225] $Temp_{Tid_C}$[i](其中i＝temp_size_C-1…1)被更新为

[0226] $Temp_{Tid_C}$[i]＝$Temp_{Tid_C}$[i-1]

[0227] $Temp_{Tid_C}$[0]被设置为当前条带的Tid

[0228] 表9-4-语法元素和相关联的二值化

[0229]

语法结构	语法元素	二值化	
		过程	输入参数
coding_tree_unit( )	alf_ctb_flag[][][]	FL	cMax = 1
	((alf_use_new_filter_set	FL	cMax = 1
	alf_used_fixed_filter	FL	cMax = 1
	alf_fixed_filter_index	TB	cMax = 15
	alf_temporal_index	TB	cMax = alf_num_available_temporal_filter_sets_luma - 1))

[0230] 表9-10-ctxInc到具有上下文编解码二进制位的语法元素的分配

[0231]

语法元素	binIdx					
	0	1	2	3	4	>= 5
((alf_use_new_filter_set	0	na	na	na	na	na
alf_used_fixed_filter	0	na	na	na	na	na))

[0232] 9 JVET-M0427中的环路内重整形(ILR)

[0233] 环路内重整形(ILR)的基本思想是将(第一域中的)原始信号(预测/重构信号)转换到第二域(重整形域)。

[0234] 环路内亮度重整形器被实施为一对查找表(LUT),但是仅两个LUT中的一个需要被信令通知,因为另一个可以根据信令通知的LUT进行计算。每个LUT都是一维、10比特、1024条目的映射表(1D-LUT)。一个LUT是前向LUT,FwdLUT,它将输入亮度代码值Y_i映射到更改的值Y_r:Y_r＝FwdLUT[Y_i]。另一个LUT是逆LUT,InvLUT,它将更改的代码值Y_r映射到$\hat{Y}_i$:$\hat{Y}_i = InvLUT[Y_r]$。($\hat{Y}_i$表示Y_i的重构值。)。

[0235] 9.1 PWL模型

[0236] 从概念上讲,分段线性(Piece-Wise Linear,PWL)是以以下方式实施的:

[0237] 假设x1、x2是两个输入枢轴点(pivot point),并且y1、y2是它们的一段的对应的输出枢轴点。x1和x2之间的任何输入值x的输出值y可以通过以下等式进行插值:

[0238] y＝((y2-y1)/(x2-x1))*(x-x1)+y1

[0239] 在定点实施方式中,等式可以被重写为:

[0240] y＝((m*x+2FP_PREC-1)>>FP_PREC)+c

[0241] 这里,m是标量,c是偏移,并且FP_PREC是指定精度的常数值。

[0242] 请注意,在CE-12软件中,PWL模型用于预先计算1024条目FwdLUT和InvLUT映射表;但是PWL模型也允许实施方式在不预先计算LUT的情况下在运行中(on-the-fly)计算相同的映射值。

[0243] 9.2在第四次VVC会议中测试CE12-2

[0244] 9.2.1亮度重整形

[0245] 环路内亮度重整形的测试2(即,建议中的CE12-2)提供了更低复杂度的流水线,这也消除了帧间条带重构中分块帧内预测的解码时延。在帧间和帧内条带的重整形域中执行帧内预测。

[0246] 无论条带类型如何,帧内预测总是在重整形域中执行。利用这样的布置,帧内预测可以在先前的TU重构完成之后立即开始。这样的布置也可以为帧内模式提供统一的过程,而不是依赖于条带的。图7示出了基于模式的CE12-2解码过程的框图。

[0247] CE12-2还测试了亮度和色度残差缩放的16段分段线性(PWL)模型,而不是CE12-1的32段PWL模型。

[0248] CE12-2中的利用环路内亮度重整形器的帧间条带重构(较浅的阴影块指示重整形域中的信号:亮度残差;预测的帧内亮度;以及重构的帧内亮度)

[0249] 9.2.2依赖于亮度的色度残差缩放

[0250] 依赖于亮度的色度残差缩放是用定点整数运算实施的乘法过程。色度残差缩放补偿与色度信号的亮度信号相互作用。色度残差缩放应用于TU级别。更具体地,利用对应的亮度预测块的平均值。

[0251] 平均值用于识别PWL模型中的索引。该索引标识了缩放因子cScaleInv。色度残差乘以这个数。

[0252] 请注意,色度缩放因子是从前向映射的预测亮度值而不是重构的亮度值计算的。

[0253] 9.2.3 ILR边信息的信令通知

[0254] 参数(当前)在片组头中被传送(类似于ALF)。这些据报道需要40-100比特。

[0255] 以下规范基于JVET-L1001的版本9。添加的语法以黄色突出显示。

[0256] 在7.3.2.1序列参数集RBSP语法中

[0257]

seq_parameter_set_rbsp( ) {	描述符
sps_seq_parameter_set_id	ue(v)
…	
sps_triangle_enabled_flag	u(1)
sps_ladf_enabled_flag	u(1)
if (sps_ladf_enabled_flag) {	
sps_num_ladf_intervals_minus2	u(2)
sps_ladf_lowest_interval_qp_offset	se(v)
for(i = 0; i < sps_num_ladf_intervals_minus2 + 1; i++) {	
sps_ladf_qp_offset[i]	se(v)
sps_ladf_delta_threshold_minus1[i]	ue(v)
}	
}	
{{ **sps_reshaper_enabled_flag**	u(1) }}
rbsp_trailing_bits( )	

[0258]

}	

[0259] 在7.3.3.1通用片组头语法中

[0260]

tile_group_header( ) {	描述符
…	
if(num_tiles_in_tile_group_minus1 > 0) {	
offset_len_minus1	ue(v)
for(i = 0; i < num_tiles_in_tile_group_minus1; i++)	
entry_point_offset_minus1[i]	u(v)
}	
{{ if (sps_reshaper_enabled_flag) {	
tile_group_reshaper_model_present_flag	u(1)
if (tile_group_reshaper_model_present_flag)	
tile_group_reshaper_model ( )	
tile_group_reshaper_enable_flag	u(1)
if (tile_group_reshaper_enable_flag && (!(qtbtt_dual_tree_intra_flag && tile_group_type == I)))	
tile_group_reshaper_chr oma_residual_scale_flag	u(1)
} }}	
byte_alignment( )	
}	

[0261] 添加新的语法表片组重整形器模型：

[0262]

{{ tile_group_reshaper_model () {	描述符
reshaper_model_min_bin_idx	ue(v)
reshaper_model_delta_max_bin_idx	ue(v)
reshaper_model_bin_delta_abs_cw_prec_minus1	ue(v)
for (i = reshaper_model_min_bin_idx; i <= reshaper_model_max_bin_idx; i++) {	
reshape_model_bin_delta_abs_CW [i]	u(v)
if (reshaper_model_bin_delta_abs_CW[i]) > 0)	
reshaper_model_bin_delta_sign_CW_flag[i]	u(1)
}	
} }}	

[0263] {{在通用序列参数集RBSP语义中,添加以下语义:}}

[0264] 等于1的sps_reshaper_enabled_flag指定在编解码视频序列(CVS)中使用重整形器。等于0的sps_reshaper_enabled_flag指定在CVS中不使用重整形器。

[0265] {{在片组头语法中,添加以下语义}}

[0266] 等于1的tile_group_reshaper_model_present_flag指定tile_group_reshaper_model()存在于片组头中。等于0的tile_group_reshaper_model_present_flag指定tile_group_reshaper_model()不存在于片组头中。当tile_group_reshaper_model_present_flag不存在时,它被推断为等于0。

[0267] 等于1的tile_group_reshaper_enabled_flag指定对当前片组启用重整形器。等于0的tile_group_reshaper_enabled_flag指定不对当前片组启用重整形器。当tile_group_reshaper_enable_flag不存在时,它被推断为等于0。

[0268] 等于1的tile_group_reshaper_chroma_residual_scale_flag指定对当前片组启用色度残差缩放。等于0的tile_group_reshaper_chroma_residual_scale_flag指定不对当前片组启用色度残差缩放。当tile_group_reshaper_chroma_residual_scale_flag不存在时,它被推断为等于0。

[0269] {{添加tile_group_reshaper_model()语法}}

[0270] reshape_model_min_bin_idx指定要用于重整形器构建过程中的最小二进制位(或段)索引。reshape_model_min_bin_idx的值应当在0至MaxBinIdx(包括0和MaxBinidx)的范围内。MaxBinIdx的值应当等于15。

[0271] reshape_model_delta_max_bin_idx指定最大允许的二进制位(或段)索引MaxBinIdx减去要用于重整形器构建过程中的最大二进制位索引。reshape_model_max_bin_idx的值被设置为等于MaxBinIdx-reshape_model_delta_max_bin_idx。

[0272] reshaper_model_bin_delta_abs_cw_prec_minus1加1指定用于表示语法reshape_model_bin_delta_abs_CW[i]的比特数。

[0273] reshape_model_bin_delta_abs_CW[i]指定第i个二进制位的绝对偏移(delta)码字值。

[0274] reshaper_model_bin_delta_sign_CW_flag[i]指定reshape_model_bin_delta_abs_CW[i]的符号如下:

[0275] -如果reshape_model_bin_delta_sign_CW_flag[i]等于0,则对应的变量RspDeltaCW[i]是正值。

[0276] -否则(reshape_model_bin_delta_sign_CW_flag[i]不等于0),对应的变量RspDeltaCW[i]是负值。

[0277] 当reshape_model_bin_delta_sign_CW_flag[i]不存在时,它被推断为等于0。

[0278] 变量RspDeltaCW[i]＝(1 2*reshape_model_bin_delta_sign_CW[i])*reshape_model_bin_delta_abs_CW[i];

[0279] 变量RspCW[i]如以下步骤被推导:

[0280] 变量OrgCW被设置为等于(1<<$BitDepth_Y$)/(MaxBinIdx+1)。

[0281] -如果reshaper_model_min_bin_idx<＝i<＝reshaper_model_max_bin_idx则RspCW[i]＝OrgCW+RspDeltaCW[i]。

[0282] -否则,RspCW[i]＝0。

[0283] 如果$BitDepth_Y$的值等于10,则RspCW[i]的值应当在32至2*OrgCW-1的范围内。

[0284] 变量InputPivot[i](其中i在0至MaxBinIdx+1(包括0和MaxBinIdx+1)的范围内)被推导如下

[0285] InputPivot[i]＝i*OrgCW

[0286] 变量ReshapePivot[i](其中i在0至MaxBinIdx+1(包括0和MaxBinIdx+1)的范围内)、变量ScaleCoef[i]和InvScaleCoeff[i](其中i在0至MaxBinIdx(包括0和MaxBinIdx)的范围内)被推导如下:

[0287]
```
shiftY = 14
ReshapePivot[ 0 ] = 0;
for( i = 0; i <= MaxBinIdx ; i++) {
  ReshapePivot[ i + 1 ] = ReshapePivot[ i ] + RspCW[ i ]
  ScaleCoef[ i ] = ( RspCW[ i ] * (1 << shiftY) + (1 << (Log2(OrgCW) - 1))) >> (Log2(OrgCW))
  if ( RspCW[ i ] == 0 )
        InvScaleCoeff[ i ] = 0
  else
        InvScaleCoeff[ i ] = OrgCW * (1 << shiftY) / RspCW[ i ]
}
```

[0288] 变量ChromaScaleCoef[i](其中i在0至MaxBinIdx(包括0和MaxBinIdx)的范围内)被推导如下:

[0289] ChromaResidualScaleLut[64]={16384,16384,16384,16384,16384,16384,16384,8192,8192,8192,8192,5461,5461,5461,5461,4096,4096,4096,4096,3277,3277,3277,3277,2731,2731,2731,2731,2341,2341,2341,2048,2048,2048,1820,1820,1820,1638,1638,1638,1638,1489,1489,1489,1489,1365,1365,1365,1365,1260,1260,1260,1260,1170,1170,1170,1170,1092,1092,1092,1092,1024,1024,1024,1024};

[0290] shiftC=11

[0291] -if(RspCW[i]==0)

[0292] ChromaScaleCoef[i]=(1<<shiftC)

[0293] -Otherwise(RspCW[i]!=0),ChromaScaleCoef[i]=ChromaResidualScaleLut[RspCW[i]>>1]

[0294] 9.2.4 ILR的使用

[0295] 在编码器侧,每个图片(或片组)首先被转换到重整形域。并且所有的编解码过程都在重整形域中被执行。对于帧内预测,临近块在重整形域中;对于帧间预测,(从解码图片缓冲区的原始域生成的)参考块首先被转换到重整形域。然后,残差被生成并编解码到比特流。

[0296] 在整个图片(或片组)完成编码/解码之后,重整形域中的样点被转换到原始域,然后去方块滤波器和其他滤波器被应用。

[0297] 对于以下情况,禁用对预测信号的正重整形:

[0298] o当前块是帧内编解码的

[0299] o当前块被编解码为CPR(当前图片参考,也称为帧内块复制,IBC)

[0300] o当前块被编解码为组合的帧间-帧内模式(CIIP),并且对帧内预测块禁用正重整形

[0301] 10双向光流(BDOF)

[0302] 10.1 BIO的概述和分析

[0303] 在BIO中,首先执行运动补偿以生成当前块(在每个预测方向上)的第一预测。第一预测用于推导块内的每个子块或像素的空域梯度、时域梯度和光流,其然后用于生成第二预测,例如子块或像素的最终预测。细节描述如下。

[0304] 双向光流(BIO)方法是在双向预测的逐块运动补偿的基础上执行的逐样点运动细化。在一些实施方式中,样点级别运动细化不使用信令。

[0305] 假设$I^{(k)}$为块运动补偿之后的来自参考k(k=0,1)的亮度值,并且分别将$\partial I^{(k)}/\partial x$和$\partial I^{(k)}/\partial y$表示为$I^{(k)}$梯度的水平和垂直分量。假设光流有效,则运动矢量场$(v_x,v_y)$由下式给出:

[0306] $$\partial I^{(k)}/\partial t+v_x\partial I^{(k)}/\partial x+v_y\partial I^{(k)}/\partial y=0. \qquad \text{等式(11)}$$

[0307] 将该光流等式与用于每个样点的运动轨迹的Hermite插值组合,最终产生与函数值$I^{(k)}$以及导数$\partial I^{(k)}/\partial x$和$\partial I^{(k)}/\partial y$相匹配的唯一的三阶多项式。t=0时,该多项式的值

为BIO预测：

[0308] $$pred_{BIO}=1/2\cdot\left(I^{(0)}+I^{(1)}+v_x/2\cdot\left(\tau_1\partial I^{(1)}/\partial x-\tau_0\partial I^{(0)}/\partial x\right)+v_y/2\cdot\left(\tau_1\partial I^{(1)}/\partial y-\tau_0\partial I^{(0)}/\partial y\right)\right)$$ 等式（12）

[0309] 图8示出了双向光流(BIO)方法中的示例光流轨迹。这里，τ_0和τ_1表示到参考帧的距离。距离τ_0和τ_1基于Ref_0和Ref_1的POC进行计算：τ_0＝POC(当前)-POC(Ref_0)，τ_1＝POC(当前)-POC(Ref_1)。如果两个预测来自相同的时间方向(都来自过去或者都来自未来)，则符号是不同的(例如，$\tau_0\cdot\tau_1<0$)。在这种情况下，如果预测不是来自相同时刻(例如，$\tau_0\neq\tau_1$)，则应用BIO。

[0310] 运动矢量场(v_x,v_y)是通过最小化点A和B中的值之间的差Δ而确定的。图8示出了运动轨迹和参考帧平面相交的示例。模型仅使用Δ的局部泰勒展开的第一线性项：

[0311] $$\Delta=\left(I^{(0)}-I^{(1)}_0+v_x\left(\tau_1\partial I^{(1)}/\partial x+\tau_0\partial I^{(0)}/\partial x\right)+v_y\left(\tau_1\partial I^{(1)}/\partial y+\tau_0\partial I^{(0)}/\partial y\right)\right)$$ 等式（13）

[0312] 上述等式中的所有值取决于样点位置，表示为(i′,j′)。假设运动在局部周围区域中是一致的，则可以在以当前预测点(i,j)为中心的(2M+1)×(2M+1)正方形窗口Ω内部最小化Δ，其中M等于2：

[0313] $$\left(v_x,v_y\right)=\underset{v_x,v_y}{\operatorname{argmin}}\sum_{[i',j]\in\Omega}\Delta^2[i',j']$$ 等式（14）

[0314] 对于该优化问题，JEM使用简化的方法，首先在垂直方向上最小化，然后在水平方向上最小化。这将引起下式：

[0315] $$v_x=(s_1+r)>m?\,clip3\left(-thBIO,thBIO,-\frac{s_3}{(s_1+r)}\right):0$$ 等式（15）

[0316] $$v_y=(s_5+r)>m?\,clip3\left(-thBIO,thBIO,-\frac{s_6-v_xs_2/2}{(s_5+r)}\right):0$$ 等式（16）

[0317] 其中，

[0318] $$s_1=\sum_{[i',j]\in\Omega}\left(\tau_1\partial I^{(1)}/\partial x+\tau_0\partial I^{(0)}/\partial x\right)^2;\ s_3=\sum_{[i',j]\in\Omega}\left(I^{(1)}-I^{(0)}\right)\left(\tau_1\partial I^{(1)}/\partial x+\tau_0\partial I^{(0)}/\partial x\right);$$
$$s_2=\sum_{[i',j]\in\Omega}\left(\tau_1\partial I^{(1)}/\partial x+\tau_0\partial I^{(0)}/\partial x\right)\left(\tau_1\partial I^{(1)}/\partial y+\tau_0\partial I^{(0)}/\partial y\right);$$ 等式（17）
$$s_5=\sum_{[i',j]\in\Omega}\left(\tau_1\partial I^{(1)}/\partial y+\tau_0\partial I^{(0)}/\partial y\right)^2;\ s_6=\sum_{[i',j]\in\Omega}\left(I^{(1)}-I^{(0)}\right)\left(\tau_1\partial I^{(1)}/\partial y+\tau_0\partial I^{(0)}/\partial y\right)$$

[0319] 为了避免被零除或被非常小的值除，正则化参数r和m可以引入到等式(15)和等式(16)中，其中：

[0320] $r=500\cdot4^{d-8}$ 等式(18)

[0321] $m=700\cdot4^{d-8}$ 等式(19)

[0322] 这里，d是视频样点的比特深度。

[0323] 为了保持对BIO的内存访问与常规双向预测运动补偿相同，所有预测和梯度值$I^{(k)}$，$\partial I^{(k)}/\partial x$，$\partial I^{(k)}/\partial y$是针对当前块内部的位置而计算的。图9A示出了块900外部的访问位置的示例。如图9A所示，在等式(17)中，以预测块的边界上的当前预测点为中心的(2m+1)×(2m+1)正方形窗口Ω需要访问块外部的位置。在JEM，块外部的值$I^{(k)}$，$\partial I^{(k)}/\partial x$，$\partial I^{(k)}/\partial y$被设置为等于块内部的最近可用值。例如，这可以被实施为填充区域

901,如图9B所示。

[0324] 利用BIO,可以为每个样点细化运动场是可能的。为了降低计算复杂度,在JEM中使用BIO的基于块的设计。可以基于4×4块来计算运动细化。在基于块的BIO中,可以聚合4×4块中的所有样点的、等式(17)中的s_n的值,然后s_n的聚合值用于4×4块的推导的BIO运动矢量偏移。更具体地,以下公式可以用于基于块的BIO推导:

[0325]
$$s_{1,b_k}=\sum_{(x,y)\in b_k}\sum_{[i',j]\in\Omega(x,y)}\left(\tau_1\partial I^{(1)}/\partial x+\tau_0\partial I^{(0)}/\partial x\right)^2;s_{3,b_k}=\sum_{(x,y)\in b_k}\sum_{[i',j]\in\Omega}\left(I^{(1)}-I^{(0)}\right)\left(\tau_1\partial I^{(1)}/\partial x+\tau_0\partial I^{(0)}/\partial x\right)$$
$$s_{2,b_k}=\sum_{(x,y)\in b_k}\sum_{[i',j]\in\Omega}\left(\tau_1\partial I^{(1)}/\partial x+\tau_0\partial I^{(0)}/\partial x\right)\left(\tau_1\partial I^{(1)}/\partial y+\tau_0\partial I^{(0)}/\partial y\right) \quad \text{等式(20)}$$
$$s_{5,b_k}=\sum_{(x,y)\in b_k}\sum_{[i',j]\in\Omega}\left(\tau_1\partial I^{(1)}/\partial y+\tau_0\partial I^{(0)}/\partial y\right)^2;s_{6,b_k}=\sum_{(x,y)\in b_k}\sum_{[i',j]\in\Omega}\left(I^{(1)}-I^{(0)}\right)\left(\tau_1\partial I^{(1)}/\partial y+\tau_0\partial I^{(0)}/\partial y\right)$$

[0326] 这里,b_k表示属于预测块的第k个4×4块的样点集合。等式(15)和等式(16)中的s_n被替换为$((s_{n,bk})>>4)$以推导相关联的运动矢量偏移。

[0327] 在一些场景中,由于噪音或不规则运动,BIO的MV细化可能不可靠。因此,在BIO中,MV细化的幅度被裁剪到阈值。基于当前图片的参考图片是否都来自一个方向来确定阈值。例如,如果当前图片的所有参考图片都来自一个方向,则阈值的值被设置为$12\times2^{14-d}$;否则,它被设置为$12\times2^{13-d}$。

[0328] 可以使用与HEVC运动补偿过程(例如,2D可分有限脉冲响应(FIR))一致的操作利用运动补偿插值来同时计算BIO的梯度。在一些实施例中,2D可分FIR的输入是与运动补偿过程和根据块运动矢量的分数部分的分数位置(fracX,fracY)相同的参考帧样点。对于水平梯度$\partial I/\partial x$,首先使用与具有去缩放(de-scaling)移位d-8的分数位置fracY相对应的BIOfilterS对信号进行垂直插值。然后在与具有去缩放移位18-d的分数位置fracX相对应的水平方向上应用梯度滤波器BIOfilterG。对于垂直梯度$\partial I/\partial y$,使用与具有去缩放移位d-8的分数位置fracY相对应的BIOfilterG来垂直应用梯度滤波器。然后在与具有去缩放移位18-d的分数位置fracX相对应的水平方向上使用BIOfilterS来执行信号位移。用于梯度计算的插值滤波器BIOfilterG和用于信号位移的插值滤波器BIOfilterF的长度可以更短(例如,6抽头),以便保持合理的复杂度。表2示出了在BIO中可以用于块运动矢量的不同分数位置的梯度计算的示例滤波器。表3示出了在BIO中可以用于预测信号生成的示例插值滤波器。

[0329] 表2:在BIO中用于梯度计算的示例性滤波器

[0330]

分数精度位置	梯度的插值滤波器(BIOfilterG)
0	{ 8, −39, −3, 46, −17, 5}
1/16	{ 8, −32, −13, 50, −18, 5}
1/8	{ 7, −27, −20, 54, −19, 5}

[0331]

3/16	{ 6, −21, −29, 57, −18, 5}
1/4	{ 4, −17, −36, 60, −15, 4}
5/16	{ 3, −9, −44, 61, −15, 4}
3/8	{ 1, −4, −48, 61, −13, 3}
7/16	{ 0, 1, −54, 60, −9, 2}
1/2	{ −1, 4, −57, 57, −4, 1}

[0332] 表2:在BIO中用于预测信号生成的示例性插值滤波器

[0333]

分数精度位置	预测信号的插值滤波器(BIOfilterS)
0	{0,0,64,0,0,0}
1/16	{1,-3,64,4,-2,0}
1/8	{1,-6,62,9,-3,1}
3/16	{2,-8,60,14,-5,1}
1/4	{2,-9,57,19,-7,2}
5/16	{3,-10,53,24,-8,2}
3/8	{3,-11,50,29,-9,2}
7/16	{3,-11,44,35,-10,3}
1/2	{3,-10,35,44,-11,3}

[0334] 在JEM,当两个预测来自不同的参考图片时,BIO可以应用于所有双向预测块。当对CU启用局部照明补偿(LIC)时,可以禁用BIO。

[0335] 在一些实施例中,在正常的MC过程之后,OBMC被应用于块。为了降低计算复杂度,在OBMC过程期间可以不应用BIO。这意味着BIO在使用块自身的MV时被应用于块的MC过程中,而在OBMC过程期间使用临近块的MV时不应用于MC过程中。

[0336] 11 JVET-N0236中的利用光流的预测细化(PROF)

[0337] 该文献建议一种利用光流来细化基于子块的仿射运动补偿预测的方法。在执行基于子块的仿射运动补偿之后,通过添加由光流等式推导的差来细化预测样点,这被称为利用光流的预测细化(PROF)。该建议的方法可以在不增加内存访问带宽的情况下实现像素级别粒度的帧间预测。

[0338] 为了实现运动补偿的更精细的粒度,该文献建议一种利用光流来细化基于子块的仿射运动补偿预测的方法。在执行基于子块的仿射运动补偿之后,通过添加由光流等式推导的差来细化亮度预测样点。建议的PROF被描述为以下四个步骤。

[0339] 步骤1)执行基于子块的仿射运动补偿以生成子块预测I(i,j)。

[0340] 步骤2)使用3抽头滤波器[-1,0,1],在每个样点位置处计算子块预测的空域梯度$g_x(i,j)$和$g_y(i,j)$。

[0341] $g_x(i,j)=I(i+1,j)-I(i-1,j)$

[0342] $g_y(i,j)=I(i,j+1)-I(i,j-1)$

[0343] 对于梯度计算,子块预测在每一侧上扩展一个像素。为了减少内存带宽和复杂性,扩展边界上的像素是从参考图片中的最近的整数像素位置复制的。因此,避免了对填充区

域的附加插值。

[0344] 步骤3)亮度预测细化由光流等式计算。

[0345] $\Delta I(i,j)=g_x(i,j)*\Delta v_x(i,j)+g_y(i,j)*\Delta v_y(i,j)$

[0346] 这里,$\Delta v(i,j)$是针对样点位置(i,j)计算的像素MV(由v(i,j)表示)和像素(i,j)所属的子块的子块MV之间的差,如图10所示。

[0347] 由于仿射模型参数和相对于子块中心的像素位置在子块与子块之间不改变,因此可以针对第一子块计算$\Delta v(i,j)$,并将其重新用于相同CU中的其他子块。假设x和y是从像素位置到子块的中心的水平和垂直偏移,$\Delta v(x,y)$可以由以下等式推导:

[0348] $$\begin{cases}\Delta v_x(x,y)=c*x+d*y\\ \Delta v_y(x,y)=e*x+f*y\end{cases}$$

[0349] 对于4参数仿射模型,

[0350] $$\begin{cases}c=f=\dfrac{v_{1x}-v_{0x}}{w}\\ e=-d=\dfrac{v_{1y}-v_{0y}}{w}\end{cases}$$

[0351] 对于6参数仿射模型,

[0352] $$\begin{cases}c=\dfrac{v_{1x}-v_{0x}}{w}\\ d=\dfrac{v_{2x}-v_{0x}}{h}\\ e=\dfrac{v_{1y}-v_{0y}}{w}\\ f=\dfrac{v_{2y}-v_{0y}}{h}\end{cases}$$

[0353] 这里,(v_{0x},v_{0y})、(v_{1x},v_{1y})、(v_{2x},v_{2y})是左上方、右上方和左下方控制点运动矢量,w和h是CU的宽度和高度。

[0354] 步骤4)最后,亮度预测细化被添加到子块预测I(i,j)。最终预测I'如以下等式生成:

[0355] $I'(i,j)=I(i,j)+\Delta I(i,j)$

[0356] 12现有实施方式的缺点

[0357] JVET-N0242设计中的非线性ALF(NLALF)具有以下问题:

[0358] (1)在NLALF中需要许多裁剪操作。

[0359] (2)在基于CTU的ALF中,当alf_num_available_temporal_filter_sets_luma等于0时,没有可用的时域亮度滤波器。然而,alf_temporal_index仍然可能被信令通知。

[0360] (3)在基于CTU的ALF中,当alf_signal_new_filter_chroma等于0时,没有为色度分量信令通知新的滤波器,并且假设使用时域色度滤波器。然而,不保证时域色度滤波器是可用的。

[0361] (4)在基于CTU的ALF中,alf_num_available_temporal_filter_sets_luma可能大于可用的时域滤波器集合。

[0362] 13用于视频编解码的自适应环路滤波的示例性方法

[0363] 当前公开的技术的实施例克服了现有实施方式的缺点,从而提供具有更高编解码效率的视频编解码。基于所公开的技术,用于自适应环路滤波的技术可以增强现有和未来的视频编解码标准,在针对各种实施方式描述的以下示例中进行了阐述。下面提供的所公开的技术的示例解释了一般概念,并且不意味着被解释为限制。在示例中,除非明确相反指出,否则在这些示例中描述的各种特征可以被组合。

[0364] 1.代替裁剪样点差,建议在滤波过程期间对中间结果应用裁剪操作。假设在滤波过程中利用的当前样点的临近样点(相邻或不相邻)可以被分类为N(N>=1)个组。

[0365] a.在一个示例中,为一组计算一个或多个中间结果,并且可以对一个或多个中间结果执行裁剪。

[0366] i.例如,对于一组,可以首先计算每个临近像素和当前像素之间的差,然后可以使用对应的ALF系数对这些差进行加权平均(表示为wAvgDiff)。可以针对该组对wAvgDiff执行一次裁剪。

[0367] b.不同的裁剪参数可以用于不同的组。

[0368] c.在一个示例中,裁剪被应用于滤波器系数乘以样点差的最终加权和。

[0369] i.例如,N=1,并且裁剪可以被执行如下,其中,K(d,b)=min(b,max(-b,d))是裁剪函数,并且k是裁剪参数。

[0370] $O(x,y)=I(x,y)+K(\Sigma_{(i,j)\neq(0,0)}w(i,j).(I(x+i,y+j)-I(x,y)),k)$

[0371] 1)另外,可替代地,加权和$\Sigma_{(i,j)\neq(0,0)}w(i,j).(I(x+i,y+j)-I(x,y))$还可以被取整为整数值,诸如经由具有或不具有取整的移位。

[0372] 2.当对一个样点进行滤波时,如果N(N>1)个临近样点共享一个滤波器系数,则可以对所有N个临近像素进行一次裁剪(例如,非线性ALF所需的)。

[0373] a.例如,如果I(x+i1,y+j1)和I(x+i2,y+j2)共享一个滤波器系数w(i1,j1)(或/和一个裁剪参数k(i1,j1)),则裁剪可以如下进行一次:clipValue(i1,j1)=K(I(x+i1,y+j1)+I(x+i2,y+j2)-2*I(x,y),k(i1,j1)),并且w(i1,j1)*clipValue(i1,j1)可以用于替换等式(14)中的w(i1,j1)*K(I(x+i1,y+j1)-I(x,y),k(i1,j1))+w(i2,j2)*K(I(x+i2,y+j2)-I(x,y),k(i2,j2))。

[0374] i.在一个示例中,i1和i2可以在对称位置处。此外,j1和j2可以在对称位置处。

[0375] 1.在一个示例中,i1等于(-i2),并且j1等于(-j2)。

[0376] ii.在一个示例中,(x+i1,y+j1)和(x,y)之间的距离以及(x+i2,j+j2)和(x,y)之间的距离可以相同。

[0377] iii.当滤波器形状在对称模式下时,项目符号2中的所公开的方法被启用。

[0378] iv.此外,可替代地,可以从比特流信令通知/推导与I(x+i1,y+j1)相关联的裁剪参数,由ClipParam表示,并且上面提到的k(i1,j1)是从信令通知的裁剪参数(诸如2*ClipParam)推导的。

[0379] b.例如,如果(i,j)∈C共享一个滤波器系数w1(或/和一个裁剪参数k1),并且C包含N个元素,则裁剪可以如下进行一次:

[0380] $$clipValue = K\left(\left(\sum_{(i,j)\in C} I(x+i,y+j) \right) - N * I(x,y), k1 \right)$$

[0381] 其中,k1是与C相关联的裁剪参数,并且clipValue*w1可以用于替换等式(14)中的以下项:

[0382] $$\sum_{(i,j)\in C} w(i,j) * K(I(x+i,y+j) - I(x,y), k(i,j))$$

[0383] i.此外,可替代地,可以从比特流信令通知/推导与I(x+i,y+j)相关联的裁剪参数,由ClipParam表示,并且k1是从信令通知的裁剪参数(诸如N*ClipParam)推导的。

[0384] ii.可替代地,$\Sigma_{(i,j)\in C}$I(x+i,y+j)或($\Sigma_{(i,j)\in CI}$(x+i,y+j))-N*I(x,y)在裁剪之前被右移。

[0385] c.在一个示例中,可以对N个临近样点中的M1(M1<=N)个进行一次裁剪。

[0386] d.在一个示例中,可以将N个临近样点分类为M2个组,并且可以对每个组进行一次裁剪。

[0387] e.在一个示例中,该方法可以被应用于某些或所有色彩分量。

[0388] i.例如,它可以被应用于亮度分量。

[0389] ii.例如,它可以被应用于Cb或/和Cr分量。

[0390] 3.可以在本公开中使用将input裁剪到包括min和max的范围[min,max]的裁剪函数K(min,max,input)。

[0391] a.在一个示例中,可以在上面的项目符号中使用将input裁剪到不包括min和max的范围(min,max)的裁剪函数K(min,max,input)。

[0392] b.在一个示例中,可以在上面的项目符号中使用将input裁剪为在包括max但不包括min的范围(min,max]内的裁剪函数K(min,max,input)。

[0393] c.在一个示例中,可以在上面的项目符号中使用将input裁剪为在包括min但不包括max的范围[min,max)内的裁剪函数K(min,max,input)。

[0394] 4.当时域ALF系数集不可用时(例如,之前没有ALF系数被编码/解码,或者编码/解码的ALF系数被标记为“不可用”),可以跳过使用哪个时域ALF系数集的指示的信令通知。

[0395] a.在一个示例中,当时域ALF系数集不可用时,如果新的ALF系数和固定的ALF系数都没有用于CTB/块/片组/片/条带/图片,则ALF被推断为不允许用于CTB/块/片组/片/条带/图片。

[0396] i.此外,可替代地,在这种情况下,即使可以指示(例如,alf_ctb_flag对于CTU/块为真)ALF被应用于CTB/块/片组/片/条带/图片,ALF也可以最终被推断为不允许用于CTB/块/片组/片/条带/图片。

[0397] b.在一个示例中,当时域ALF系数集不可用时,仅新的ALF系数或固定的ALF系数等可以被指示用于一致性比特流中的CTB/块/片组/片/条带/图片。

[0398] i.例如,alf_use_new_filter或alf_use_fixed_filter应当为真。

[0399] c.在一个示例中,如果满足以下条件,则比特流被认为是非一致性比特流:当时域ALF系数集不可用时,对于其中指示ALF要被采用的CTB/块/片组/片/条带/图片,新的ALF系数和固定的ALF系数都不被指示用于它。

[0400] i.例如,alf_use_new_filter和alf_use_fixed_filter都为假的比特流被认为是非一致性比特流。

[0401] d.在一个示例中,当alf_num_available_temporal_filter_sets_luma等于0时,可以不信令通知alf_temporal_index。

[0402] e.建议的方法可以被不同地应用于不同的色彩分量。

[0403] 5.多少个时域ALF系数集可以用于片组/片/条带/图片/CTB/块/视频单元可以取决于可用的时域ALF系数集(表示为ALF_{avai}),例如被标记为“可用”的先前编码/解码的ALF系数集。

[0404] a.在一个示例中,不多于ALF_{avai}个时域ALF系数集可以用于片组/片/条带/图片/CTB/块。

[0405] b.不多于min(N,ALF_{avai})个时域ALF系数集可以用于片组/片/条带/图片/CTB/块,其中,N>=0。例如,N=5。

[0406] 6.新的ALF系数集可以在它们被编码/解码之后被标记为“可用”。同时,当遇到IRAP(帧内随机访问点)访问单元或/和IRAP图片或/和IDR(瞬时解码刷新)访问单元或/和IDR图片时,所有“可用”ALF系数集可以全部被标记为“不可用”。

[0407] a.“可用”ALF系数集可以用作后面编解码的图片/片/片组/条带/CTB/块的时域ALF系数集。

[0408] b.可以在最大尺寸等于N(N>0)的一个ALF系数集列表中维护“可用”ALF系数集。

[0409] i.可以按先进先出的顺序维护ALF系数集列表。

[0410] c.当被标记为“不可用”时,相关联的ALF APS信息从比特流移除或者被其他ALF APS信息替换。

[0411] 7.可以为每个时域层维护一个ALF系数集列表。

[0412] 8.可以为K个临近时域层维护一个ALF系数集列表。

[0413] 9.根据图片是否是仅从前面的图片(以显示顺序)预测的,可以为不同的图片维护不同的ALF系数集列表。

[0414] a.例如,可以为仅从前面的图片预测的图片维护一个ALF系数集列表。

[0415] b.例如,可以为从前面的图片和后面的图片两者预测的图片维护一个ALF系数集列表。

[0416] 10.在遇到IRAP访问单元或/和IRAP图片或/和IDR访问单元或/和IDR图片之后,可以清空ALF系数集列表。

[0417] 11.可以为不同的色彩分量维护不同的ALF系数集列表。

[0418] a.在一个示例中,为亮度分量维护一个ALF系数集列表。

[0419] b.在一个示例中,为Cb或/和Cr分量维护一个ALF系数集列表。

[0420] 12.可以维护一个ALF系数集列表,然而,对于不同的图片/片组/片/条带/CTU,列表中的条目可以被分配有不同的索引(或优先级)。

[0421] a.在一个示例中,ALF系数集可以被分配有它和当前图片/片组/片/条带/CTU之间的升序绝对时域层差的升序索引。

[0422] b.在一个示例中,ALF系数集可以被分配有它和当前图片/片组/片/条带/CTU之间的升序绝对POC(图片顺序计数)差的升序索引。

[0423] c.在一个示例中,假设有K个ALF系数集被当前图片/片组/片/条带/CTU允许,则它们可以是具有最小索引的K个ALF系数集。

[0424] d.在一个示例中,对哪个时域ALF系数集由当前图片/片组/片/条带/CTU使用的指示还可以取决于所分配的索引而不是列表中的原始条目索引。

[0425] 13.可以将在ALF中使用的临近样点分类为K(K>=1)个组,并且可以为每个组信令通知一个裁剪参数集合。

[0426] 14.可以为某些或所有固定的ALF滤波器集预定义裁剪参数。

[0427] a.可替代地,可以为由当前片组/条带/图片/片使用的某些或所有固定的滤波器集信令通知裁剪参数。

[0428] i.在一个示例中,可以仅为某些色彩分量(例如,亮度分量)信令通知裁剪参数。

[0429] b.可替代地,当使用固定的ALF滤波器集时,可以不执行裁剪。

[0430] i.在一个示例中,可以对某些色彩分量执行裁剪,而对其他色彩分量不执行裁剪。

[0431] 15.裁剪参数可以与ALF系数一起存储,并且可以由后面编解码的CTU/CU/片/片组/条带/图片继承。

[0432] a.在一个示例中,当时域ALF系数集由CTU/CU/片/片组/条带/图片使用时,还可以使用对应的ALF裁剪参数。

[0433] i.在一个示例中,可以仅为某些色彩分量(例如,亮度分量)继承裁剪参数。

[0434] b.可替代地,当时域ALF系数集由CTU/CU/片/片组/条带/图片使用时,可以信令通知裁剪参数。

[0435] i.在一个示例中,可以仅为某些色彩分量(例如,亮度分量)信令通知裁剪参数。

[0436] c.在一个示例中,可以为某些色彩分量继承裁剪参数,并且可以为其他色彩分量信令通知裁剪参数。

[0437] d.在一个示例中,当使用时域ALF系数集时,不执行裁剪。

[0438] i.在一个示例中,可以对某些色彩分量执行裁剪,而对其他色彩分量不执行裁剪。

[0439] 16.是否使用非线性ALF可以取决于ALF滤波器集类型(例如,固定的ALF滤波器集、时域ALF滤波器集或信令通知的ALF系数集)。

[0440] a.在一个示例中,如果当前CTU使用固定的ALF滤波器集或时域ALF滤波器集(也称为,使用先前信令通知的滤波器集),则非线性ALF可以不用于当前CTU。

[0441] b.在一个示例中,当alf_luma_use_fixed_filter_flag等于1时,非线性ALF可以用于当前条带/片组/片/CTU。

[0442] 17.可以根据ALF滤波器集类型(例如,固定的ALF滤波器集、时域ALF滤波器集或信令通知的ALF系数集)来有条件地信令通知非线性ALF裁剪参数。

[0443] a.在一个示例中,可以为所有ALF滤波器集信令通知非线性ALF裁剪参数。

[0444] b.在一个示例中,可以仅为信令通知的ALF滤波器系数集信令通知非线性ALF裁剪参数。

[0445] c.在一个示例中,可以仅为固定的ALF滤波器系数集信令通知非线性ALF裁剪参数。

[0446] 上面描述的示例可以被并入可以在视频解码器和/或视频编码器处实施的下面描述的方法(例如,方法1110、1120、1130、1140、1150和1160)的上下文中。

[0447] 图11A示出了用于视频处理的示例性方法的流程图。方法1110包括,在操作1112,对视频的当前视频块执行滤波过程,其中该滤波过程使用滤波器系数并且包括具有至少一个中间结果的两个或更多个操作。

[0448] 方法1110包括,在操作1114,对至少一个中间结果应用裁剪操作。

[0449] 方法1110包括,在操作1116,基于至少一个中间结果来执行当前视频块和视频的比特流表示之间的转换。在一些实施例中,至少一个中间结果基于滤波器系数和当前视频块的当前样点与当前样点的临近样点之间的差的加权和。

[0450] 图11B示出了用于视频处理的示例性方法的流程图。方法1120包括,在操作1122,将视频的当前视频块编码为视频的比特流表示,其中,当前视频块是用自适应环路滤波器(ALF)编解码的。

[0451] 方法1120包括,在操作1124,基于一个或多个时域自适应滤波器集合的可用性或使用,在比特流表示中选择性地包括一个或多个时域自适应滤波器集合内的时域自适应滤波器集合的指示。

[0452] 图11C示出了用于视频处理的示例性方法的流程图。方法1130包括,在操作1132,基于视频的比特流表示中的时域自适应滤波器集合的指示来确定一个或多个时域自适应滤波器集合的可用性或使用,其中该一个或多个时域自适应滤波器集合包括适用于用自适应环路滤波器(ALF)编解码的视频的当前视频块的时域自适应滤波器集合。

[0453] 方法1130包括,在操作1134,通过基于该确定来选择性地应用时域自适应滤波器集合,从比特流表示生成解码的当前视频块。

[0454] 图11D示出了用于视频处理的示例性方法的流程图。方法1140包括,在操作1142,基于可用的时域自适应环路滤波(ALF)系数集,为用自适应环路滤波器编解码的当前视频块确定多个时域ALF系数集,其中,可用的时域ALF系数集已经在该确定之前被编码或解码,并且其中,多个ALF系数集用于包括当前视频块的片组、片、条带、图片、编解码树块(CTB)或视频单元。

[0455] 方法1140包括,在操作1144,基于多个时域ALF系数集,执行当前视频块和当前视频块的比特流表示之间的转换。

[0456] 图11E示出了用于视频处理的示例性方法的流程图。方法1150包括,在操作1152,为视频的当前视频块和视频的比特流表示之间的转换确定视频的视频区域的头中的自适应环路滤波(ALF)的指示等于与比特流表示相关联的自适应参数集(APS)网络抽象层(NAL)单元中的ALF的指示。

[0457] 方法1150包括,在操作1154,执行转换。

[0458] 图11F示出了用于视频处理的示例性方法的流程图。方法1160包括,在操作1162,基于视频的视频区域使用的自适应环路滤波器的类型,对视频的当前视频块和视频的比特流表示之间的转换选择性地启用非线性自适应环路滤波(ALF)操作。

[0459] 方法1160包括,在操作1164,在该选择性地启用之后执行转换。

[0460] 10所公开的技术的示例实施方式

[0461] 10.1实施例#1

[0462] 假设分别为亮度和色度维护一个ALF系数集列表,并且这两个列表的尺寸分别为lumaALFSetSize和chromaALFSetSize。ALF系数集列表的最大尺寸分别为lumaALFSetMax

(例如,lumaALFSetMax等于5)和chromaALFSetMax(例如,chromaALFSetMax等于5)。

[0463] 新添加的部分以双粗体花括号括起来,即**{{a}}**表示“a”被添加,并且删除的部分以双方括号括起来,即[[a]]表示“a”被删除。

[0464] 7.3.3.2自适应环路滤波器数据语法

[0465]

alf_data( ) {	描述符
alf_chroma_idc	tu(v)
alf_signal_new_filter_luma	u(1)
if(alf_signal_new_filter_luma > 0) {	
alf_luma_use_fixed_filter_flag	u(1)
if(alf_luma_use_fixed_filter_flag){	
alf_luma_fixed_filter_set_index	tb(v)
alf_luma_fixed_filter_usage_pattern	u(1)
if (alf_luma_fixed_filter_usage_pattern > 0)	
for (i = 0; i < NumAlfFilters; i++)	
alf_luma_fixed_filter_usage[i]	u(1)
alf_num_available_temporal_filter_sets_luma	tb(1)
alf_luma_num_filters_signalled_minus1	tb(v)
alf_luma_type_flag	u(1)
if(alf_luma_num_filters_signalled_minus1 > 0) {	
for(filtIdx = 0; filtIdx < NumAlfFilters; filtIdx++)	
alf_luma_coeff_delta_idx[filtIdx]	tu(v)
}	
alf_luma_coeff_delta_flag	u(1)
if (!alf_luma_coeff_delta_flag && alf_luma_num_filters_signalled_minus1 > 0)	
alf_luma_coeff_delta_prediction_flag	u(1)
alf_luma_min_eg_order_minus1	tu(v)
for(i = 0; i < (alf_luma_type_flag == 1) ? 2 : 3; i++)	
alf_luma_eg_order_increase_flag[i]	u(1)

[0466]

if (alf_luma_coeff_delta_flag) {	
for(sigFiltIdx = 0; sigFiltIdx <= alf_luma_num_filters_signalled_minus1; sigFiltIdx++)	
alf_luma_coeff_flag[sigFiltIdx]	u(1)
}	
for(sigFiltIdx = 0; sigFiltIdx <= alf_luma_num_filters_signalled_minus1; sigFiltIdx++){	
if (alf_luma_coeff_flag[sigFiltIdx]) {	
for (j = 0; j < (alf_luma_type_flag == 1) ? 6 : 12; j++) {	
alf_luma_coeff_delta_abs[sigFiltIdx][j]	uek(v)
if(alf_luma_coeff_delta_abs[sigFiltIdx][j])	
alf_luma_coeff_delta_sign[sigFiltIdx][j]	u(1)
}	
}	
}	
}	
if (alf_chroma_idc > 0) {	
alf_signal_new_filter_chroma	u(1)
if (alf_signal_new_filter_chroma){	
alf_chroma_min_eg_order_minus1	tu(v)
for(i = 0; i < 2; i++)	
alf_chroma_eg_order_increase_flag[i]	u(1)
for(j = 0; j < 6; j++) {	
alf_chroma_coeff_abs[j]	uek(v)
if(alf_chroma_coeff_abs[j] > 0)	
alf_chroma_coeff_sign[j]	u(1)
}	
}	
}	
}	

[0467] 7.3.4.2编解码树单元语法

[0468]

coding_tree_unit( ) {	**描述符**
xCtb = (CtbAddrInRs % PicWidthInCtbsY) << CtbLog2SizeY	
yCtb = (CtbAddrInRs / PicWidthInCtbsY) << CtbLog2SizeY	
if(slice_alf_enable_flag){	
alf_ctb_flag[0][xCtb >> Log2CtbSize][yCtb >> Log2CtbSize]	ae(v)
if (alf_ctb_flag[0][xCtb >> Log2CtbSize][yCtb >> Log2CtbSize] {	
coding_alf_ctu_filter_set_index(CtbAddrInRs)	
}	
if(alf_chroma_idc == 1 || alf_chroma_idc == 3)	
alf_ctb_flag[1][xCtb >> Log2CtbSize][yCtb >> Log2CtbSize]	ae(v)
if(alf_chroma_idc == 2 || alf_chroma_idc == 3)	
alf_ctb_flag[2][xCtb >> Log2CtbSize][yCtb >> Log2CtbSize]	ae(v)
}	
if(slice_type = = I && qtbtt_dual_tree_intra_flag) {	
dual_tree_implicit_qt_split (xCtb, yCtb, CtbLog2SizeY, 0)	
else	
coding_quadtree(xCtb, yCtb, CtbLog2SizeY, 0, SINGLE_TREE)	
}	

[0469]

coding_alf_ctu_filter_set_index(CtbAddrInRs, slice_type){	描述符
xCtb = (CtbAddrInRs % PicWidthInCtbsY) << CtbLog2SizeY	
yCtb = (CtbAddrInRs / PicWidthInCtbsY) << CtbLog2SizeY	
alf_use_new_filter	ae(1)
if (alf_use_new_filter == 0){	
alf_use_fixed_filter	ae(1)
}	
if (alf_use_new_filter){	
alf_luma_ctb_filter_set_index[xCtb >> Log2CtbSize][yCtb >> Log2CtbSize] = 16	
}	
else if (alf_use_fixed_filter){	
alf_fixed_filter_index	tb(v)
alf_luma_ctb_filter_set_index[xCtb >> Log2CtbSize][yCtb >> Log2CtbSize] = alf_fixed_filter_index	
}	
Else {{ if (alf_num_available_temporal_filter_sets_luma > 0) }} {	
alf_temporal_index	tb(v)
alf_luma_ctb_filter_set_index[xCtb >> Log2CtbSize][yCtb >> Log2CtbSize] = 16 + alf_temporal_index	
}	

[0470] 等于1的alf_signal_new_filter_luma指定新的亮度滤波器集被信令通知。等于0的alf_signal_new_filter_luma指定新的亮度滤波器集不被信令通知。当不存在时,它为0。

[0471] 等于1的alf_luma_use_fixed_filter_flag指定固定的滤波器集用于信令通知自适应环路滤波器。等于0的alf_luma_use_fixed_filter_flag指定固定的滤波器集不用于信令通知自适应环路滤波器。

[0472] alf_num_available_temporal_filter_sets_luma指定可以用于当前条带的可用时域滤波器集的数量,它可以来自0..[[5]]{{lumaALFSetSize}}。当不存在时,它为0。

[0473] {{约束alf_signal_new_filter_luma或alf_luma_use_fixed_filter_flag在alf_num_available_temporal_filter_sets_luma等于零时必须等于1。}}

[0474] 等于1的alf_signal_new_filter_chroma指定新的色度滤波器被信令通知。等于0的alf_signal_new_filter_chroma指定新的色度滤波器不被信令通知。

[0475] {{约束alf_signal_new_filter_chroma在chromaALFSetSize等于0时必须等于1。}}

[0476] 10.2实施例#2

[0477] 假设分别为亮度和色度维护一个ALF系数集列表,并且这两个列表的尺寸分别为lumaALFSetSize和chromaALFSetSize。ALF系数集列表的最大尺寸分别为lumaALFSetMax(例如,lumaALFSetMax等于5)和chromaALFSetMax(例如,chromaALFSetMax等于5)。

[0478] 新添加的部分以双粗体花括号括起来,即**{{a}}**表示“a”被添加,并且删除的部分以双方括号括起来,即[[a]]表示“a”被删除。

[0479] 7.3.3.2自适应环路滤波器数据语法

[0480]

alf_data( ) {	描述符
alf_chroma_idc	tu(v)
alf_signal_new_filter_luma	u(1)
if(alf_signal_new_filter_luma > 0) {	
alf_luma_use_fixed_filter_flag	u(1)
if(alf_luma_use_fixed_filter_flag){	
alf_luma_fixed_filter_set_index	tb(v)
alf_luma_fixed_filter_usage_pattern	u(1)
if (alf_luma_fixed_filter_usage_pattern > 0)	
for (i = 0; i < NumAlfFilters; i++)	
alf_luma_fixed_filter_usage[i]	u(1)
{{ if (lumaALFSetSize > 0) }}	
alf_num_available_temporal_filter_sets_luma	tb(1)
alf_luma_num_filters_signalled_minus1	tb(v)
alf_luma_type_flag	u(1)
if(alf_luma_num_filters_signalled_minus1 > 0) {	
for(filtIdx = 0; filtIdx < NumAlfFilters; filtIdx++)	
alf_luma_coeff_delta_idx[filtIdx]	tu(v)
}	
alf_luma_coeff_delta_flag	u(1)
if (!alf_luma_coeff_delta_flag && alf_luma_num_filters_signalled_minus1 > 0)	
alf_luma_coeff_delta_prediction_flag	u(1)
alf_luma_min_eg_order_minus1	tu(v)
for(i = 0; i < (alf_luma_type_flag == 1) ? 2 : 3; i++)	
alf_luma_eg_order_increase_flag[i]	u(1)

[0481]

if (alf_luma_coeff_delta_flag) {	
for(sigFiltIdx = 0; sigFiltIdx <= alf_luma_num_filters_signalled_minus1; sigFiltIdx++)	
alf_luma_coeff_flag[sigFiltIdx]	u(1)
}	
for(sigFiltIdx = 0; sigFiltIdx <= alf_luma_num_filters_signalled_minus1; sigFiltIdx++){	
if (alf_luma_coeff_flag[sigFiltIdx]) {	
for (j = 0; j < (alf_luma_type_flag == 1) ? 6 : 12; j++) {	
alf_luma_coeff_delta_abs[sigFiltIdx][j]	uek(v)
if(alf_luma_coeff_delta_abs[sigFiltIdx][j])	
alf_luma_coeff_delta_sign[sigFiltIdx][j]	u(1)
}	
}	
}	
}	
if (alf_chroma_idc > 0) {	
alf_signal_new_filter_chroma	u(1)
if (alf_signal_new_filter_chroma){	
alf_chroma_min_eg_order_minus1	tu(v)
for(i = 0; i < 2; i++)	
alf_chroma_eg_order_increase_flag[i]	u(1)
for(j = 0; j < 6; j++) {	
alf_chroma_coeff_abs[j]	uek(v)
if(alf_chroma_coeff_abs[j] > 0)	
alf_chroma_coeff_sign[j]	u(1)
}	
}	
}	
}	

[0482] 7.3.4.2编解码树单元语法

[0483]

coding_tree_unit( ) {	**描述符**
xCtb = (CtbAddrInRs % PicWidthInCtbsY) << CtbLog2SizeY	
yCtb = (CtbAddrInRs / PicWidthInCtbsY) << CtbLog2SizeY	
if(slice_alf_enable_flag){	
alf_ctb_flag[0][xCtb >> Log2CtbSize][yCtb >> Log2CtbSize]	ae(v)
if (alf_ctb_flag[0][xCtb >> Log2CtbSize][yCtb >> Log2CtbSize] {	
coding_alf_ctu_filter_set_index(CtbAddrInRs)	
}	
if(alf_chroma_idc == 1 || alf_chroma_idc == 3)	
alf_ctb_flag[1][xCtb >> Log2CtbSize][yCtb >> Log2CtbSize]	ae(v)
if(alf_chroma_idc == 2 || alf_chroma_idc == 3)	
alf_ctb_flag[2][xCtb >> Log2CtbSize][yCtb >> Log2CtbSize]	ae(v)
}	
if(slice_type = = I && qtbtt_dual_tree_intra_flag) {	
dual_tree_implicit_qt_split (xCtb, yCtb, CtbLog2SizeY, 0)	
else	
coding_quadtree(xCtb, yCtb, CtbLog2SizeY, 0, SINGLE_TREE)	
}	

[0484]

coding_alf_ctu_filter_set_index(CtbAddrInRs, slice_type){	描述符
xCtb = (CtbAddrInRs % PicWidthInCtbsY) << CtbLog2SizeY	
yCtb = (CtbAddrInRs / PicWidthInCtbsY) << CtbLog2SizeY	
alf_use_new_filter	ae(1)
if (alf_use_new_filter == 0){	
alf_use_fixed_filter	ae(1)
}	
if (alf_use_new_filter){	
alf_luma_ctb_filter_set_index[xCtb >> Log2CtbSize][yCtb >> Log2CtbSize] = 16	
}	
else if (alf_use_fixed_filter){	
alf_fixed_filter_index	tb(v)
alf_luma_ctb_filter_set_index[xCtb >> Log2CtbSize][yCtb >> Log2CtbSize] = alf_fixed_filter_index	
}	
Else {{ if (alf_num_available_temporal_filter_sets_luma > 0) }} {	
alf_temporal_index	tb(v)
alf_luma_ctb_filter_set_index[xCtb >> Log2CtbSize][yCtb >> Log2CtbSize] = 16 + alf_temporal_index	
}	

[0485] 等于1的alf_signal_new_filter_luma指定新的亮度滤波器集被信令通知。等于0的alf_signal_new_filter_luma指定新的亮度滤波器集不被信令通知。当不存在时,它为0。

[0486] 等于1的alf_luma_use_fixed_filter_flag指定固定的滤波器集用于信令通知自适应环路滤波器。等于0的alf_luma_use_fixed_filter_flag指定固定的滤波器集不用于信令通知自适应环路滤波器。

[0487] alf_num_available_temporal_filter_sets_luma指定可以用于当前条带的可用时域滤波器集的数量,它可以来自0..[[5]]{{lumaALFSetSize}}。当不存在时,它为0。

[0488] {{约束alf_signal_new_filter_luma或alf_luma_use_fixed_filter_flag在alf_num_available_temporal_filter_sets_luma等于零时必须等于1。}}

[0489] 等于1的alf_signal_new_filter_chroma指定新的色度滤波器被信令通知。等于0的alf_signal_new_filter_chroma指定新的色度滤波器不被信令通知。

[0490] {{约束alf_signal_new_filter_chroma在chromaALFSetSize等于0时必须等于1。}}

[0491] 在一些实施例中,可以实施以下技术解决方案:

[0492] A1.一种用于视频处理的方法,包括:对视频的当前视频块执行滤波过程,其中该滤波过程使用滤波器系数并且包括具有至少一个中间结果的两个或更多个操作;对至少一个中间结果应用裁剪操作;以及基于至少一个中间结果来执行当前视频块和视频的比特流表示之间的转换,其中,至少一个中间结果基于滤波器系数和当前视频块的当前样点与当前样点的临近样点之间的差的加权和。

[0493] A2.根据解决方案A1所述的方法,还包括针对当前样点将当前样点的临近样点分类为多个组,其中,利用不同的参数对多个组中的每个组中的中间结果应用裁剪操作。

[0494] A3.根据解决方案A2所述的方法,其中,至少一个中间结果包括当前样点和多个组

中的每个组中的临近样点之间的差的加权平均值。

[0495] A4.根据解决方案A1所述的方法,其中,当前视频块的样点的多个临近样点共享滤波器系数,并且其中,对多个临近样点中的每一个应用一次裁剪操作。

[0496] A5.根据解决方案A4所述的方法,其中,多个临近样点中的至少两个临近样点的位置相对于当前视频块的样点对称。

[0497] A6.根据解决方案A4或A5所述的方法,其中,与滤波过程相关联的滤波器形状为对称模式。

[0498] A7.根据解决方案A4至A6中任一项所述的方法,其中,裁剪操作的一个或多个参数在比特流表示中信令通知。

[0499] A8.根据解决方案A1所述的方法,其中,当前视频块的样点包括N个临近样点,其中,对N个临近样点中的M1个临近样点应用一次裁剪操作,其中,M1和N是正整数,并且M1≤N。

[0500] A9.根据解决方案A1所述的方法,还包括针对当前视频块的样点将该样点的N个临近样点分类为M2个组,其中,对M2个组中的每一个应用一次裁剪操作,并且其中,M2和N是正整数。

[0501] A10.根据解决方案A1所述的方法,其中,对与当前视频块相关联的亮度分量应用裁剪操作。

[0502] A11.根据解决方案A1所述的方法,其中,对与当前视频块相关联的Cb分量或Cr分量应用裁剪操作。

[0503] A12.根据解决方案A1至A11中任一项所述的方法,其中,裁剪操作被定义为K(min,max,input),其中,input是裁剪操作的输入,min是裁剪操作的输出的标称最小值,并且max是裁剪操作的输出的标称最大值。

[0504] A13.根据解决方案A12所述的方法,其中,裁剪操作的输出的实际最大值小于标称最大值,并且其中,裁剪操作的输出的实际最小值大于标称最小值。

[0505] A14.根据解决方案A12所述的方法,其中,裁剪操作的输出的实际最大值等于标称最大值,并且其中,裁剪操作的输出的实际最小值大于标称最小值。

[0506] A15.根据解决方案A12所述的方法,其中,裁剪操作的输出的实际最大值小于标称最大值,并且其中,裁剪操作的输出的实际最小值等于标称最小值。

[0507] A16.根据解决方案A12所述的方法,其中,裁剪操作的输出的实际最大值等于标称最大值,并且其中,裁剪操作的输出的实际最小值等于标称最小值。

[0508] A17.根据解决方案A1所述的方法,其中,滤波过程包括配置有多个自适应环路滤波(ALF)滤波器系数集的ALF过程。

[0509] A18.根据解决方案A17所述的方法,其中,针对多个ALF滤波器系数集中的一个或多个预定义用于裁剪操作的至少一个参数。

[0510] A19.根据解决方案A17所述的方法,其中,用于裁剪操作的至少一个参数在包括当前视频块的片组、条带、图片或片的比特流表示中信令通知。

[0511] A20.根据解决方案A19所述的方法,其中,仅为与当前视频块相关联的一个或多个色彩分量信令通知至少一个参数。

[0512] A21.根据解决方案A17所述的方法,其中,多个ALF滤波器系数集中的至少一个和

用于裁剪操作的一个或多个参数被存储在相同的存储器位置中,并且其中,多个ALF滤波器系数集中的至少一个或者该一个或多个参数由包括当前视频块的编解码的编解码树单元(CTU)、编解码单元(CU)、片、片组、条带或图片继承。

[0513] A22.根据解决方案A21所述的方法,其中,裁剪操作被配置为在确定时域ALF系数集用于包括当前视频块的CTU、CU、片、片组、条带或图片的滤波过程时,使用对应于多个ALF滤波器系数集中的时域ALF系数集的一个或多个参数。

[0514] A23.根据解决方案A22所述的方法,其中,对应于时域ALF系数集的一个或多个参数仅用于与当前视频块相关联的一个或多个色彩分量。

[0515] A24.根据解决方案A21所述的方法,其中,在确定时域ALF系数集用于包括当前视频块的CTU、CU、片、片组、条带或图片的滤波过程时,对应于多个ALF滤波器系数集中的时域ALF系数集的一个或多个参数在比特流表示中信令通知。

[0516] A25.根据解决方案A24所述的方法,其中,仅为与当前视频块相关联的一个或多个色彩分量信令通知对应于时域ALF系数集的一个或多个参数。

[0517] A26.根据解决方案A21所述的方法,其中,信令通知与当前视频块相关联的第一色彩分量的一个或多个参数的第一参数集合,并且其中,继承与当前视频块相关联的第二色彩分量的一个或多个参数的第二参数集合。

[0518] A27.根据解决方案A1至A26中任一项所述的方法,其中,该转换从比特流表示生成当前视频块。

[0519] A28.根据解决方案A1至A26中任一项所述的方法,其中,该转换从当前视频块生成比特流表示。

[0520] A29.一种视频系统中的装置,包括处理器和其上具有指令的非暂时性存储器,其中该指令在由处理器执行时使得处理器实施根据解决方案A1至A28中任一项所述的方法。

[0521] A30.一种存储在非暂时性计算机可读介质上的计算机程序产品,该计算机程序产品包括用于执行根据解决方案A1至A28中任一项所述的方法的程序代码。

[0522] 在一些实施例中,可以实施以下技术解决方案:

[0523] B1.一种用于视频处理的方法,包括:将视频的当前视频块编码为视频的比特流表示,其中,当前视频块是用自适应环路滤波器(ALF)编解码的;以及基于一个或多个时域自适应滤波器集合的可用性或使用,在比特流表示中选择性地包括一个或多个时域自适应滤波器集合内的时域自适应滤波器集合的指示。

[0524] B2.根据解决方案B1所述的方法,其中,在时域自适应滤波器集合不可用的情况下,该集合的指示从比特流表示中排除。

[0525] B3.根据解决方案B1或B2所述的方法,其中,在时域自适应滤波器集合不是不可用的情况下,该集合的指示被包括在比特流表示中。4.根据解决方案B1至3中任一项所述的方法,其中,在一个或多个时域自适应滤波器集合都不可用的情况下,该指示从比特流表示中排除。

[0526] B5.根据解决方案B1至B3中任一项所述的方法,其中,在一个或多个时域自适应滤波器集合都不可用的情况下,使用固定滤波器的指示应当等于真。

[0527] B6.根据解决方案B1至B3中任一项所述的方法,其中,在一个或多个时域自适应滤波器集合都不可用的情况下,使用时域自适应滤波器的指示应当等于假。

[0528] B7.根据解决方案B1至B3中任一项所述的方法,其中,在一个或多个时域自适应滤波器集合都不可用的情况下,固定滤波器的索引的指示被包括在比特流表示中。

[0529] B8.一种用于视频处理的方法,包括:基于视频的比特流表示中的时域自适应滤波器集合的指示来确定一个或多个时域自适应滤波器集合的可用性或使用,其中该一个或多个时域自适应滤波器集合包括适用于用自适应环路滤波器(ALF)编解码的视频的当前视频块的时域自适应滤波器集合;以及通过基于该确定来选择性地应用时域自适应滤波器集合,从比特流表示生成解码的当前视频块。

[0530] B9.根据解决方案B8所述的方法,其中,在时域自适应滤波器集合不可用的情况下,该生成在不应用时域自适应滤波器集合的情况下被执行。

[0531] B10.根据解决方案B8或B9所述的方法,其中,在时域自适应滤波器集合不是不可用的情况下,执行该生成包括应用时域自适应滤波器集合。

[0532] B11.根据解决方案B1至B10中任一项所述的方法,其中,一个或多个时域自适应滤波器集合被包括在自适应参数集(APS)中,并且其中,该指示是APS索引。

[0533] B12.根据解决方案B1至B10中任一项所述的方法,还包括:基于不同方向上的梯度计算,确定一个或多个时域自适应滤波器集合中的至少一个时域自适应滤波器集合的滤波器索引。

[0534] B13.根据解决方案B1至B11中任一项所述的方法,还包括:确定一个或多个时域自适应滤波器集合都不可用,并且新的ALF系数集和固定的ALF系数集不用于包括当前视频块的编解码树块(CTB)、块、片组、片、条带或图片;以及基于该确定来推断自适应环路滤波被禁用。

[0535] B14.根据解决方案B1至B11中任一项所述的方法,其中,响应于一个或多个时域自适应滤波器集合中的至少一个不可用,比特流表示包括新的ALF系数集的使用的第一指示和固定的ALF系数集的使用的第二指示,并且其中,恰好第一指示和第二指示中的一个在比特流表示中为真。

[0536] B15.根据解决方案B14所述的方法,其中,比特流表示符合与ALF的操作相关联的格式规则。

[0537] B16.根据解决方案B1至B11中任一项所述的方法,其中,响应于一个或多个时域自适应滤波器集合都不可用,比特流表示包括ALF被启用并且新的ALF系数集和固定的ALF系数集不用于包括当前视频块的编解码树块(CTB)、块、片组、片、条带或图片的指示。

[0538] B17.根据解决方案B16所述的方法,其中,比特流表示不符合与ALF的操作相关联的格式规则。

[0539] B18.根据解决方案B1至B17中任一项所述的方法,其中,ALF被应用于与当前视频块相关联的一个或多个色彩分量。

[0540] B19.一种用于视频处理的方法,包括:基于可用的时域自适应环路滤波(ALF)系数集,为用自适应环路滤波器编解码的当前视频块确定多个时域ALF系数集,其中,可用的时域ALF系数集已经在该确定之前被编码或解码,并且其中,多个ALF系数集用于包括当前视频块的片组、片、条带、图片、编解码树块(CTB)或视频单元;以及基于多个时域ALF系数集,执行当前视频块和当前视频块的比特流表示之间的转换。

[0541] B20.根据解决方案B19所述的方法,其中,多个时域ALF系数集的最大数量被设置

为等于可用的时域ALF系数集的数量。

[0542] B21.根据解决方案B20所述的方法,其中,时域ALF系数集的数量被设置为等于可用的时域ALF系数集的数量和预定义数量N中的较小者,其中,N是整数,并且其中,N≥0。

[0543] B22.根据解决方案B21所述的方法,其中,N=5。

[0544] B23.一种视频处理的方法,包括:作为视频的当前视频块和视频的比特流表示之间的转换的一部分,处理一个或多个新的自适应环路滤波(ALF)系数集,其中,当前视频块是用自适应环路滤波器编解码的;以及在该处理之后,将一个或多个新的ALF系数集指定为可用的ALF系数集。

[0545] B24.根据解决方案B23所述的方法,还包括:遇到帧内随机访问点(IRAP)访问单元、IRAP图片、瞬时解码刷新(IDR)访问单元或IDR图片;以及基于该遇到,将可用的ALF系数集指定为不可用的ALF系数集。

[0546] B25.根据解决方案B23或B24所述的方法,其中,可用的ALF系数集中的至少一个是当前视频块之后的视频块的时域ALF系数集。

[0547] B26.根据解决方案B23至B25中任一项所述的方法,其中,在最大尺寸为N的ALF系数集列表中维护可用的ALF系数集,其中,N是整数。

[0548] B27.根据解决方案B26所述的方法,其中,ALF系数集列表按先进先出(FIFO)的顺序被维护。

[0549] B28.根据解决方案B1至B27中任一项所述的方法,其中,为与当前视频块相关联的每个时域层维护一个ALF系数集列表。

[0550] B29.根据解决方案B1至B27中任一项所述的方法,其中,为与当前视频块相关联的K个临近时域层维护一个ALF系数集列表。

[0551] B30.根据解决方案B1至B27中任一项所述的方法,其中,为包括当前视频块的当前图片维护第一ALF系数集列表,并且其中,为当前图片之后的图片维护第二ALF系数集列表。

[0552] B31.根据解决方案B30所述的方法,其中,当前图片之后的图片是基于当前图片预测的,并且其中,第一ALF系数集列表与第二ALF系数集列表相同。

[0553] B32.根据解决方案B30所述的方法,其中,当前图片是基于当前图片之后的图片和当前图片之前的图片预测的,并且其中,第一ALF系数集列表与第二ALF系数集列表相同。

[0554] B33.根据解决方案B23所述的方法,还包括:遇到帧内随机访问点(IRAP)访问单元、IRAP图片、瞬时解码刷新(IDR)访问单元或IDR图片;以及在该遇到之后,清空一个或多个ALF系数集列表。

[0555] B34.根据解决方案B23所述的方法,其中,为与当前视频块相关联的不同的色彩分量维护不同的ALF系数集列表。

[0556] B35.根据解决方案B34所述的方法,其中,不同的色彩分量包括亮度分量、Cr分量和Cb分量中的一个或多个。

[0557] B36.根据解决方案B23所述的方法,其中,为多个图片、片组、片、条带或编解码树单元(CTU)维护一个ALF系数集列表,并且其中,对于多个图片、片组、片、条带或编解码树单元(CTU)中的每一个,一个ALF系数集列表中的索引是不同的。

[0558] B37.根据解决方案B36所述的方法,其中,该索引为升序,并且基于与当前视频块相关联的第一时域层索引和与包括当前视频块的当前图片、片组、片、条带或编解码树单元

(CTU)相关联的第二时域层索引。

[0559] B38.根据解决方案B36所述的方法,其中,该索引为升序,并且基于与当前视频块相关联的图片顺序计数(POC)和与包括当前视频块的当前图片、片组、片、条带或编解码树单元(CTU)相关联的第二POC。

[0560] B39.根据解决方案B36所述的方法,其中,该索引包括分配给可用的ALF系数集的最小索引。

[0561] B40.根据解决方案B23所述的方法,其中,该转换包括裁剪操作,并且该方法还包括:针对当前视频块的样点,将样点的临近样点分类为多个组;以及将在比特流表示中信令通知的单个参数集合用于多个组中的每个组的裁剪操作。

[0562] B41.根据解决方案B23所述的方法,其中,该转换包括裁剪操作,并且其中,针对一个或多个新的ALF系数集预定义用于裁剪操作的参数集合。

[0563] B42.根据解决方案B23所述的方法,其中,该转换包括裁剪操作,并且其中,用于裁剪操作的参数集合在一个或多个新的ALF系数集的比特流表示中信令通知。

[0564] B43.一种用于视频处理的方法,包括:为视频的当前视频块和视频的比特流表示之间的转换确定视频的视频区域的头中的自适应环路滤波(ALF)的指示等于与比特流表示相关联的自适应参数集(APS)网络抽象层(NAL)单元中的ALF的指示;以及执行转换。

[0565] B44.根据解决方案B43所述的方法,其中,视频区域是图片。

[0566] B45.根据解决方案B43所述的方法,其中,视频区域是条带。

[0567] B46.一种用于视频处理的方法,包括:基于视频的视频区域使用的自适应环路滤波器的类型,对视频的当前视频块和视频的比特流表示之间的转换选择性地启用非线性自适应环路滤波(ALF)操作;以及在该选择性地启用之后执行转换。

[0568] B47.根据解决方案B46所述的方法,其中,视频区域是编解码树单元(CTU),并且其中,非线性ALF操作在确定自适应环路滤波器的类型包括固定的ALF集或时域ALF集时被禁用。

[0569] B48.根据解决方案B46所述的方法,其中,视频区域是条带、片组、片或编解码树单元(CTU),并且其中,非线性ALF操作在确定自适应环路滤波器的类型包括固定的ALF集时被启用。

[0570] B49.根据解决方案B46所述的方法,还包括:在比特流表示中选择性地信令通知用于非线性ALF操作的一个或多个裁剪参数。

[0571] B50.根据解决方案B49所述的方法,其中,一个或多个裁剪参数被信令通知。

[0572] B51.根据解决方案B49所述的方法,其中,为在比特流表示中信令通知的ALF滤波器系数集信令通知一个或多个裁剪参数。

[0573] B52.根据解决方案B49所述的方法,其中,一个或多个裁剪参数在确定自适应环路滤波器的类型包括固定的ALF集时被信令通知。

[0574] B53.根据解决方案B19至B52中任一项所述的方法,其中,该转换从比特流表示生成当前视频块。

[0575] B54.根据解决方案B19至B52中任一项所述的方法,其中,该转换从当前视频块生成比特流表示。

[0576] B55.一种视频系统中的装置,包括处理器和其上具有指令的非暂时性存储器,其

中该指令在由处理器执行时使得处理器实施根据解决方案B1至B54中任一项所述的方法。

[0577] B56.一种存储在非暂时性计算机可读介质上的计算机程序产品,该计算机程序产品包括用于执行根据解决方案B1至B54中任一项所述的方法的程序代码。

[0578] 在一些实施例中,可以实施以下技术解决方案:

[0579] C1.一种用于视频处理的方法,包括:对当前视频块执行滤波过程,其中该滤波过程包括具有至少一个中间结果的两个或更多个操作;对至少一个中间结果应用裁剪操作;以及基于滤波操作,执行当前视频块到当前视频块的比特流表示之间的转换。

[0580] C2.根据解决方案C1所述的方法,还包括针对当前视频块的样点将样点的临近样点分类为多个组,其中,利用不同的参数对多个组中的每个组中的中间结果应用裁剪操作。

[0581] C3.根据解决方案C2所述的方法,其中,至少一个中间结果包括当前样点和多个组中的每个组中的临近样点之间的差的加权平均值。

[0582] C4.根据解决方案C2所述的方法,其中,滤波过程使用滤波器系数,并且其中,至少一个中间结果包括滤波器系数和当前样点与临近样点之间的差的加权和。

[0583] C5.根据解决方案C1所述的方法,其中,当前视频块的样点的多个临近样点共享滤波器系数,并且其中,对多个临近样点中的每一个应用一次裁剪操作。

[0584] C6.根据解决方案C5所述的方法,其中,与滤波操作相关联的滤波器形状为对称模式。

[0585] C7.根据解决方案C5或C6所述的方法,其中,裁剪操作的一个或多个参数在比特流表示中信令通知。

[0586] C8.根据解决方案C1至C7中任一项所述的方法,其中,裁剪操作被定义为K(min,max,input),其中,input是裁剪操作的输入,min是裁剪操作的输出的标称最小值,并且max是裁剪操作的输出的标称最大值。

[0587] C9.根据解决方案C8所述的方法,其中,裁剪操作的输出的实际最大值小于标称最大值,并且其中,裁剪操作的输出的实际最小值大于标称最小值。

[0588] C10.根据解决方案C8所述的方法,其中,裁剪操作的输出的实际最大值等于标称最大值,并且其中,裁剪操作的输出的实际最小值大于标称最小值。

[0589] C11.根据解决方案C8所述的方法,其中,裁剪操作的输出的实际最大值小于标称最大值,并且其中,裁剪操作的输出的实际最小值等于标称最小值。

[0590] C12.根据解决方案C8所述的方法,其中,裁剪操作的输出的实际最大值等于标称最大值,并且其中,裁剪操作的输出的实际最小值等于标称最小值。

[0591] C13.一种用于视频处理的方法,包括,基于时域自适应环路滤波系数集的不可用性,执行当前视频块和当前视频块的比特流表示之间的转换,使得比特流表示省略时域自适应环路滤波系数集的指示。

[0592] C14.根据解决方案C13所述的方法,还包括:确定新的自适应环路滤波(ALF)系数和固定的ALF系数不用于包括当前视频块的编解码树块(CTB)、块、片组、片、条带或图片;以及推断自适应环路滤波被禁用。

[0593] C15.根据解决方案C13所述的方法,其中,一致性比特流包括新的自适应环路滤波(ALF)系数的指示或固定的ALF系数的指示。

[0594] C16.一种用于视频处理的方法,包括:基于可用的时域自适应环路滤波(ALF)系数

集,为当前视频块确定一个或多个时域ALF系数集,其中,可用的时域ALF系数集已经在该确定之前被编码或解码;以及基于一个或多个时域ALF系数集,执行当前视频块和当前视频块的比特流表示之间的转换。

[0595] C17.根据解决方案C16所述的方法,其中,一个或多个时域ALF系数集的最大数量为$ALF_{available}$。

[0596] C18.根据解决方案C17所述的方法,其中,一个或多个时域ALF系数集的数量为min(N,$ALF_{available}$),其中,N是整数,并且其中,N≥0。

[0597] C19.根据解决方案C18所述的方法,其中,N=5。

[0598] C20.一种视频处理的方法,包括:为当前视频块处理一个或多个新的自适应环路滤波(ALF)系数集;在该处理之后,将一个或多个新的ALF系数集指定为可用的ALF系数集;以及基于可用的ALF系数集,执行当前视频块和当前视频块的比特流表示之间的转换。

[0599] C21.根据解决方案C20所述的方法,还包括:遇到帧内随机访问点(IRAP)访问单元、IRAP图片、瞬时解码刷新(IDR)访问单元或IDR图片;以及将可用的ALF系数集指定为不可用的ALF系数集。

[0600] C22.根据解决方案C20或C21所述的方法,其中,可用的ALF系数集是当前视频块之后的视频块的时域ALF系数集。

[0601] C23.根据解决方案C20至C22中任一项所述的方法,其中,在最大尺寸为N的ALF系数集列表中维护可用的ALF系数集,其中,N是整数。

[0602] C24.根据解决方案C23所述的方法,其中,ALF系数集列表按先进先出(FIFO)的顺序被维护。

[0603] C25.根据解决方案C13至C24中任一项所述的方法,其中,为与当前视频块相关联的每个时域层维护一个ALF系数集列表。

[0604] C26.根据解决方案C13至C24中任一项所述的方法,其中,为与当前视频块相关联的K个临近时域层维护一个ALF系数集列表。

[0605] C27.根据解决方案C13至C24中任一项所述的方法,其中,为包括当前视频块的当前图片维护第一ALF系数集列表,并且其中,为当前图片之后的图片维护第二ALF系数集列表。

[0606] C28.根据解决方案C27所述的方法,其中,当前图片之后的图片是基于当前图片预测的,并且其中,第一ALF系数集列表与第二ALF系数集列表相同。

[0607] C29.根据解决方案C20所述的方法,还包括:遇到帧内随机访问点(IRAP)访问单元、IRAP图片、瞬时解码刷新(IDR)访问单元或IDR图片;以及在该遇到之后,清空一个或多个ALF系数集列表。

[0608] C30.根据解决方案C20所述的方法,其中,为当前视频块的不同的色彩分量维护不同的ALF系数集列表。

[0609] C31.一种视频系统中的装置,包括处理器和其上具有指令的非暂时性存储器,其中该指令在由处理器执行时使得处理器实施根据解决方案C1至C30中任一项所述的方法。

[0610] C32.一种存储在非暂时性计算机可读介质上的计算机程序产品,该计算机程序产品包括用于执行根据解决方案C1至C30中任一项所述的方法的程序代码。

[0611] 图12是视频处理装置1200的框图。装置1200可以用于实施本文描述的一种或多种

方法。装置1200可以体现在智能电话、平板电脑、计算机、物联网(Internet of Things,IoT)接收器等中。装置1200可以包括一个或多个处理器1202、一个或多个存储器1204、以及视频处理硬件1206。(多个)处理器1202可以被配置为实施本文档中描述的一种或多种方法(包括但不限于方法1100和1150)。存储器(多个存储器)1204可以用于存储用于实施本文描述的方法和技术的数据和代码。视频处理硬件1206可以用于以硬件电路实施本文档中描述的一些技术。

[0612] 在一些实施例中,可以使用如关于图12所述的在硬件平台上实施的装置来实施视频编解码方法。

[0613] 所公开的技术的一些实施例包括做出启用视频处理工具或模式的决定或确定。在示例中,当视频处理工具或模式被启用时,编解码器将在视频块的处理中使用或实施该工具或模式,但可能不一定基于该工具或模式的使用来修改产生的比特流。也就是说,当基于决定或确定来启用视频处理工具或模式时,从视频的块到视频的比特流表示的转换将使用该视频处理工具或模式。在另一示例中,当视频处理工具或模式被启用时,解码器将在知道比特流已经基于视频处理工具或模式进行修改的情况下处理比特流。也就是说,将使用基于决定或确定而启用的视频处理工具或模式来执行从视频的比特流表示到视频的块的转换。

[0614] 所公开的技术的一些实施例包括做出禁用视频处理工具或模式的决定或确定。在示例中,当视频处理工具或模式被禁用时,编解码器将不在视频的块到视频的比特流表示的转换中使用该工具或模式。在另一示例中,当视频处理工具或模式被禁用时,解码器将在知道比特流没有使用基于决定或确定而启用的视频处理工具或模式进行修改的情况下处理比特流。

[0615] 图13是示出可以在其中实施本文公开的各种技术的示例视频处理系统1300的框图。各种实施方式可以包括系统1300的一些或所有组件。系统1300可以包括用于接收视频内容的输入1302。视频内容可以以例如8或10比特多分量像素值的原始或未压缩格式而接收,或者可以是压缩或编解码格式。输入1302可以表示网络接口、外围总线接口或存储接口。网络接口的示例包括诸如以太网、无源光网络(Passive Optical Network,PON)等的有线接口和诸如Wi-Fi或蜂窝接口的无线接口。

[0616] 系统1300可以包括可以实施本文档中描述的各种编解码方法的编解码组件1304。编解码组件1304可以将来自输入1302的视频的平均比特率减小到编解码组件1304的输出,以产生视频的编解码表示。编解码技术因此有时被称为视频压缩或视频转码技术。编解码组件1304的输出可以被存储,或者经由如由组件1306表示的通信连接而发送。在输入1302处接收的视频的存储或通信传送的比特流(或编解码)表示可以由组件1308用于生成像素值或传送到显示接口1310的可显示视频。从比特流表示生成用户可视视频的过程有时被称为视频解压缩。此外,虽然某些视频处理操作被称为“编解码”操作或工具,但是将理解,编解码工具或操作在编解码器处被使用,并且反转编解码结果的对应的解码工具或操作将由解码器执行。

[0617] 外围总线接口或显示接口的示例可以包括通用串行总线(Universal Serial Bus,USB)、或高清晰度多媒体接口(High Definition Multimedia Interface,HDMI)、或显示端口(Displayport)等。存储接口的示例包括SATA(Serial Advanced Technology

Attachment,串行高级技术附件)、PCI、IDE接口等。本文档中描述的技术可以体现在各种电子设备中,诸如移动电话、膝上型电脑、智能电话、或能够执行数字数据处理和/或视频显示的其他设备。

[0618] 从前面可以理解,为了说明的目的,本文已经描述了当前公开的技术的特定实施例,但是在不脱离本发明的范围的情况下,可以进行各种修改。因此,当前公开的技术不受除了所附权利要求之外的限制。

[0619] 本专利文档中描述的主题和功能操作的实施方式可以在各种系统、数字电子电路中被实施,或者在计算机软件、固件或硬件(包括本说明书中公开的结构及其结构等同物)中被实施,或者在它们中的一个或多个的组合中被实施。本说明书中描述的主题的实施方式可以被实施为一个或多个计算机程序产品,即编解码在有形和非暂时性计算机可读介质上的计算机程序指令的一个或多个模块,该计算机程序指令用于由数据处理装置执行或控制数据处理装置的操作。计算机可读介质可以是机器可读存储设备、机器可读存储基板、存储器设备、影响机器可读传播信号的物质的组合、或它们中的一个或多个的组合。术语“数据处理单元”或“数据处理装置”包含用于处理数据的所有装置、设备和机器,包括例如可编程处理器、计算机、或多个处理器或计算机。除了硬件之外,装置还可以包括为所讨论的计算机程序创建执行环境的代码,例如,构成处理器固件、协议栈、数据库管理系统、操作系统、或它们中的一个或多个的组合的代码。

[0620] 计算机程序(也已知为程序、软件、软件应用、脚本或代码)可以以任何形式的编程语言(包括编译或解释语言)编写,并且其可以以任何形式部署,包括作为独立程序或作为适合在计算环境中使用的模块、组件、子例程或其他单元。计算机程序不一定对应于文件系统中的文件。程序可以存储在保存其他程序或数据(例如,存储在标记语言文档中的一个或多个脚本)的文件的一部分中,存储在专用于所讨论的程序的单个文件中,或存储在多个协调文件中(例如,存储一个或多个模块、子程序或代码部分的文件)。计算机程序可以被部署以在一个计算机上或在位于一个站点上或跨多个站点分布并通过通信网络互连的多个计算机上执行。

[0621] 本说明书中描述的过程和逻辑流程可以由执行一个或多个计算机程序的一个或多个可编程处理器执行,以通过对输入数据进行操作并生成输出来执行功能。过程和逻辑流程也可以由专用逻辑电路执行,并且装置也可以被实施为专用逻辑电路,例如,FPGA(Field Programmable Gate Array,现场可编程门阵列)或ASIC(Application Specific Integrated Circuit,专用集成电路)。

[0622] 适合于执行计算机程序的处理器包括例如通用和专用微处理器、以及任何类型的数字计算机的任何一个或多个处理器。通常,处理器将从只读存储器或随机存取存储器或两者接收指令和数据。计算机的基本元件是用于执行指令的处理器和用于存储指令和数据的一个或多个存储器设备。通常,计算机还将包括用于存储数据的一个或多个大容量存储设备(例如,磁盘、磁光盘或光盘),或可操作地耦合以从该一个或多个大容量存储设备接收数据或向该一个或多个大容量存储设备传递数据、或者从其接收数据并向其传递数据。然而,计算机不需要这样的设备。适用于存储计算机程序指令和数据的计算机可读介质包括所有形式的非易失性存储器、介质和存储器设备,包括例如半导体存储器设备,例如EPROM、EEPROM和闪存设备。处理器和存储器可以由专用逻辑电路补充或并入专用逻辑电路中。

[0623] 虽然本专利文档包含许多细节,但这些细节不应被解释为对任何发明或可能要求保护的范围的限制,而是作为特定于特定发明的特定实施例的特征的描述。在本专利文档中在单独的实施例的上下文中描述的某些特征也可以在单个实施例中组合实施。相反,在单个实施例的上下文中描述的各种特征也可以在多个实施例中分开实施或以任何合适的子组合实施。此外,尽管特征可以在上面描述为以某些组合起作用并且甚至最初如此要求保护,但是在一些情况下可以从组合排除来自所要求保护的组合的一个或多个特征,并且所要求保护的组合可以针对子组合或子组合的变化。

[0624] 类似地,虽然在附图中以特定顺序描绘了操作,但是这不应该被理解为需要以所示的特定顺序或以先后顺序执行这样的操作或者执行所有示出的操作以实现期望的结果。此外,在本专利文档中描述的实施例中的各种系统组件的分离不应被理解为在所有实施例中都需要这样的分离。

[0625] 仅描述了一些实施方式和示例,并且可以基于本专利文档中描述和示出的内容来进行其他实施方式、增强和变化。

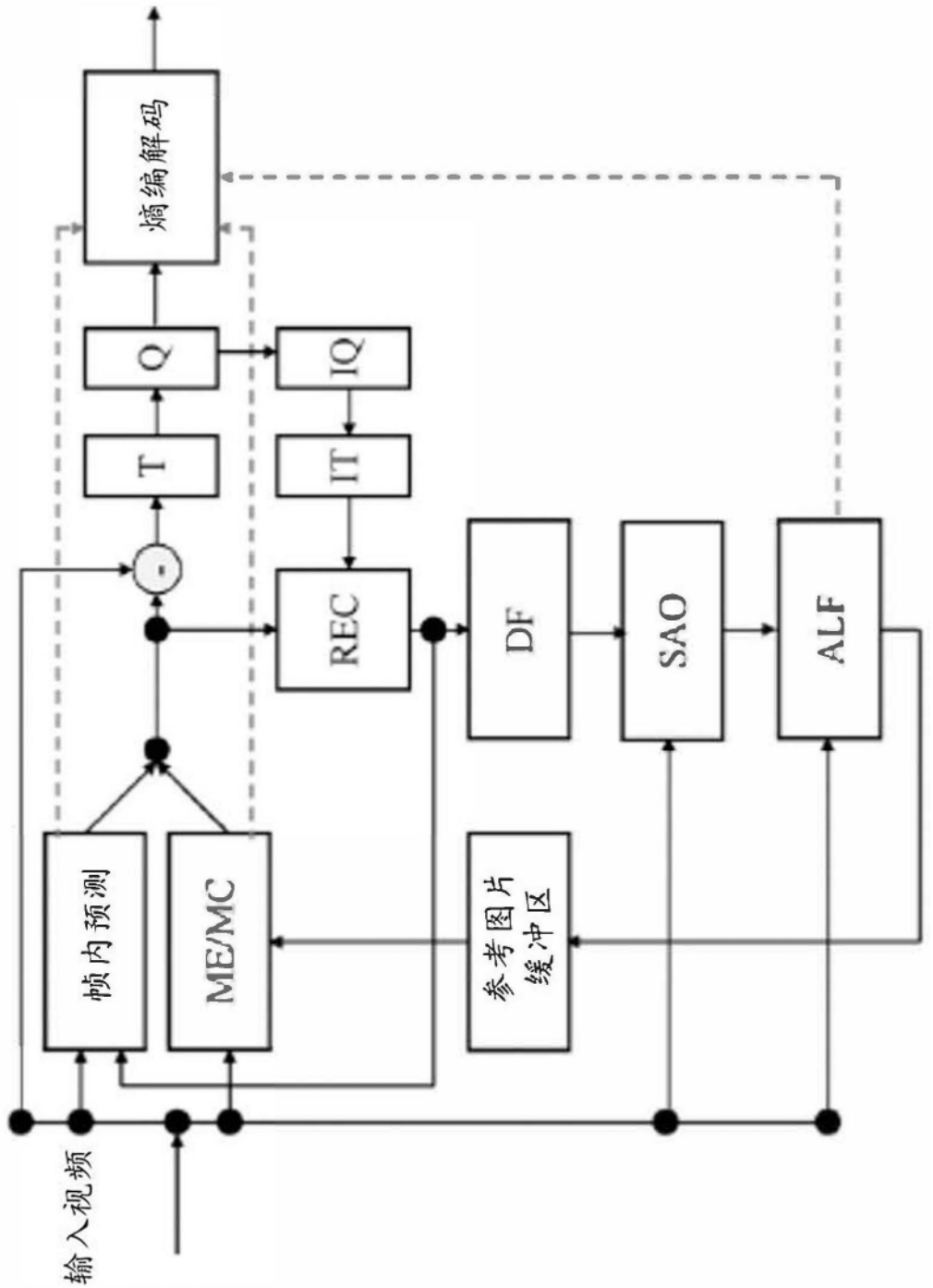


图1

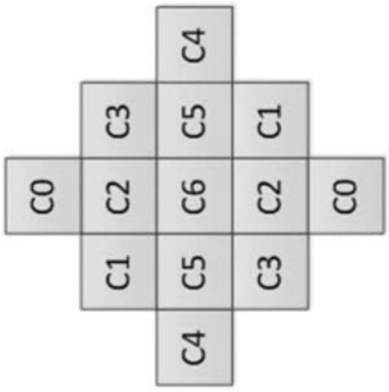

C4
C3
C5
C1
C0
C2
C6
C2
C0
C1
C5
C3
C4

图2A

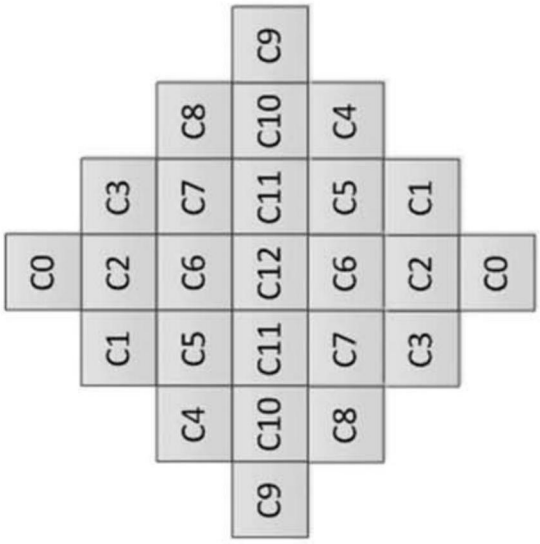

C9
C8
C10
C4
C3
C7
C11
C5
C1
C0
C2
C6
C12
C6
C2
C0
C1
C5
C11
C7
C3
C4
C10
C8
C9

图2B

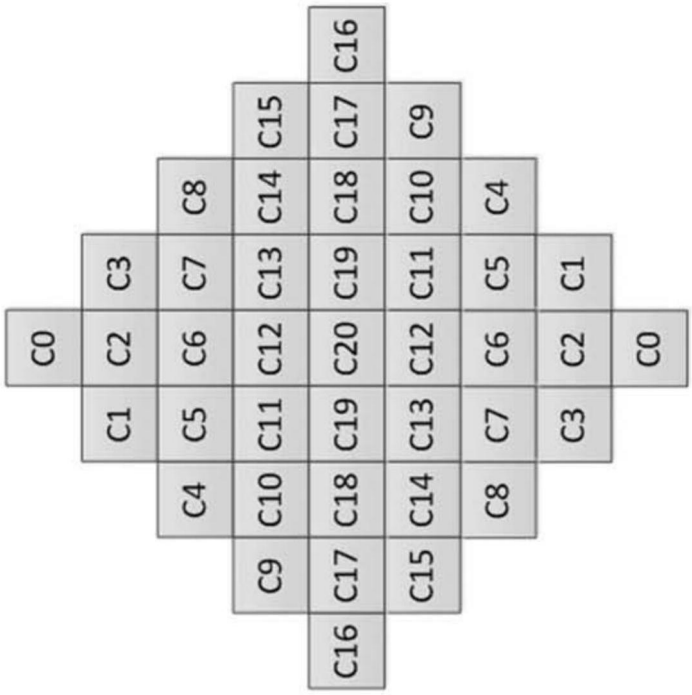

C16
C15 C17 C9
C8 C14 C18 C10 C4
C3 C7 C13 C19 C11 C5 C1
C0 C2 C6 C12 C20 C12 C6 C2 C0
C1 C5 C11 C19 C13 C7 C3
C4 C10 C18 C14 C8
C9 C17 C15
C16

图2C

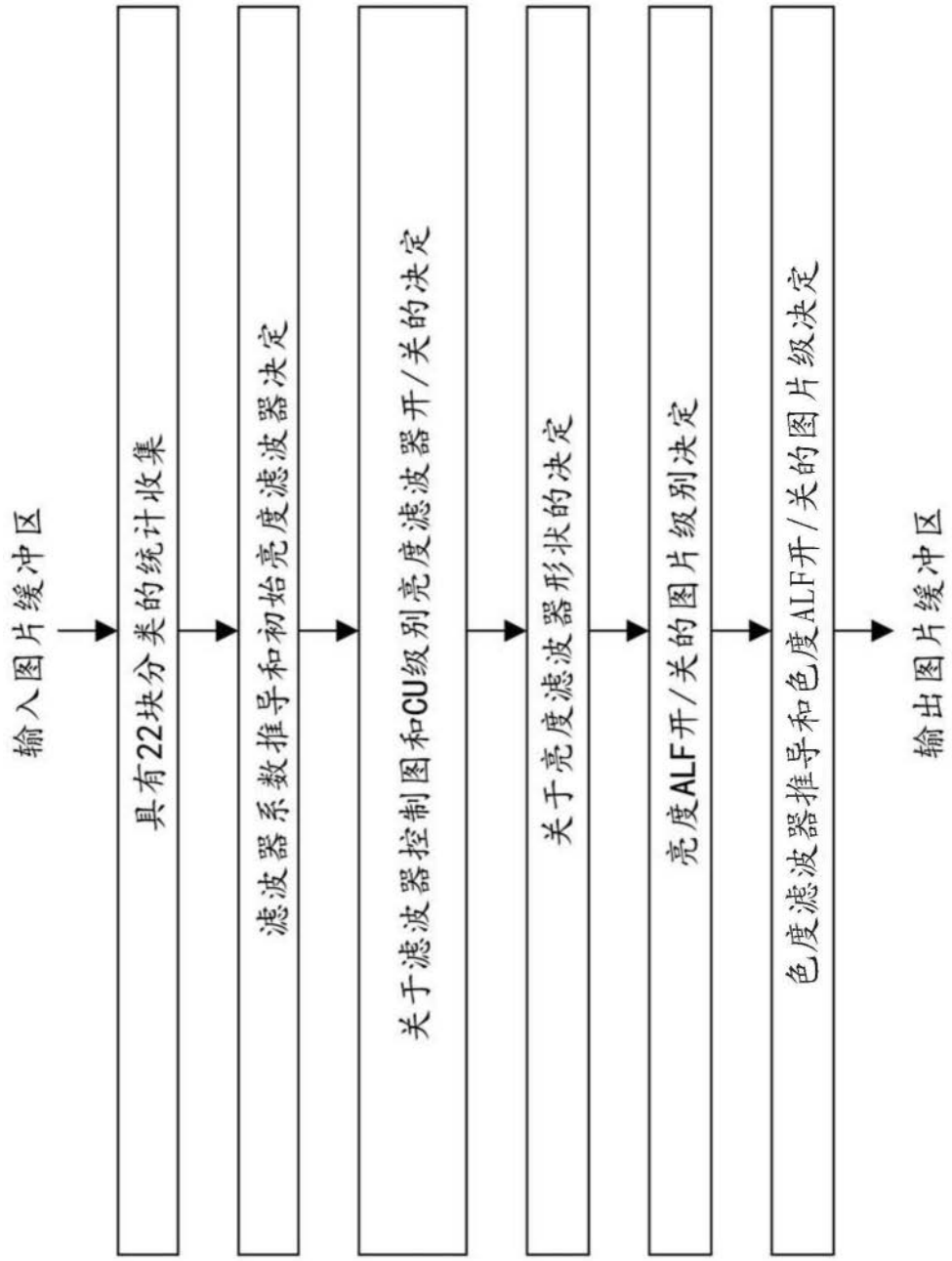

输入图片缓冲区
具有22块分类的统计收集
滤波器系数推导和初始亮度滤波器决定
关于滤波器控制图和CU级别亮度滤波器开/关的决定
关于亮度滤波器形状的决定
亮度ALF开/关的图片级别决定
色度滤波器推导和色度ALF开/关的图片级决定
输出图片缓冲区

图3

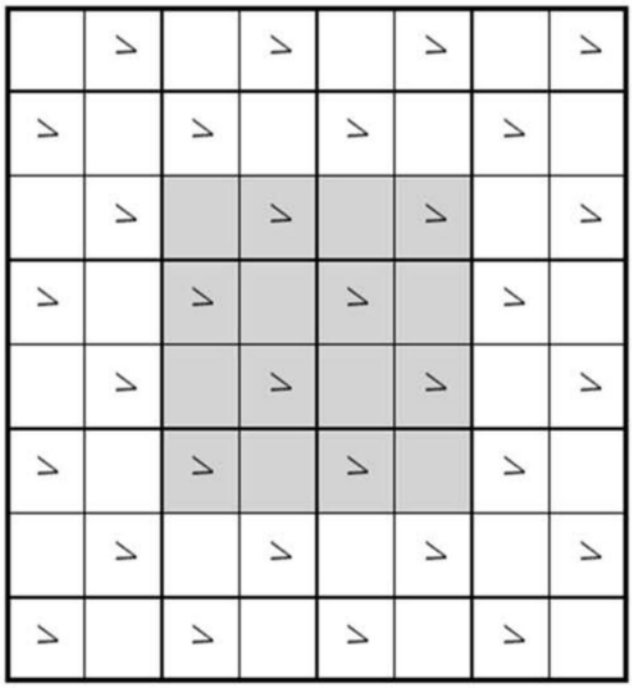


图4A

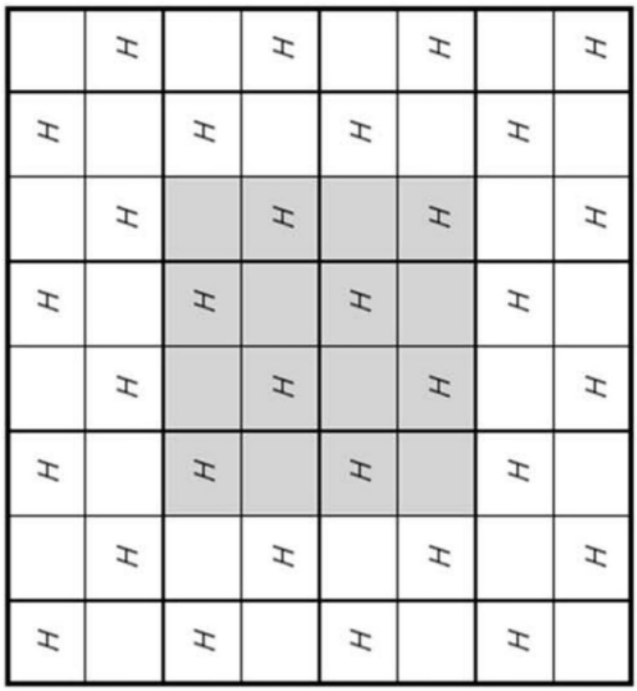


图4B

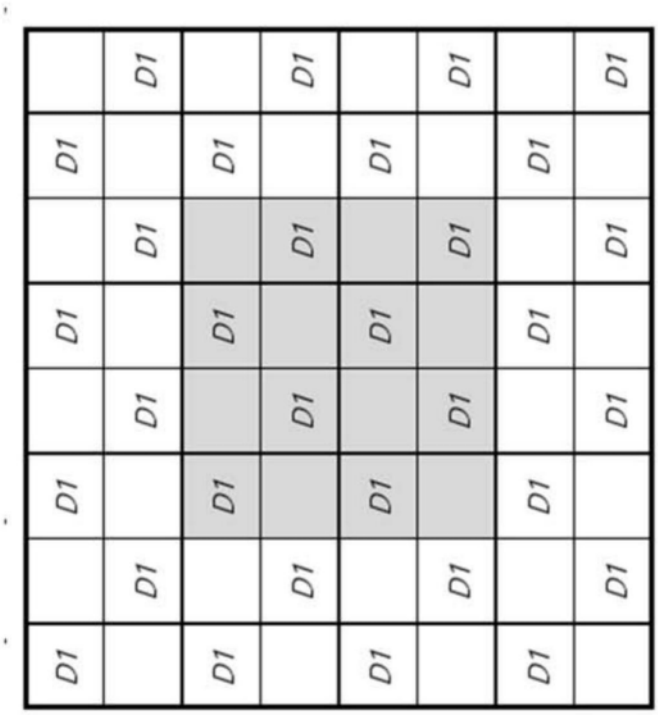


图4C

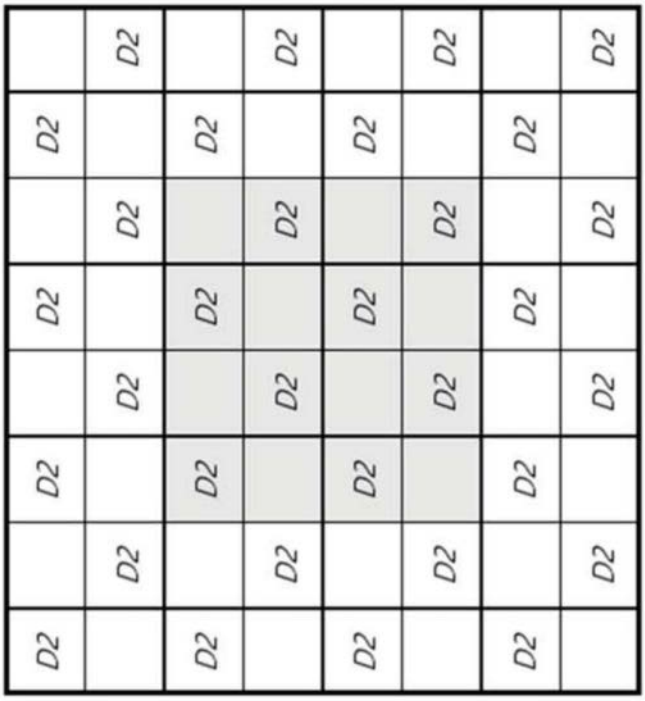


图4D

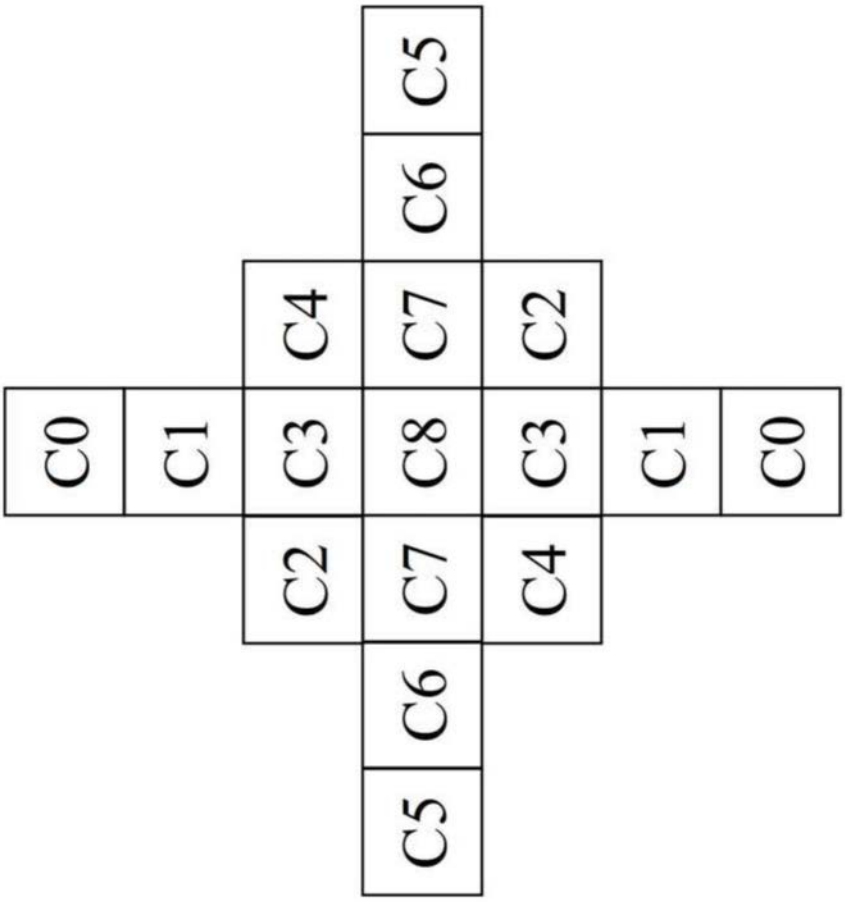


图5

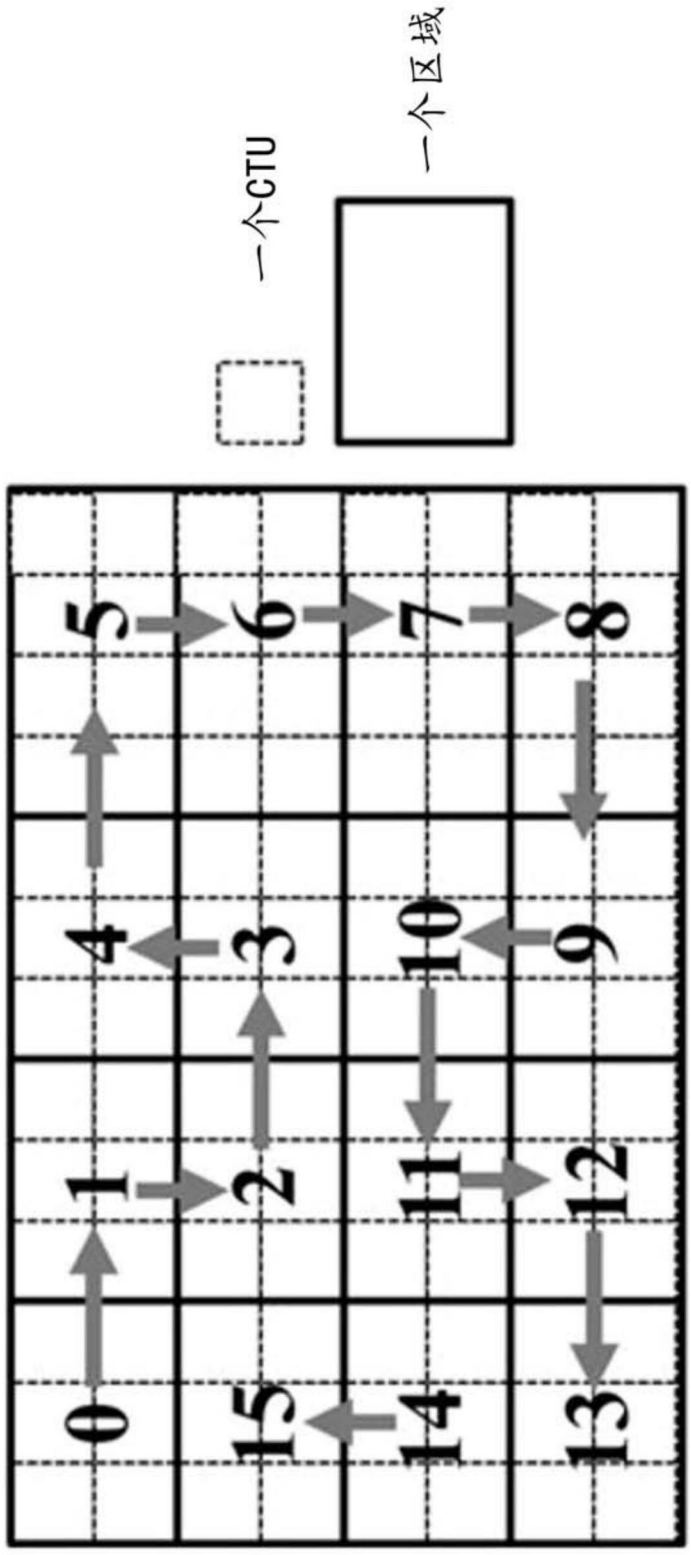


图6

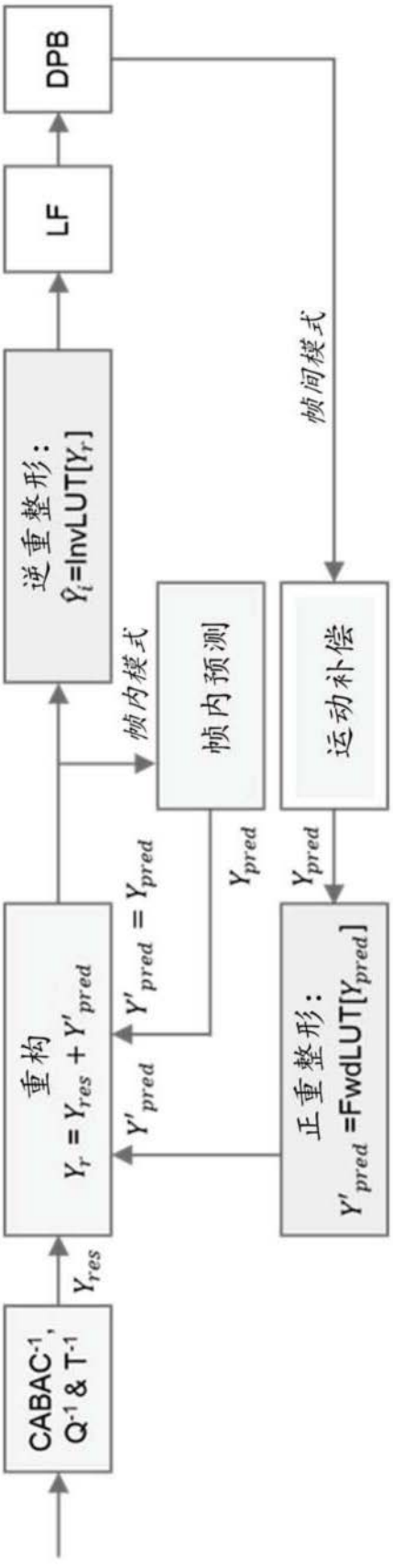

DPB
LF
逆重整形：
$\hat{Y}_i$=InvLUT[Y_r]
帧间模式
帧内模式
帧内预测
运动补偿
Y_{pred}
Y_{pred}
Y'_{pred} = Y_{pred}
重构
$Y_r = Y_{res} + Y'_{pred}$
Y'_{pred}
正重整形：
Y'_{pred} =FwdLUT[Y_{pred}]
Y_{res}
CABAC^{-1},
Q^{-1} & T^{-1}

图7

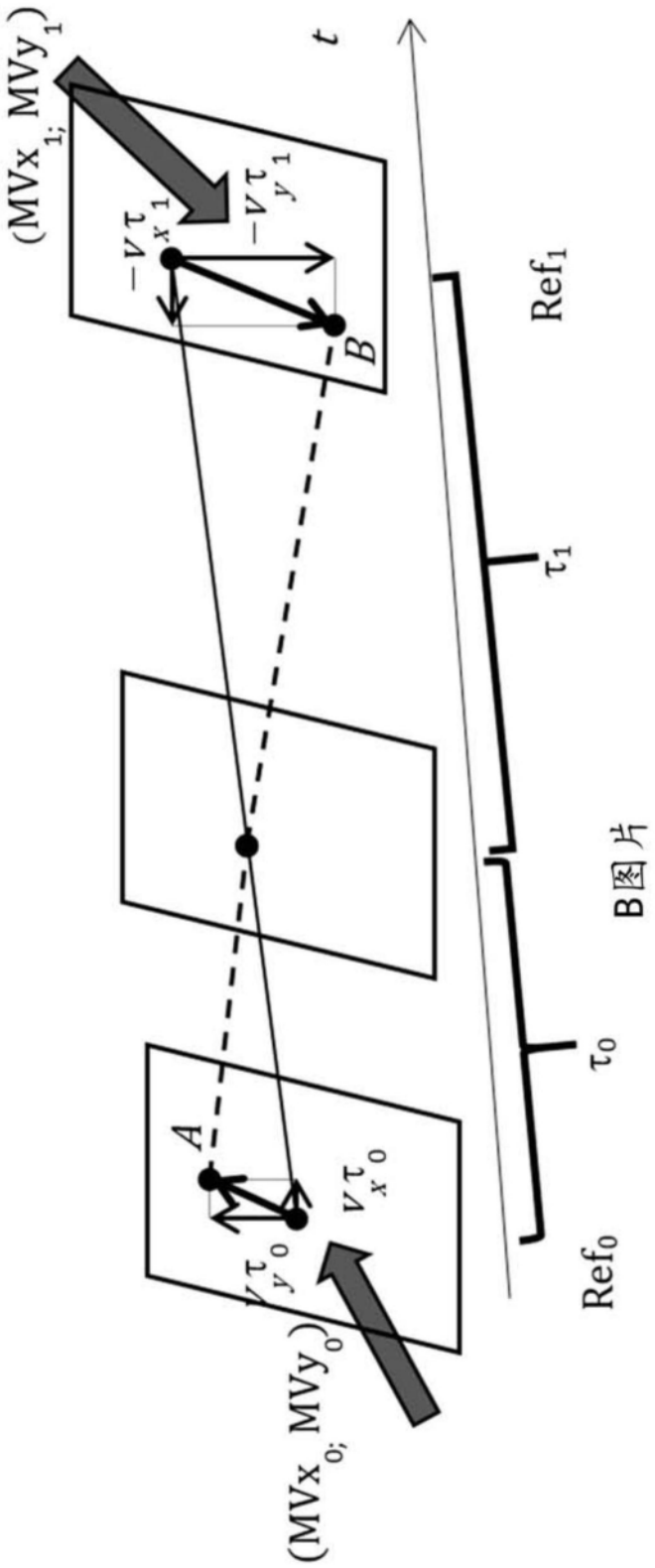

t
(MVx_1, MVy_1)
$-v\tau_{x1}$
$-v\tau_{y1}$
B
Ref_1
τ_1
B图片
τ_0
Ref_0
A
$v\tau_{x0}$
$v\tau_{y0}$
(MVx_0, MVy_0)

图8

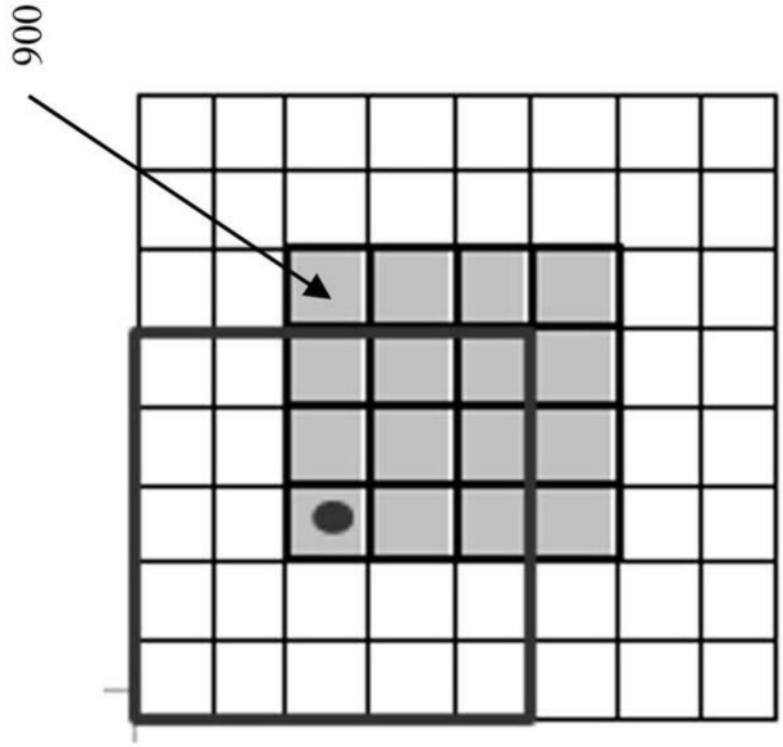


图9A

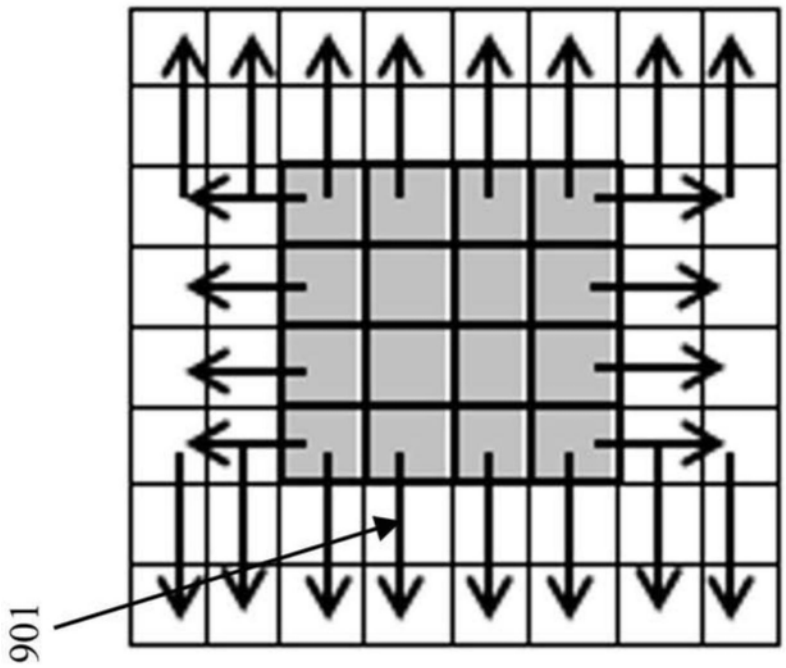


图9B

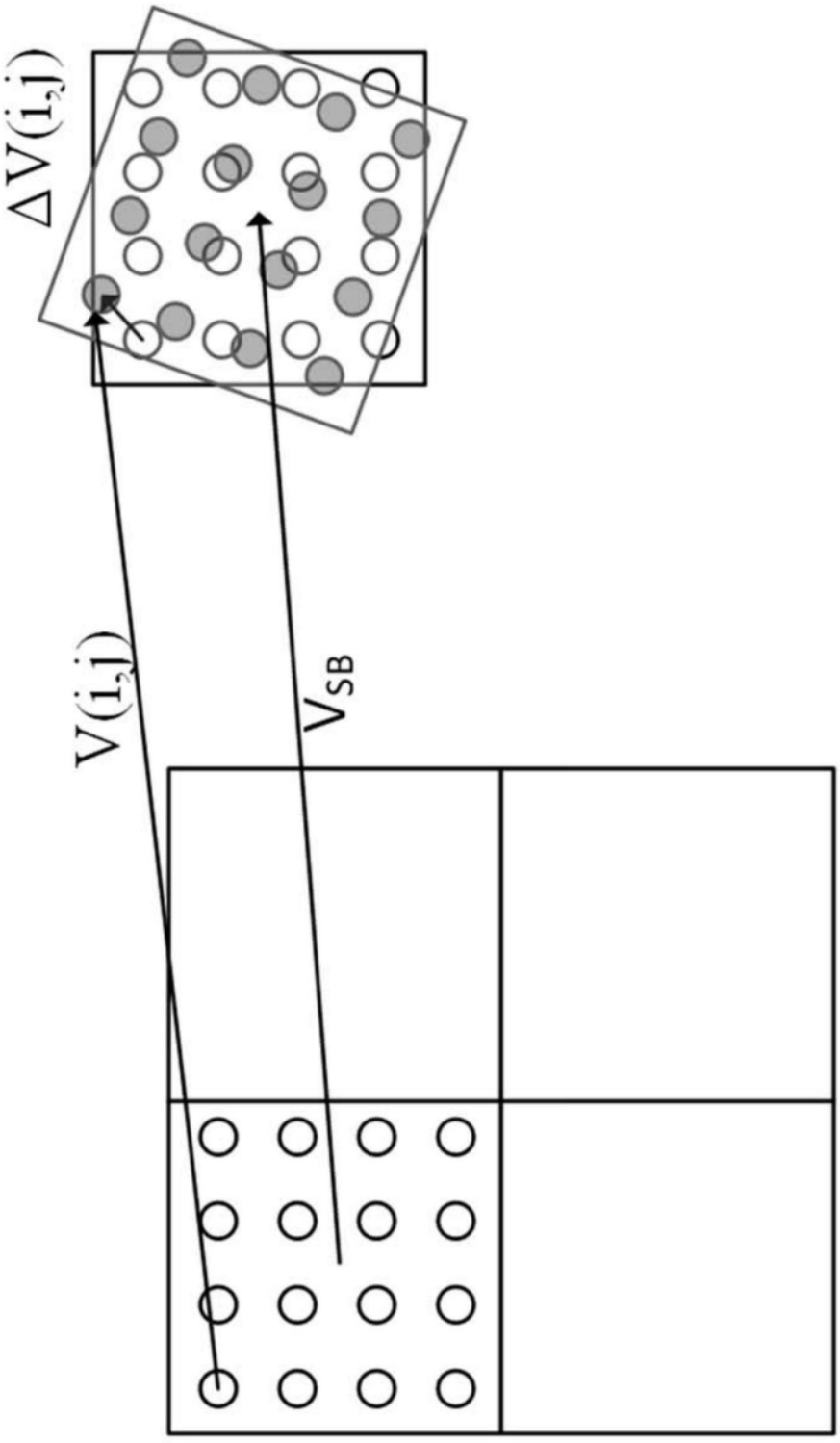


图10

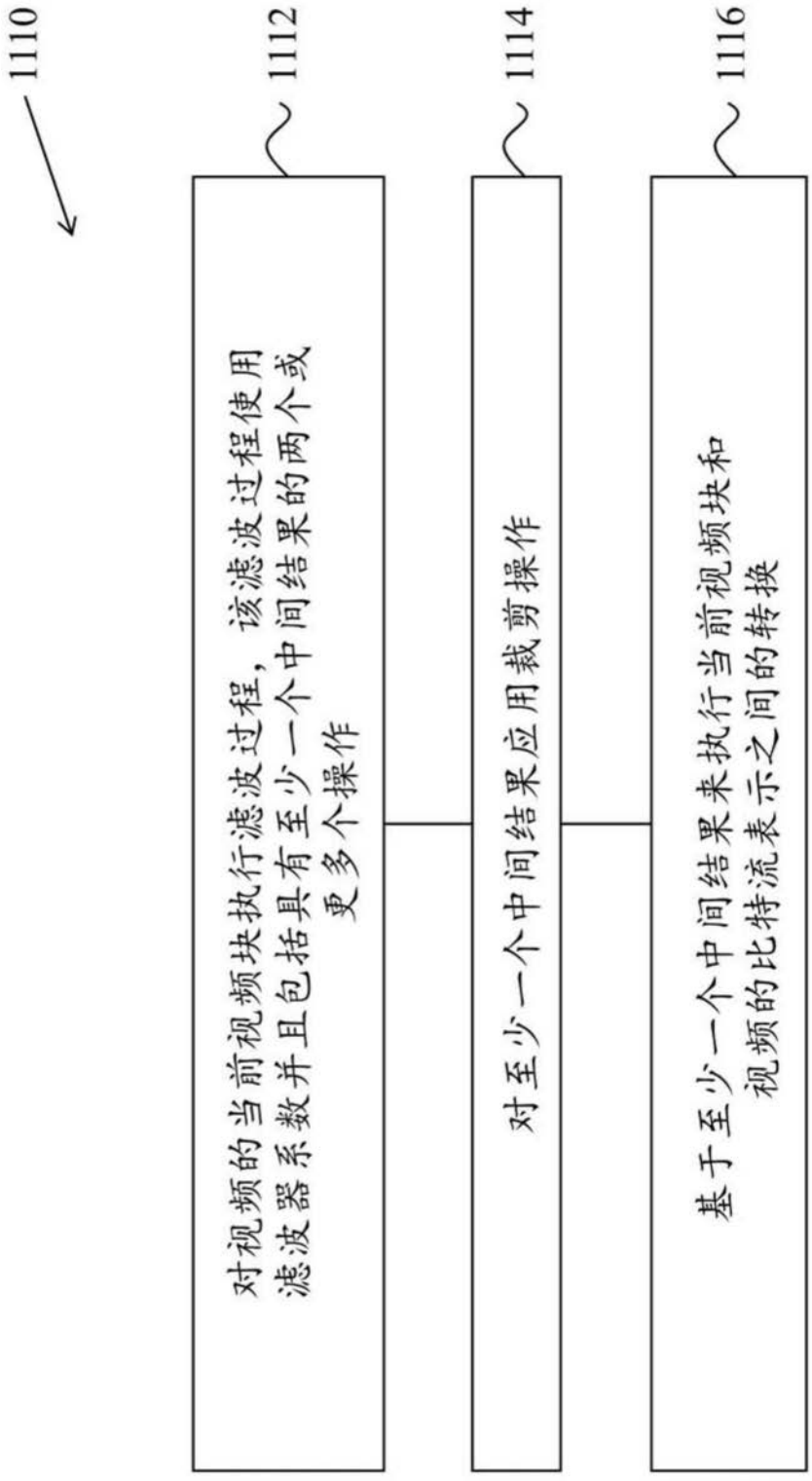

1110
1112
对视频的当前视频块执行滤波过程，该滤波过程使用滤波器系数并且包括具有至少一个中间结果的两个或更多个操作
1114
对至少一个中间结果应用裁剪操作
1116
基于至少一个中间结果来执行当前视频块和视频的比特流表示之间的转换

图11A

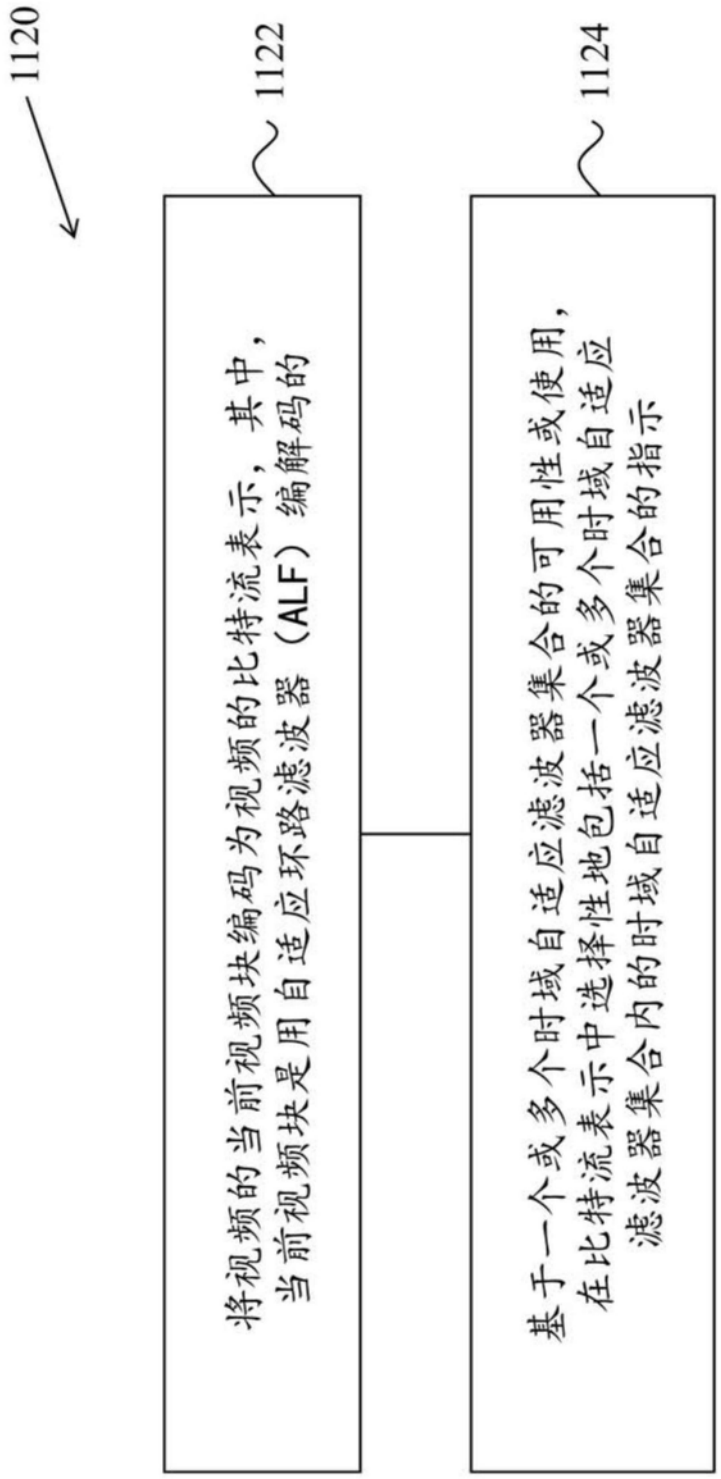

1120
1122
将视频的当前视频块编码为视频的比特流表示，其中，当前视频块是用自适应环路滤波器（ALF）编解码的
1124
基于一个或多个时域自适应滤波器集合的可用性或使用，在比特流表示中选择性地包括一个或多个时域自适应滤波器集合内的时域自适应滤波器集合的指示

图11B

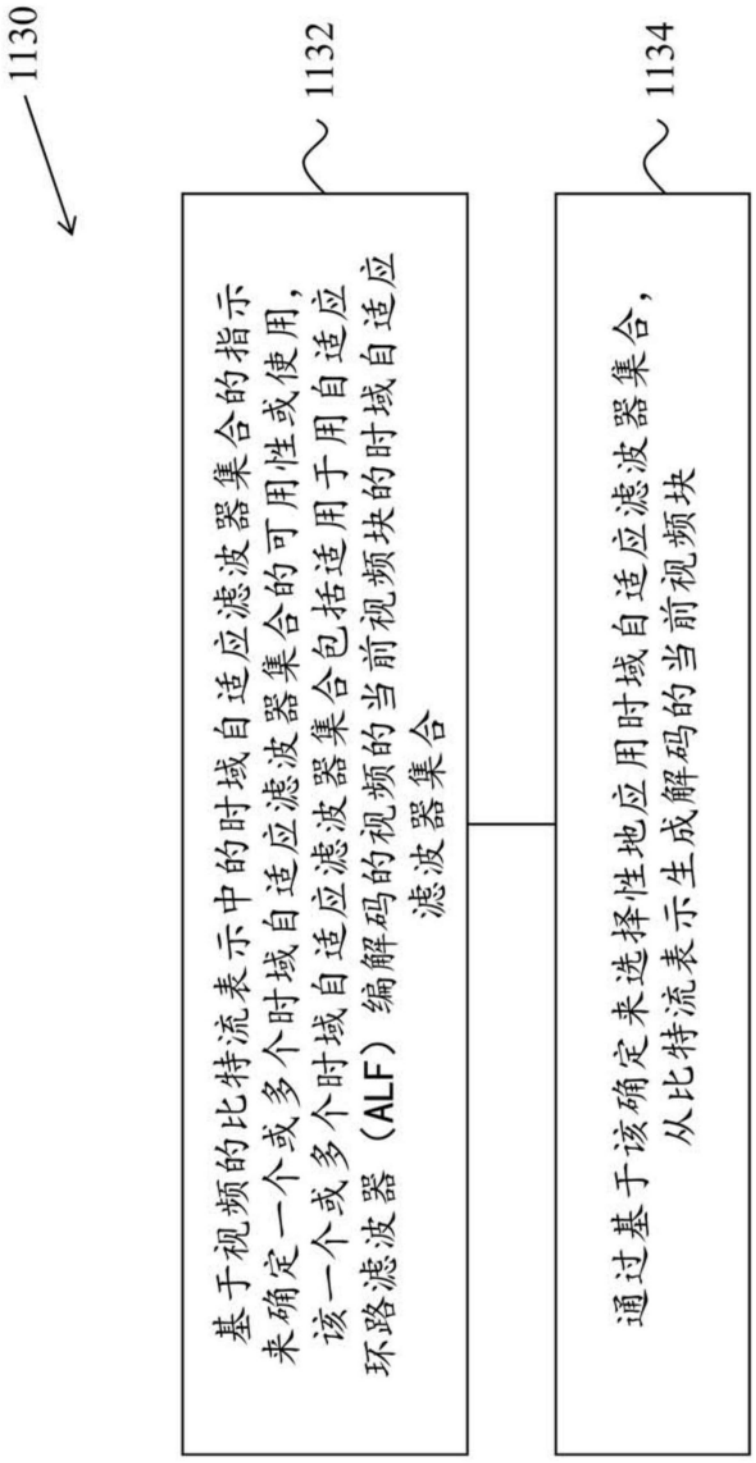

1130
1132
基于视频的比特流表示中的时域自适应滤波器集合的指示来确定一个或多个时域自适应滤波器集合的可用性或使用，该一个或多个时域自适应滤波器集合包括适用于用自适应环路滤波器（ALF）编解码的视频的当前视频块的时域自适应滤波器集合
1134
通过基于该确定来选择性地应用时域自适应滤波器集合，从比特流表示生成解码的当前视频块

图11C

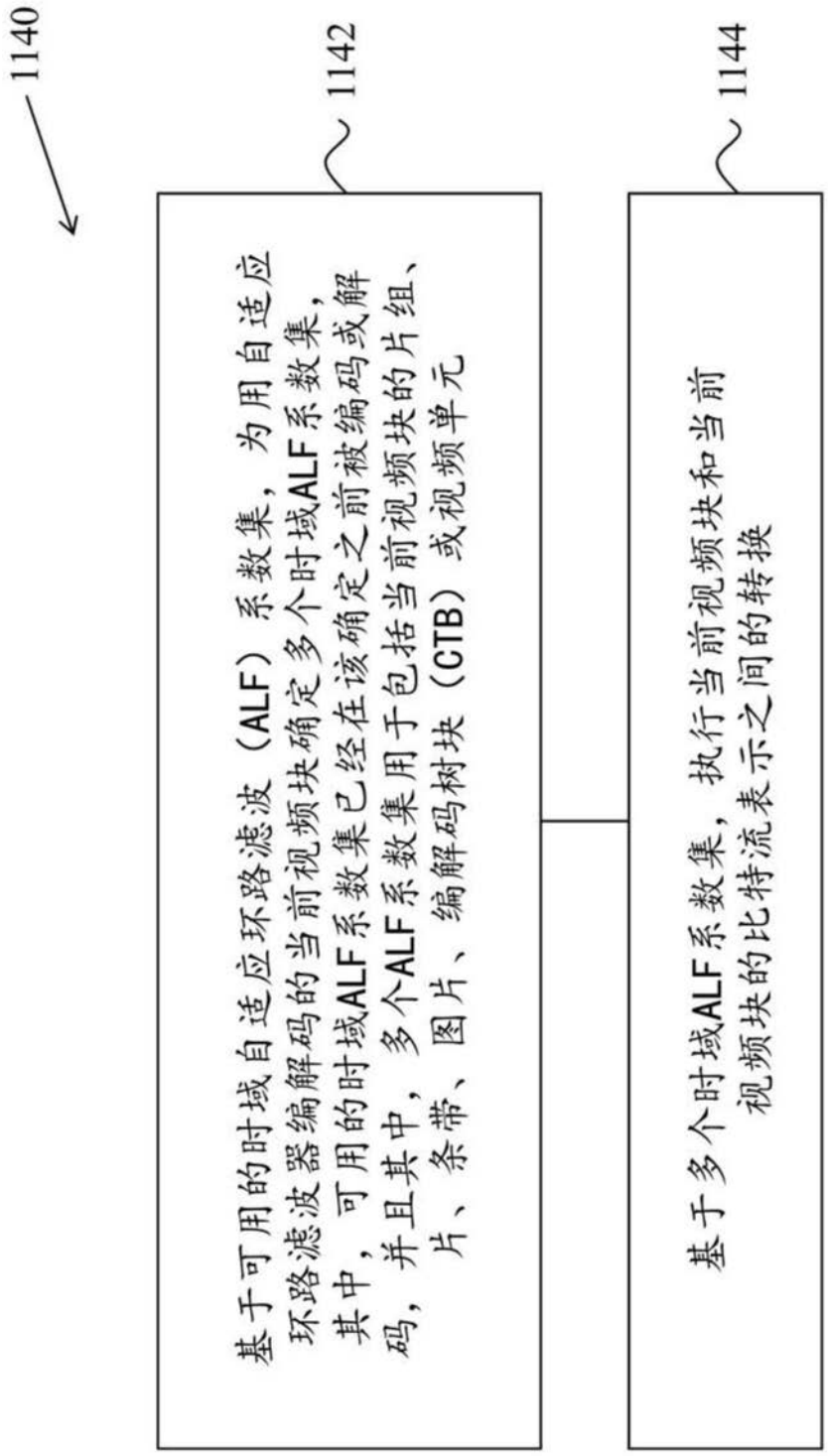

1140
1142
基于可用的时域自适应环路滤波（ALF）系数集，为用自适应环路滤波器编解码的当前视频块确定多个时域ALF系数集，其中，可用的时域ALF系数集已经在该确定之前被编码或解码，并且其中，多个ALF系数集用于包括当前视频块的片组、片、条带、图片、编解码树块（CTB）或视频单元
1144
基于多个时域ALF系数集，执行当前视频块和当前视频块的比特流表示之间的转换

图11D

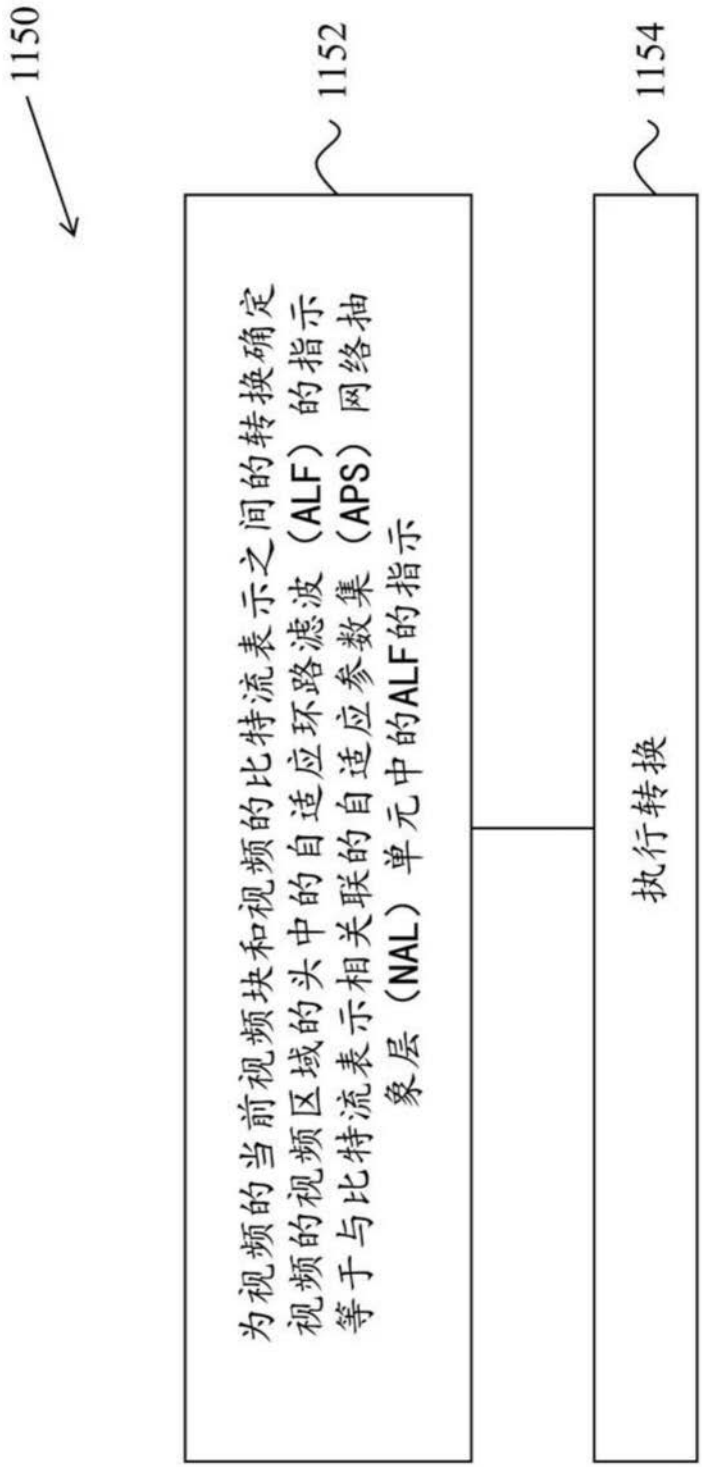

1150
1152
为视频的当前视频块和视频的比特流表示之间的转换确定视频的视频区域的头中的自适应环路滤波（ALF）的指示等于与比特流表示相关联的自适应参数集（APS）网络抽象层（NAL）单元中的ALF的指示
1154
执行转换

图11E

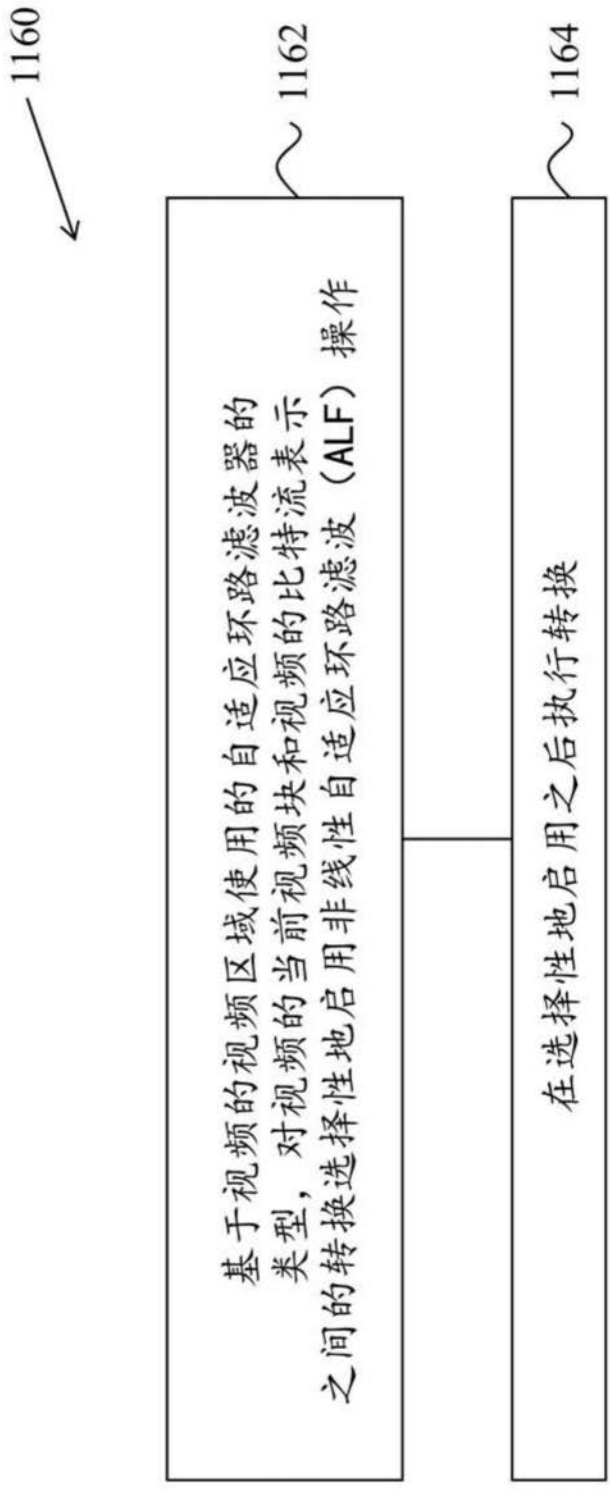

1160
1162
基于视频的视频区域使用的自适应环路滤波器的类型，对视频的当前视频块和视频的比特流表示之间的转换选择性地启用非线性自适应环路滤波（ALF）操作
1164
在选择性地启用之后执行转换

图11F

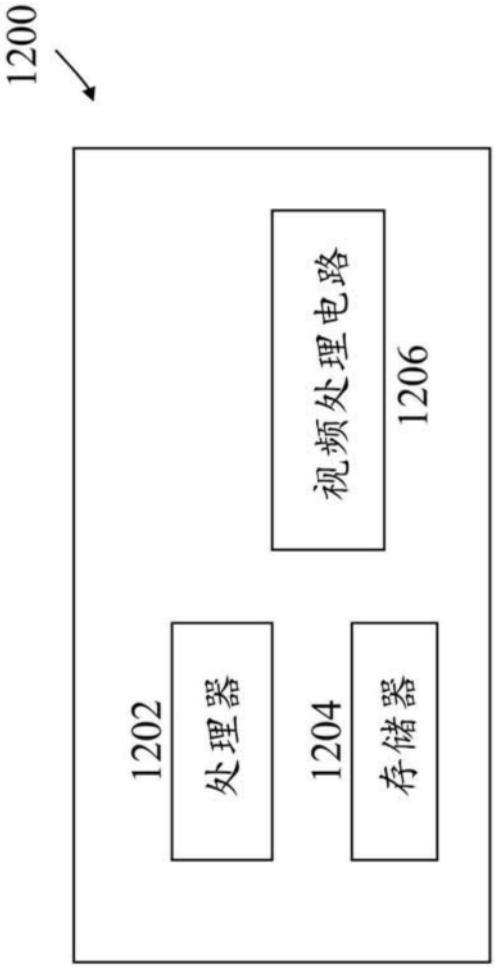


图12

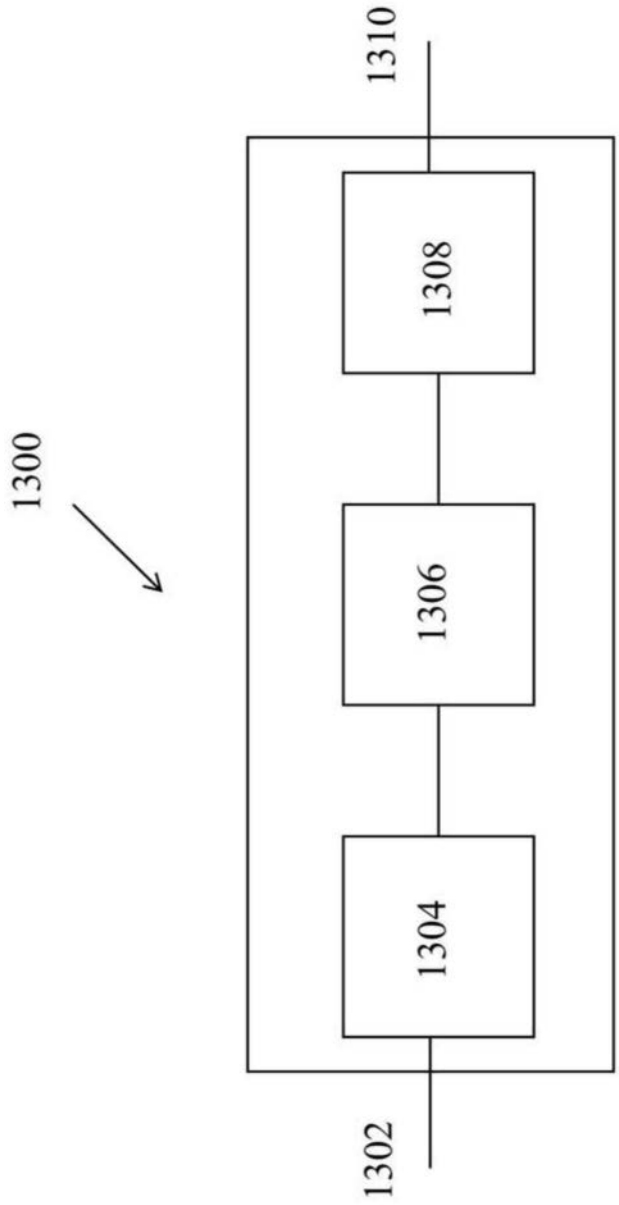


图13