bit error spreads

p197, QP prediction

编码效率提升 50

dct 能量集中性

bib:iso, 新一代...,High Efficiency Video Coding (HEVC): Algorithms and Architectures,AI: https://zhuanlan.zhihu.com/p/40034222,chapter 6,dct: https://blog.csdn.net/u012596983/article/details/39

摘要

The new HEVC standard enables a major advance in compression relative to its predecessors, and its development was a large collaborative effort that distilled the collective knowledge of the whole industry and academic community into a single coherent and extensible design. This book collects the knowledge of some of the key people who have been directly involved in developing or deploying the standard to help the community understand the standard itself and its implications. A detailed presentation is provided for each of the standard' s fundamental building blocks and how they fit together to make HEVC the powerful package that it is. The compression performance of the standard is analyzed, and architectures for its implementation are described.

1 绪论

视觉是人类感知世界最重要的方式。中文说 "眼见为实", 英文强调 "Seeing is believing", 人们总是乐于接受所能看到的信息, 因此人类的科技一直不懈地致力于为人们提供更多、更好的用于 "看" 的信息。黑白电视的发明曾经给人们巨大的震撼, 很快, 人们对于色彩的天然渴望, 又促使彩色电视出现。进入数字时代之后, 数字视频更是紧随着 IT 技术的浪潮, 获得了非常迅速的发展。追求更高的清晰度, 是数字视频技术领域从未停止的步伐。如今, 各式各样的视频应用已经渗透到人类社会的各个领域, 可以说, 视频应用是现代人类社会运转的重要组成部分。

1.1 视频编码概述

视频包含的数据量十分庞大。以常见的标清电影视频格式 (720p) 来说, 未经压缩每秒需要处理的数据量为 $1280 \times 720(pixels/frame) \times 3(Byte/pixel) \times 30(frames/second) = 79.1MB/s$!, 远远超过了一般用户数据处理能力的极限, 更不用说 4K 视频以及流媒体应用了。因此, 视频编码技术是视频应用的一个十分重要的部分, 也称视频压缩, 目的是尽可能去除视频数据中的冗余, 减少压缩或编码后的数据量。所幸的是视频中包含大量的冗余信息, 通过去除掉这些冗余视频可以获得极高的压缩比。

1.2 数据压缩原理

由香农的信息论, 一个离散信源 X 可以最小平均码长 $l = H(X)$ 信源熵来表示而不损失信息, 这种不失真的压缩方法称为无损压缩, 又称熵编码, 常见原理有 Lempel-Ziv 算法,Huffman 编码, 算

术编码,Golomb 码,RLE(游程编码) 等。但是熵编码一般难以达到 5:1 以上的压缩率, 故视频压缩都采用有损压缩方式。

有损压缩的原理可以用率失真原理来解释。对一离散信源, 给定一定失真度 D, 则可以以最小平均码长 R(D) 其率失真函数来表示。具体实现上就是对原信源引入难以察觉的失真来换取更小的信源熵, 从而得到更高的压缩率。例如人眼对图像的细节, 即图像中的高频成分不十分敏感, 则在压缩时丢掉部分高频信息可能并不会被人眼察觉。**如图**。此外, 即使压缩产生的失真能被人感知到, 但是如果这不会影响对视频内容的理解的话人们也通常愿意接受质量稍差的音、视频或者图像。在音视频和图像压缩算法中, 大量利用了人类的感知特性, 尽可能使压缩产生的失真发生在人不容易察觉到的地方。

1.3 视频编码标准

各种各样的视频应用从一开始就催生了多种视频编码方法。为了使编码后的吗刘能够在大范围内互通和规范解码, 从 20 世纪 80 年代起, 国际组织开始对视频编码建立国际标准。视频编码的国际标准通常代表这同时代最先进的视频编码技术。本文介绍的就是目前国际上最先进的视频编码标准 H.265/HEVC。

1.3.1 H.26x 系列

H.261 混合编码 (Hybrid Coding) 始祖

1.3.2 MPEG 系列

1.4 H.265/HEVC 简介

1.4.1 标准化历程

近年来, 随着高清、超高请视频 (分辨率达 $4K \times 8K$ $8K \times 4K$) 应用逐步走进人们的视野, 视频压缩技术收到了巨大的挑战。此外, 各式各样的视频应用也随着哇个尼罗河存储技术的发展不断涌现。如今, 数字视频广播、移动无线视频、远程监测、医学成像和便携摄影等, 都已走进人们的生活。视频应用的多样化和高清化趋势对视频压缩性能提出了更高的要求。为此,2010 年 4 月 VCEG 和 MPEG 再次组建视频编码联合组 (Joint Collaborative Team on Video Coding, JCT-VC), 联手制定新一代视频编码标准——H.265/HEVC。

2013 年 6 月 7 日,ITU-T 网站上正式发布了 H.265/HEVC 标准, 该标准可以免费下载。2013 年 11 月 25 日,ISO/IEC 正式发布了 H.265/HEVC 标准。

标准发布之后, 相关标准的进一步工作仍然在继续。JCT-VC 现有的工作主要集中在就 H.265/HEVC 的扩展内容进行完善, 如更高的比特深度、4:2:2、4:4:4 色度采样视频、可伸缩 HEVC 编码 (Scalable HEVC,SHVC) 和多角立体编码等。

1.4.2 编码框架

从根本上来说,H.265/HEVC 视频编码标准的编码狂及并没有革命性的改变。类似与以往的国际标准,H.265/HEVC 仍旧采用混合编码框架, 包括变换、量化、熵编码、帧内预测、帧间预测以及环路滤波等模块。但是,H.265/HEVC 几乎在每个模块都引入了新的编码技术。

1. 帧内预测

   图像具有很强的空间相关性。通过已编码像素块来预测当前像素块有很好的去相关作用, 从而达到去除冗余的效果。

2. 帧间预测

   视频具有很强的时间相关性。通过已编码帧来预测当前帧来获取块的运动信息可以达到去相干的作用, 从而去除冗余。

3. 变换编码与量化

   变换编码通过对块进行频域变换把能量集中在低频区域, 量化把无限精度的数值以有限精度表达, 从而达到压缩效果。在 HEVC 中为了减少计算复杂度将变换编码与量化结合起来。该模块是编码中唯一有损的部分, 同时也是速率控制的关键。

4. 环路滤波

   基于块编码后的图像会出现方块效应, 采用去方块滤波后可削弱其影响。对一信号采取有限频率逼近会出现振铃现象, 可通过 SAO(样点自适应补偿) 滤波来削弱。环路滤波目标即为削弱编码过程中预测、变换和量化等环节中引入的失真。

5. 熵编码

   经各路编码后的变换系数、运动矢量、参数集等仍有压缩空间。实际存储或传输的数据为经过熵编码后的二进制流。

除此之外, 为了适应不同的网络环境与视频应用,H.265/HEVC 也采用了视频编码层 (Video Coding Layer, VCL) 和网络适配层 (Network Abstract Layer, NAL) 的双层架构; 随着处理器多核架构的发展, 多核并行处理成为提高编解码能力的有效手段,H.265/HEVC 在设计时充分考虑到了并行处理的必要性, 在数据单元划分与语法结构上均为其做了准备; 为了在满足信道带宽和传输时延限制的情况下有效传输视频数据, 速率控制是实际编码器的很重要的组成成分。H.265/HEVC 的官方编码器 HM 也提出了一些速率控制算法等。这些在本文最后会简要介绍。

2 编码结构

在基于块的视频混合编码架构中, 每一张图像都被分解为不同的像素块, 不同的像素块组合构成了不同的片 (Slice) 作为独立的解码单元。HEVC 标准也继承了这一点。但在图像与块的划分上 HEVC 创新性地提出了基于四叉树的递归划分方式, 能更灵活、高效地表示视频场景中的不同纹理细节、运动变化的视频内容或者视频对象。实验证实相较于上一代编码标准 H.264 | MPEG-4 AVC,HEVC 一半以上性能提升都归功于这种灵活的划分方式。

2.1 HEVC 编码结构简介

HEVC 标准以在以往获得很大成功的混合编码架构为基础。在该架构中, 首先一帧的图像被分解成不同的块, 而后每一块采用预测编码来消除其空间或者时间上的冗余。其中帧内预测以同一帧中已解码的像素块作为参考、帧间预测以已解码的其他帧作为参考。帧内预测利用了同一图像中临近块间的空间冗余进行压缩, 帧间预测利用了大量的块运动信息来消除视频的时间冗余。无论是哪种方式, 最终预测的误差, 即预测像素与原始像素的残差经变换编码与标量量化被去相关, 最终经过熵编码传输。图2.1以框图形式表示了整个编码流程。

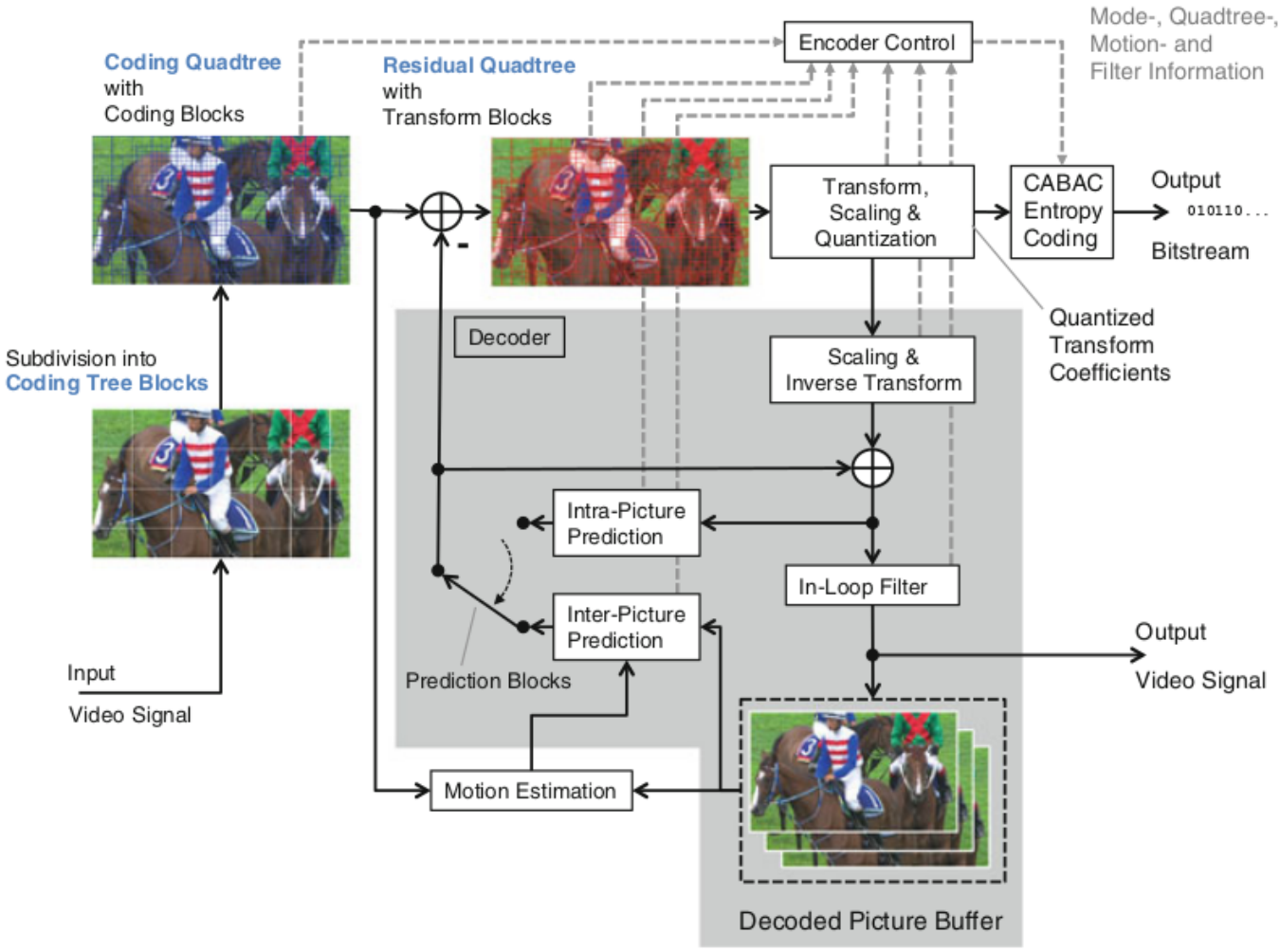

图 2.1: Block diagram of an HEVC encoder with built-in decoder (gray shaded)

该图同时展示了 HEVC 特有的编码结构, 这些将在后面的部分分别叙述。首先我们看一下这个分块结构:HEVC 中每一帧都被分解为同样大小不相交的方块, 又称树形编码块 (Coding Tree Block,CTB), 每一个 CTB 作为四叉树编码结构的根节点, 其大小由编码器指定。CTB 还可被进一步细分为编码块 (Coding Block,CB),CB 是编码器决定帧内预测与帧间预测进行编码的基本单元。每一帧划分为 CTB 与 CTB 划分为 CB 的方式将在章节2.2.1与2.2.2里描述。为了进行预测而划分 CB 方式在很大程度上与与 CTB 无关, 具体实现将在章节2.2.3里介绍。变换编码依赖于 CB 的预测残差, 但可在 CB 的基础上被进一步细分为变换编码块 (Transform Block,TB), 具体方式将在章节2.2.4里介绍。最后, 章节??提供了一些 HEVC 的实验数据, 并将其与以前的编码标准做对比。

Slice 层面的分割为并行处理提供了方便, 将在并行处理部分讲解。

2.2 块分割

自 H.261 以来, 所有 ITU-T 与 ISO/IEC 视频编码标准都使用混合编码架构, 而不同标准间的区别则主要体现在对同一块样本块不同标准可供选择的编码模式集的不同。一方面, 所选编码模式决定了该样本块是使用帧内预测还是帧间预测; 另一方面, 解码器需要知道一样本块是怎么被分解为帧内/帧间预测块的。通常来说被预测的块还需要传递其参数, 对于帧内预测需要传递预测模式, 帧间预测则需要传递运动矢量信息。

为了给编码器与解码器的开发者足够的自由, 同时保证不同厂家生产的设备的互操作性, 视频编码标准仅规定了编码码流的语法语义与解码过程, 具体编码过程则未作要求。因此, 编码效率在很大程度上依赖于决定编码语义元素的算法, 其中包括编码模式的选择、预测参数、量化参数与已量化变换矩阵索引等。一个简单且行之有效的方法是拉格朗日率失真优化 (Rate–distortion optimization,RDO)。在这种方式中, 参数 p^* 的选择由在可选参数集 $\mathscr{A}$ 中最小化损失函数 D 与编码比特数 R 的加权和来决定,

$$p^* = \arg\min_{\forall p \in \mathscr{A}} D(p) + \lambda \cdot R(p) \tag{2.1}$$

其中拉格朗日参数 λ 为一常数, 用来权衡失真 D 与比特数 R, 因此视频质量与码率被兼顾。

一个混合编码可达到的编码效率取决于很多设计准则, 像是内插滤波器的设计, 熵编码的效率, 以及环路滤波方法, 然而两代编码标准的性能提升最关键在于可供选择的块编码方式的增加, 这体现在帧间预测时更高精度的运动矢量、更灵活的帧编码顺序、更多的参考帧的获取, 帧内预测时更多的预测模式, 以及更多的运动矢量预测器 (?), 更多的变换块大小以及运动补偿块大小等。

然而也不是分块与预测方式越多越好。考虑一给定的样本块, 我们可以把它分为更小的子块选取更佳的预测参数来获得更小的误差, 代价则是更高的的码率; 同样若不再细分, 则随码率的下降误差也会增加, 具体哪种更好取决于待编码块。当可供选择的分块方式增加时, 大体上我们需要更多的比特数来表示所选的模式, 同时编解码复杂对也会相应增加。因此, 标准的设计需要综合考虑这些方面。

由于个人电脑计算能力的提升, 新的视频编码标准大体上会支持更多的编码选项。在 HEVC 标准设计时, 考虑到高清 (High Definition,HD) 与超高请 (Ultra High Definition,UHD) 视频的需求,HEVC 支持了更大的编码块用以进行预测与变换; 同时为了不遗漏细节, 小的编码块也是十分重要的。这两种截然相反的目标被 HEVC 以一种开创性的, 同时简单而有效的递归四叉树划分法解决了。除此之外, 这种四叉树划分方式也支持一种快速最优化算法 [?] 来计算拉格朗日率失真代价。

2.2.1 CTU 与 CTB

以往 ITU-T 与 ISO/IEC 提出的编码标准中, 每一帧都被分解为宏块, 每个宏块包含大小为 16×16 的亮度采样块。色度采样块由视频采样率决定, 对于 4:2:0 采样率的视频其包含两块 8×8 的色度采样块。宏块为编解码处理的基本单元, 对每一个宏块均需要确定其预测方式。

尽管目前为止 H.262 | MPEG-2 与 H.264 | MPEG-4 AVC 标准也被用于保存与传输高清 (HD) 视频内容, 其最初设计目标是分辨率在 QCIF(176×144) 到标清 ($720 \times 480, 720 \times 576$) 的视频。由

于分辨率可高达 3840×2016 与 7680×4320 的 HD 与 UHD 视频的兴起,HEVC 最初设计时就考虑到了高分辨率视频。在如此高的分辨率下视频中会有大块平坦区域与大块运动矢量一致的区块, 因此采用更大的分块大小能明显减少视频容量。同时 HEVC 标准也被设计用于为所有现有视频内容提供优于上一代视频编码标准 H.264 | MPEG-4 AVC 的压缩效率, 因此也必须设计出小尺寸的编码块。基于此,HEVC 提供了灵活的基于四叉树的分块方式。

在 HEVC 中, 为使图像总体整体 CTB 数目一致, 每一分块均包含一个亮度 CTB 与两个色度 CTB。每一亮度采样块与其对应的色度采样块, 加上其语法元素构成了一个树形编码单元 (Coding Tree Unit,CTU), 作为 HEVC 的基本处理单元, 概念上与宏块相似。亮度 CTB 大小为 $2^N \times 2^N$, 对于 4:2:0 采样率视频来说每一色度 CTB 大小为 $2^{N-1} \times 2^{N-1}$。N 可取值 4,5 或 6, 分别代表的 CTU 大小为 $16 \times 16, 32 \times 32$与$64 \times 64$, 并于一开始在序列参数集 (Sequence Parameter Set,SPS) 中传输。图2.2表示将一分辨率为 1280×720 的视频分解为 16×16 宏块与 64×64 的 CTU 的结果。

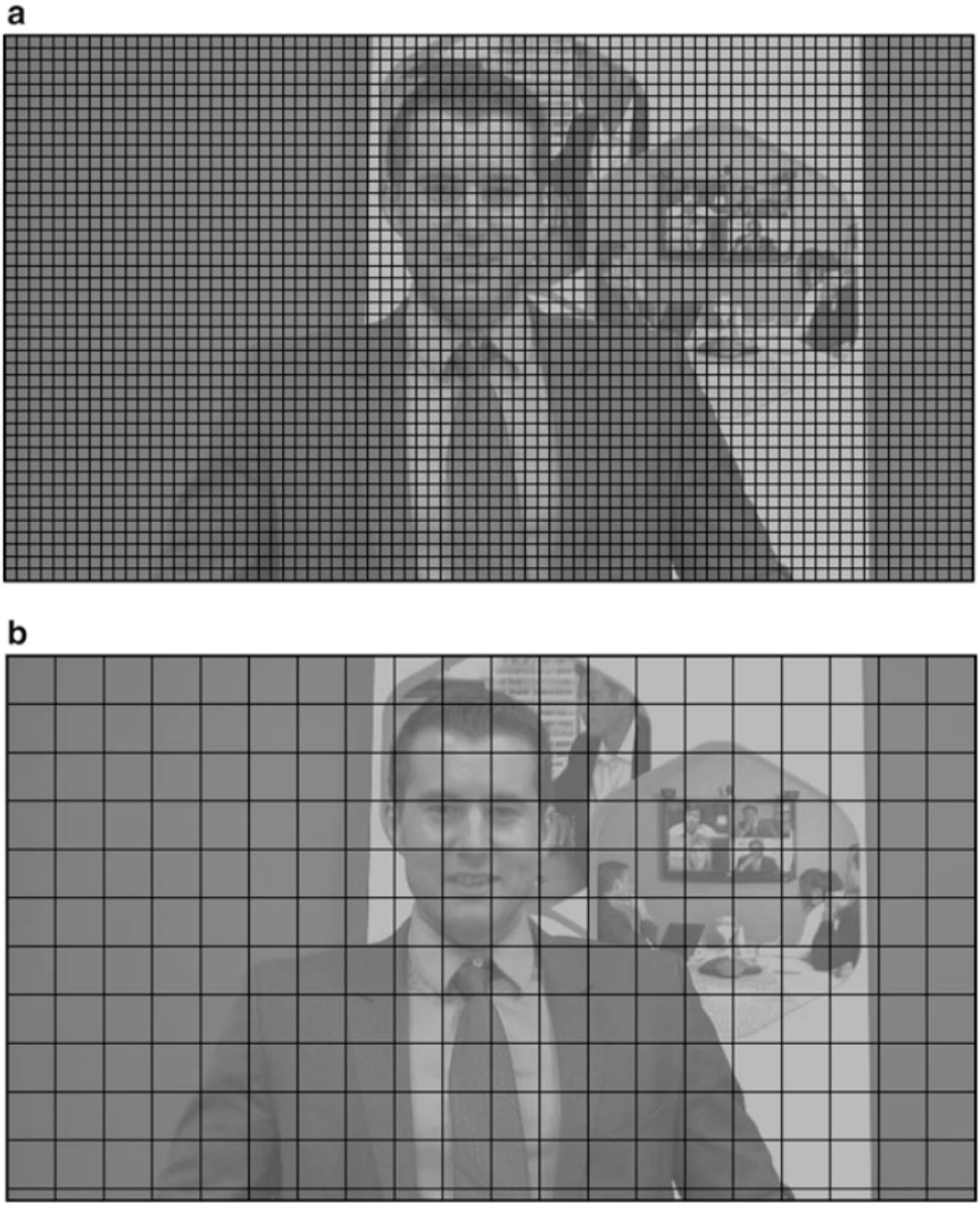

图 2.2: 图像划分方式:(a)16 × 16 宏块;(b)64 × 64 CTU。可见对于该图像来说 16 × 16 的分块方式明显不如 64 × 64 的方式编码来得有效率

编码器可自主选择大的 CTU 用以适应更高分辨率视频或者更高的编码效率, 或者选择更小的 CTU 用于适应低分辨率视频或者更好的保真度。

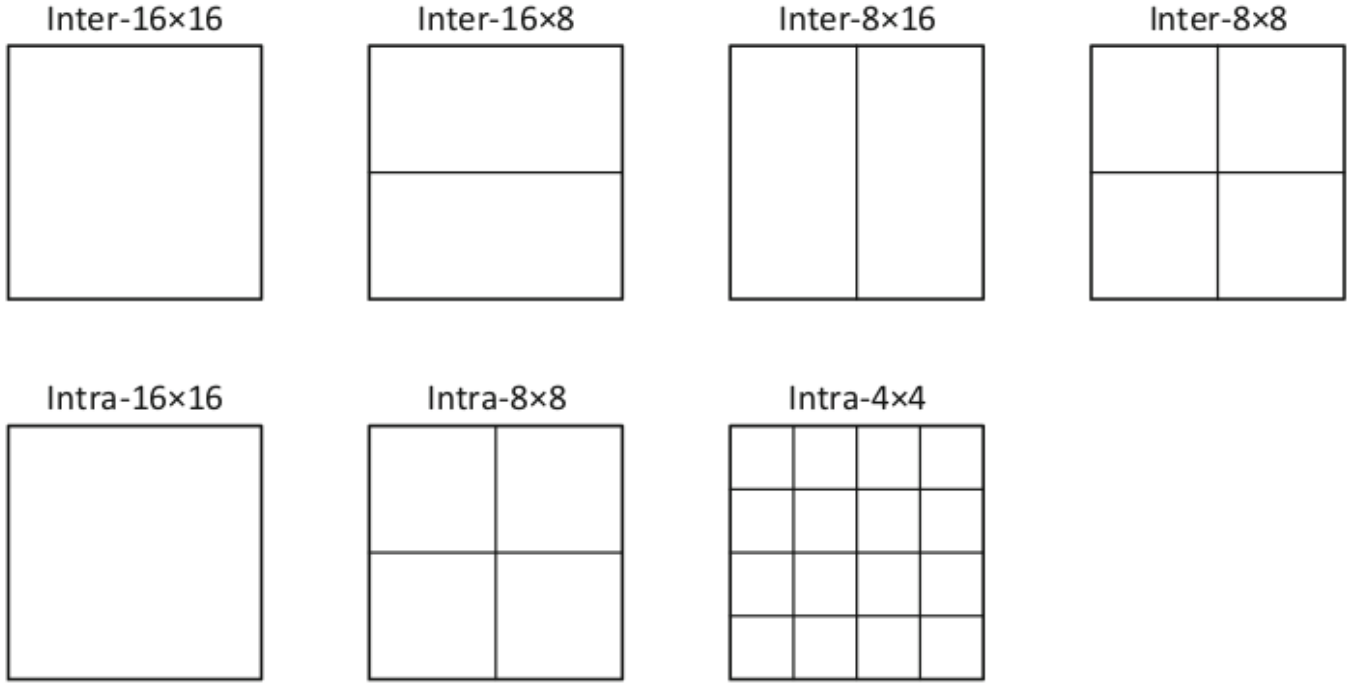

图 2.3: Macroblock partitioning modes supported in the High profile of H.264 | MPEG-4 AVC for inter-picture coding (top line) and intra-picture coding (bottom line). If the $Inter-8\times8$ is chosen, the 8×8 sub-macroblocks can be further partitioned into 8×4, 4×8, or 4×4 blocks

2.2.2 CTU 与 CU

在以往的 ITU-T 与 ISO/IEC 视频编码标准中, 宏块为预测的基本单元, 每一宏块均需确定其预测方式为帧内预测还是帧间预测, 在不同的预测方式中, 宏块需再分解为更小的子块来分别进行预测。

在上一代视频编码标准 H.264 | MPEG-4 AVC 中, 帧内预测有三种分块方式, 分别为 $Intra-4\times4$,$Intra-8\times8$ 与 $Intra-16\times16$; 帧间预测有四种分块方式, 分别为 $Inter-16\times16$,$Inter-16\times8$,$Inter-8\times16$ 与 $Inter-8\times8$, 图2.3表示了这些方式。每种预测方式中一块的预测参数由其相邻块的已解码信息判断。

在 HEVC 中,CTU 的大小可以达到 64×64, 因此宏块的方式明显不可取。一方面基于 CTU 的预测方式选择过于粗糙, 难以很好地重构原图像; 另一方面如果要支持如 H.264 | MPEG-4 AVC 宏块形式的块分割方法的话语法会变的很复杂, 无法改变的块大小也很不适合于视频压缩中。

为了解决这些问题,HEVC 将每一个 CTU 以四叉树的方式进一步分解为编码单元 (Coding Unit,CU), 如图2.4所示。与 CTU 类似, 每个 CU 包含一个亮度采样块与两个色度采样块与其语法元素, 作为决定预测方式的基本单元, 而在预测编码与变换编码中一个 CU 还可进一步被分解为更小的预测单元 (Prediction Unit,PU) 与变换单元 (Transform Unit,TU)。

在 CTU 层面上, 一个 CTU 是否被分解由标志位 `split_cu_flag` 决定, 而其分解后的每一块是否被分解由另一标志位 `split_cu_flag` 决定。当无待分解块时, 分解结束。CU 的最小大小在序列参数集 (Sequence Parameter Set,SPS) 中指定, 其大小在 8×8 与 CTU 大小之间。一般编码配置充分利用了 CU 大小的可变性, 其尺寸在 8×8 与 64×64 之间。

CU 编码采用深度优先顺序, 或者称为 Z 型扫描顺序。使用这种编码方法可以保证除了在左上

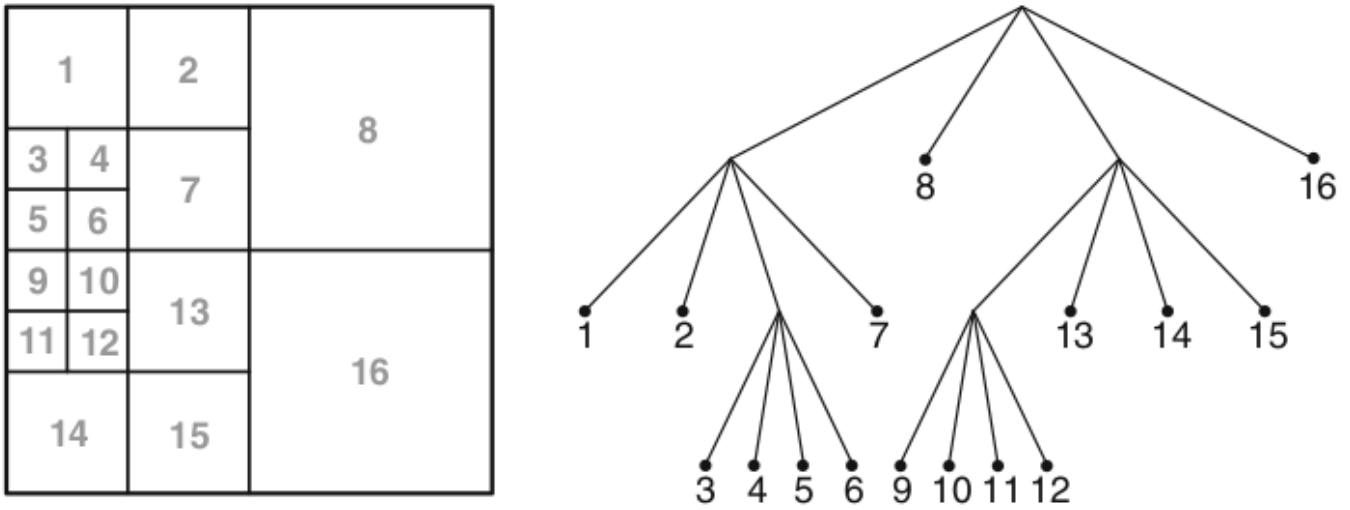

图 2.4: Example for the partitioning of a 64×64 coding tree unit (CTU) into coding units (CUs) of 8×8 to 32×32 luma samples. The partitioning can be described by a quadtree, also referred to as coding tree, which is shown on the right. The numbers indicate the coding order of the CUs

边缘处的 CU, 其余 CU 编码时其左侧或上侧的 CU 已经编码, 则其样值与预测参数可以拿来预测当前 CU 的编码参数。

整体上来说,CU 与以前编码标准中的宏块很相似, 但是 CU 大小可变, 这就给予了 HEVC 更高的灵活性。

2.2.3 PB 与 PU

对每一 CU, 需要决定其预测方式; 而一旦其预测方式确定了, 其编码参数也要相应地确定下来。

对于帧内预测来说, 共有 35 种空域预测模式。如果 CU 尺寸与 SPS 中规定的最小 CU 尺寸一致, 亮度 CB 可进一步被分解为四个同样大小的子块, 需要分别传输其预测模式。而色度 CB 与其对应 CU 尺寸无关, 两色度 CB 采用同种预测方式。色度 CB 可选五种预测方式, 其中一种与亮度 CB 一致或在亮度 CB 传输四个预测模式时为其第一个。实际的帧内预测并不一定以已确定预测模式的编码块为单位进行。实际上每一编码块有可能被分解为更多的变换块 (Transform Block,TB), 而残差的计算基于已重建的像素块进行, 如图2.5所示, 当变换块增大时, 由于预测点距离增大误差一般会更大, 但对应的比特率也会减少, 因此这种可选特性提供了一种权衡质量与比特率的手段。

如果一 CU 采用帧间预测方式, 其亮度与色度 CB 可被进一步分解为预测块 (Prediction Block,PB), 每一预测块共享同一运动参数。运动参数包括运动候选 (1 或 2)、参考图像索引以及每种运动候选的运动矢量。某一 CU 的亮度 CB 与其两色度 CB 采用同种划分方式。亮度 PB, 其对应的两色度 PB, 与其语法元素构成了预测单元 (Prediction Unit,PU), 实际传输 PU 的预测参数。

HEVC 支持 8 种 PU 划分方法, 如图2.7所示, 一个 CU 可以整体作为一个 PU 来进行预测, 也可划分为子块来进行, 其中 $(M/2) \times (M/2)$ 方式仅当最小 CU 尺寸大于 8×8 时可用, 非对称形式仅对 8×8 尺寸以上的 CU 可用, 因此帧间预测的最小尺寸为 8×4 与 4×8。

如此多的区块划分方式为提高编码效率提供了可能, 但是编码器需要遍历所有方式, 运算复杂度很高。同时为表示这么多种预测方式所对应的语法元素最终甚至可能降低编码效率。为此, 可在

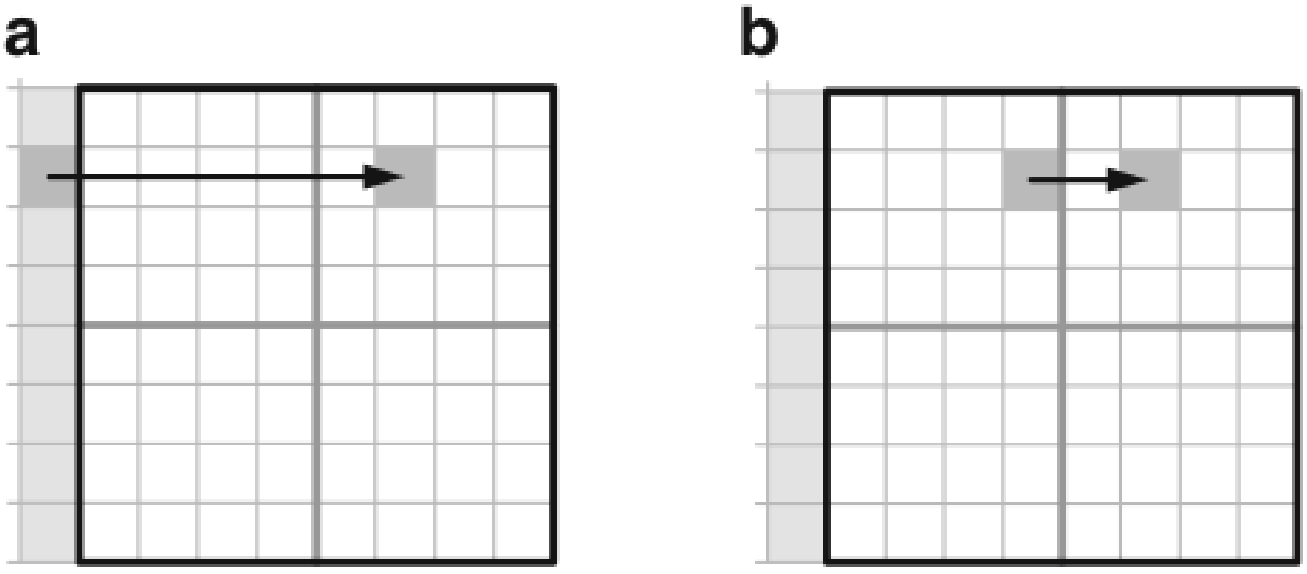

图 2.5: Illustration of the horizontal intra prediction of a selected sample inside an 8×8 coding block with 4×4 transform blocks, if the intra prediction is applied on the basis of coding blocks (a) or transform blocks (b)

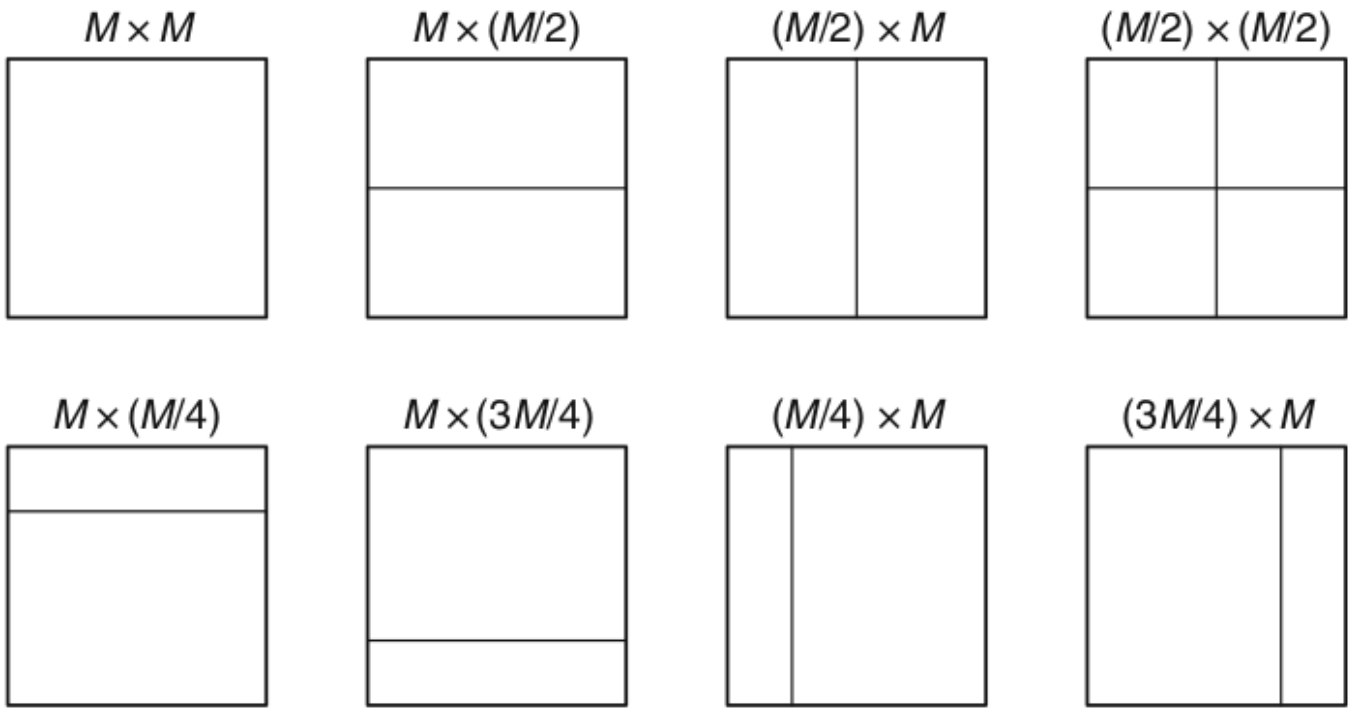

图 2.6: Supported partitioning modes for splitting a coding unit (CU) into one, two, or four prediction units (PU). The $(M/2) \times (M/2)$ mode and the modes shown in the bottom row are not supported for all CU sizes

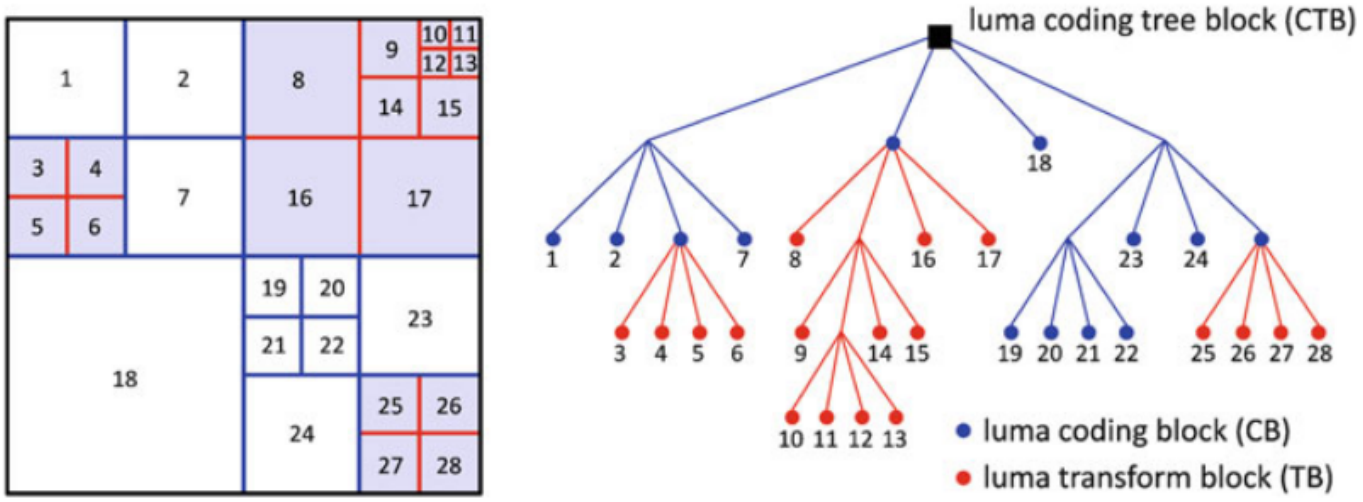

图 2.7: Example for the partitioning of a 64 × 64 luma coding tree block (black) into coding blocks (blue) and transform blocks (red). In the illustration on the right, the blue lines show the corresponding coding tree with the coding tree block (black square) at its root and the coding blocks (blue circles) at its leaf nodes; the red lines show the non-degenerated residual quadtrees with the transform blocks (red circles) as leaf nodes. Note that the transform blocks chosen identical to the corresponding coding blocks are not explicitly marked in this figure. The numbers indicate the coding order of the transform blocks

SPS 中指定可选的分块模式集来权衡质量与码率。

2.2.4 残差四叉树 (RQT),TB 与 TU

如上一章节所述, 在预测残差的变换编码中, 一个 CB 可被分解为多个变换块 (Transform block,TB), 这种分解基于一种叫残差四叉树 (Residual Quadtree,RQT) 的结构递归进行。在每个 RQT 中,CB 为该树的根节点, 每个 TB 为位于该树的叶子节点上, 图2.7为将一 64 × 64 亮度 CTB 分解为亮度 CB 与亮度 TB 的过程。色度 CB 基于同样的结构被分解为色度 TB, 但有一个例外, 我们将在后面提到。

允许不同大小的变换块给我们以不同尺度分析空频特性的可能: 更大的变换块有更高的频率分辨率, 然而更小的变换块有更高的空间分辨率, 这两者的权衡可在编码器层面控制。

每个 RQT 有三个参数: 最大深度 d_{max}、最小变换块尺寸 n_{min} 与最大变换块尺寸 n_{max}, 并与 SPS 中传输。后两者可取值 2 到 5, 代表的变换块尺寸为 4 × 4 到 32 × 32。最大深度 d_{max} 限制了该 RQT 的深度, 如 $d_{max} = 1$ 时, 一个亮度 CB 可作为一个亮度 TB 或被分解为 4 个亮度 TB, 但不可再分。需要注意的是有时变换块尺寸被隐含在参数中, 如即使 $d_{max} = 0, n_{max} = 5$, 对于一 64×64 的亮度 CB, 仍必须将其分为四个 32 × 32 的亮度 TB。

若去相干变换作用于多个预测块, 变换残差经常包含预测边界, 这会导致高频分量能量增加从而降低编码效率。由于这个原因, 在 $d_{max} = 0$ 时 HEVC 也包含了一个隐式分割的条件。若 $d_{max} = 0$, 且一个 CU 采用帧间预测方式, 同时一个 CU 被分解为多个 PU, 此时亮度 CB 总是被分解为四个亮度 TB。当 $d_{max} > 0$ 且 CU 采用帧间预测方式时,RQT 分割与 PU 分割无关, 因此一个 TB 可能包含多个 PB。虽然这样可能会降低该 CB 的编码效率, 但是同时别的 CB 的编码效率会增加。研

究表明 [?] 当采用这种方式时其 Bjøntegaard Delta bit rate (BD rate) 会增加 $0.4-0.7\%$。帧内预测则不同, 一个 TB 不可跨越多个预测区块, 则当一个亮度 CB 尺寸等于最小 CB 尺寸, 且其被分解为四个预测块传输不同的预测参数时, 该 CB 必被分解为四个 TB。同时分解得到的四个 TB 可能再次被分解。

对每个 CU, 至多一个 RQT 语法元素被同时传输, 该 RQT 同时决定了所有颜色分量的划分方式。但有一个例外, 对于 4:2:0 采样率的视频来说, 若亮度 TB 尺寸为 4×4 时, 色度 TB 不被再分。一个尺寸大于 4×4 的亮度 TB, 或者四个尺寸为 4×4 的亮度 TB, 与其伴随的两色度 TB 加上其他语法元素组成了一个变换单元 (Transform Unit,TU)。

由于 RQT 位于 CU 中, 必须对每个 CU 传输其语法元素。例如当 CU 传递完其预测模式,PU 分块以及 PU 相关语法元素后, 若其变换系数等级不为 0, 语法元素 `split_transform_flag` 会被传输来表示其是否为叶子节点。当隐式推测发生时, 则由译码器推测其具体数值。

除了 RQT 的具体结构外, 也需要知道对于一 TB 或者整个 CU 来说变换系数等级是否为 0。对采用帧间预测的 CU 来说, 标志位 `rqt_root_cbf` 用来表示是否至少有一个 TB 的等级不为 0。当其为 1 时如上所述传输, 否则不再传输残差矩阵, 其数值被视作 0。该语法元素对于低码率编码以及可精确预测的区域有十分重要的意义。对 skip 模式的 CU 来说, 标志位 `cu_skip_flag` 被置为 1, 此时没有残差需要传输, 对应的也没有 RQT 语法元素需要传输。然而对于帧内预测的 CU 来说,`rqt_root_cbf` 总被视为 1, 因此总可以认为至少一个 TB 的等级不为 0。

更进一步说, 当 `rqt_root_cbf=1` 时, 对每个亮度 TB 与其对应的两个色度 TB 也需要传输另一个 cbf 为。亮度由 `cbf_luma` 表示, 色度标志位 `cbf_cb` 与 `cbf_cr` 与 `split_transform_flag` 被交错编码。这种编码方式在有一个或所有色度 TB 残差均为 0, 而亮度 TB 残差不为 0 时编码效率进一步增加。Details [?, ?]。

随 RQT 与其他编码树分解深度的增加, 可供选择的分解方式也呈双指数速率 $2^{4^{d-1}}$ 增加。然而, 文章 [?] 指出, 通过采用通用 BFOS 算法 [?], 率失真下的最优分块方法并不一定需要穷举才能得到。实际上不采用 early termination strategy 的话, 运算复杂度与 $(4^d-1)/3$ 成正比, 为了进一步降低运算复杂度, 还可使用 heuristic early-pruning techniques [?, ?]。

文章 [?] 针对 RQT 分块结构提出了一种算法, 思想是当所有未量化变换系数低于合理选取的量化器步长阈值时, 进一步细分可被终止。采用这种策略, 编码器运行时间可减少 $5-15\%$, 而编码器效率仅有很少损失。当 RQT 更深时, 耗时会减少更多。Details [?]。

2.2.5 HEVC 官方编/解码器 HM 工作流程

图与分别为 HM 中的 HEVC 视频编码器与解码器的函数调用关系, 以供参考。

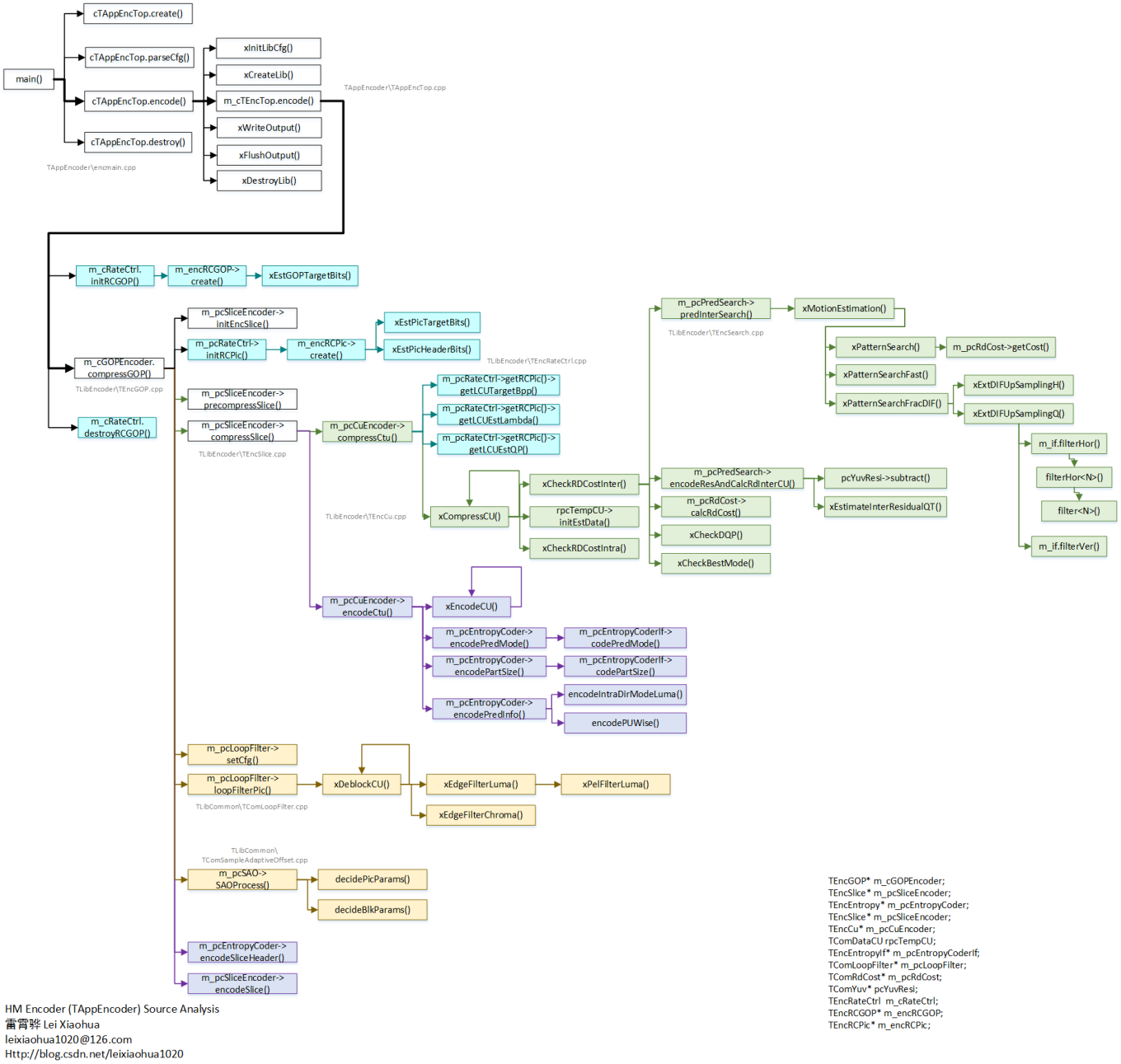

图 2.8: 编码器流程https://blog.csdn.net/leixiaohua1020/article/details/49912113

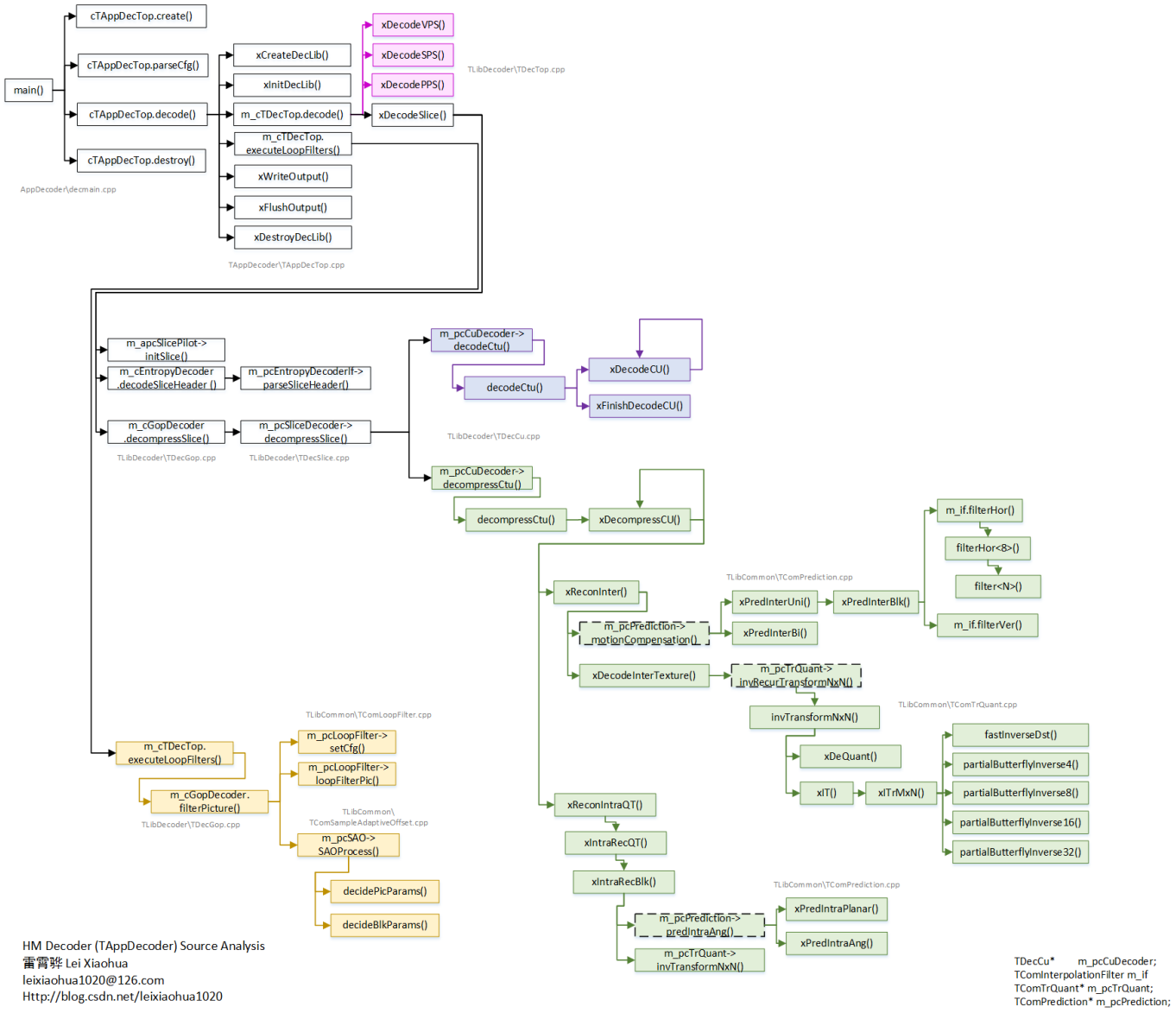

图 2.9: 译码器流程https://blog.csdn.net/leixiaohua1020/article/details/49912013

3 预测编码

4 变换编码与量化

5 环路滤波

6 熵编码

上下文参考自适应二进制算术编码 (Context-Based Adaptive Binary Arithmetic Coding,CABAC) 是 HEVC 使用的熵编码方法。尽管它最早在 H.264/AVC 标准中提出并拥有高于绝大多数熵编码的压缩率, 其数据依存性使得它难以并行处理, 因此仅在 Main Profile 以及更高档次下可以使用。对应地在 HEVC 熵编码标准化过程中, 编码效率以及吞吐量两方面被细致地研究了。本章节先介绍熵编码的一些概念, 再给出 CABAC 的实现方式以及其压缩效率,HEVC 中 CABAC 的设计准则见文献 [?], 本文并不讨论。

6.1 熵编码概述

由 Shannon 的信息论, 一个离散信源 X 的信源熵为 $H(X) = E\left[log\frac{1}{P(x_i)}\right]$, 这也代表了对该信源进行可正确解码的编码后表示每个码字平均需要的最短码长, 而原信源平均码长等于信源熵的充要条件是其编码后的序列服从等概率分布 [?]。

这告诉了我们两件事:

1. 我们必须以不小于原信源信源熵的空间来储存信息, 额外需消耗的空间叫做这种表达方式的冗余。冗余的存在是能对信源进行压缩的前提与基础。

2. 熵编码的实质是对离散信源进行适当的变换, 使变换后新的符号序列信源尽可能为等概率分布, 从而使新信源的每个码符号平均所含的信息量达到最大。这一点和密码学的表述十分相近, 只不过密码学是要用少量信息来让源信源看似多了很多信息。

实际通信中, 信源通常输出的是符号序列, 而符号间彼此有一定相关性, 其联合熵可用来表征信源输出一个序列所提供的平均信息量, 定义为 $H_N(X) = H(X_1X_2\cdots X_N)$, 则当信源足够长时每个符号平均信息量可表示为 $h(\mathcal{X}) = \lim_{N\to\infty}\frac{H_N(X)}{N}$, 称为该信源的熵率。进一步来说, 对于平稳 Markov 信源 (给定当前信源符号, 未来符号与过去无关) 若将整个随机过程视为一整体来编码, 可以证明 [?] 条件熵 $H(X_N|X_1X_2\cdots X_{N-1})$ 是 N 的非增函数, 则有不等式:

$$h(\mathcal{X}) = \lim_{N\to\infty}\frac{H_N(X)}{N} \le H(X_N|X_1X_2\cdots X_{N-1}) \le H(X_i) \tag{6.1}$$

因此自适应编码一般来说压缩率总是比非自适应算法要高。

Huffman 于 1952 年提出了一种针对已知信源构造最优变长码的方法, 被称作 Huffman 编码, 其基本思想是为出现频率高的码字分配较短的码长, 因此能够实现压缩, 且有平均码长满足 $H(X) \le \bar{l} < H(X) + 1$。

6.2 算术编码

算术编码也为一种熵编码方法, 但由于它把整个信源当成一个符号处理, 可以为单个码字分配小于 1 的码长, 因此压缩效率更高。

算术编码原理是: 根据信源概率将 $[0,1)$ 区间划分为互不重叠的子区间, 子区间宽度为各符号序列概率, 这样信源符号就和各子区间一一对应。每输入一个符号就将选择的区间进一步划分, 则最终该区间就对应于输入码字, 实际输出的是该区间下长度最短的码字, 具体编码过程见图6.1, 其中 I,K,W 概率分别为 0.5,0.25,0.25。解码过程就是根据所传输数据判断它所对应的编码区间, 从而得到传输符号。

算数编码平均码长满足 $H(X) \leq \bar{l} < H(x) + \frac{2}{N}$, 其中 N 为序列长度。

普通的算术编码在实现上有几个问题:

1. 实际计算机精度不可能无限长, 因此运算过程中可能导致溢出, 这点可以通过比例缩放解决。
2. 编解码过程中需要不断进行查表并进行浮点数运算, 当符号个数大的时候开销很大。
3. 传输过程中有一个 bit 出现问题则代表整个解码序列出差错, 且在接收到所有序列前无法解码。

其中针对2, 本章节作者写了一个算数编/解码程序`github`, 信源符号视为 1 字节 ASCII 码 256 个字符, 在使用了如 Hash 索引, 二分查找等快速算法的情况下平均编码速度 36MB/s, 解码速度 12MB/s(i7-6700 单核,g++ 环境), 效率不高。

相对来说,CABAC 以二进制信源作为编码元素, 并采用自适应方法编码, 总体上有以下优点:

1. 只编码二进制元素, 因此运算复杂度较低, 并且可以针对最大概率符号 (Most Probable Symbol,MPS) 进行概率建模。
2. 概率模型由当前上下文自适应选择, 因此可以建模的很好。
3. 通过量化过的区间与概率状态在区间划分时不使用除法, 提高了运行效率。

6.3 CABAC

CABAC 在编码的最后一步进行, 此时视频已经变成了一系列语法元素。语法元素描述了该视频信号要怎么在解码器处重建, 包括了 CTU、PU、TU 的结构语法, 预测方式 (帧内还是帧间), 预测参数以及预测残差,SAO 偏移等。在 HEVC 中也有一些语法元素并不经过 CABAC 编码, 其他的高级语法元素采用零阶指数 Golomb 码或定长码来编码, 具体哪些需要编码见 [**?**]。

6.3.1 CABAC 概览

CABAC 编码主要包括三个基本步骤:

1. 二进制化。
2. 上下文建模。

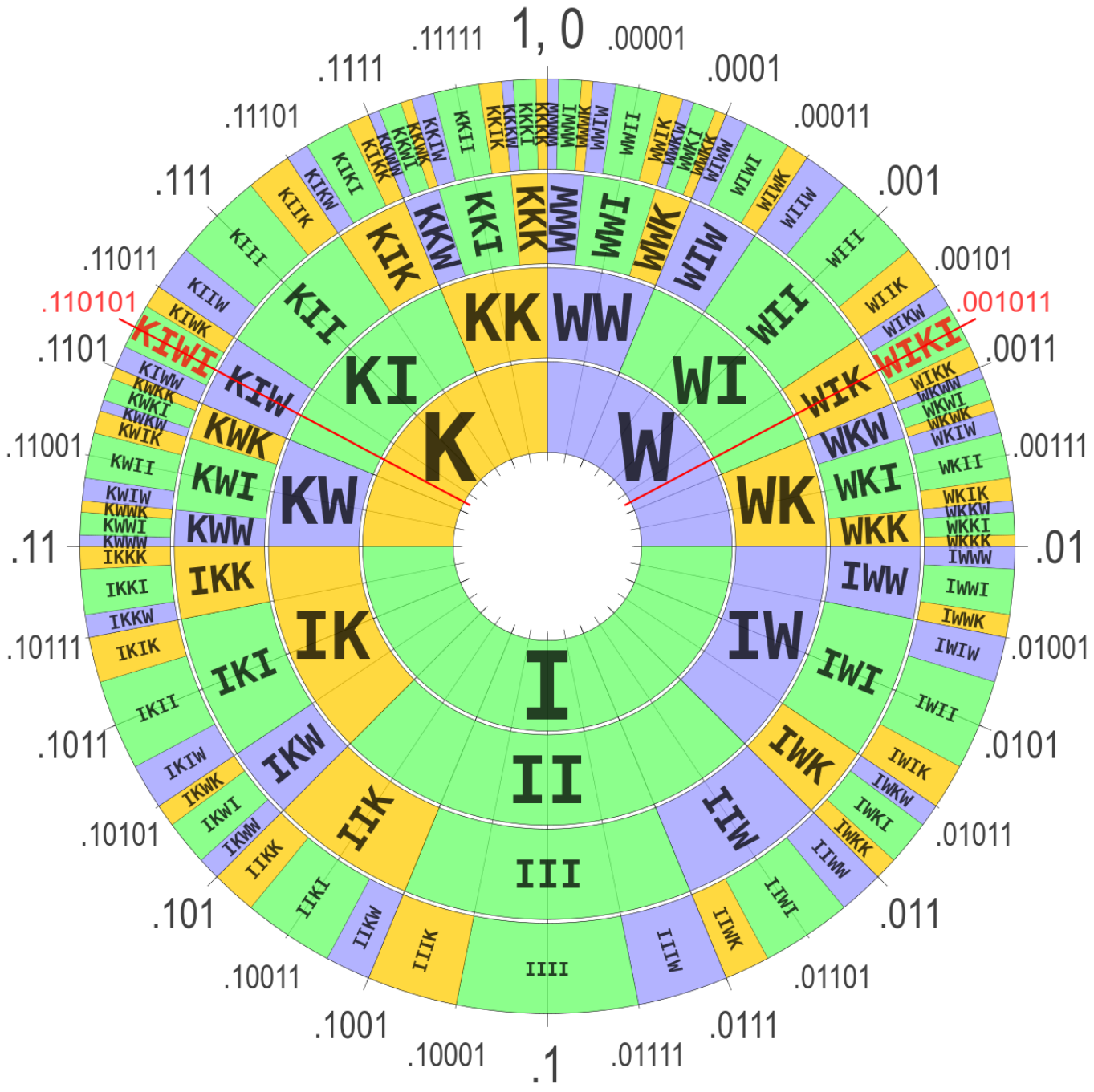

图 6.1: The above example visualised as a circle, the values in red encoding "WIKI" and "KIWI" https://en.wikipedia.org/wiki/Arithmetic_coding

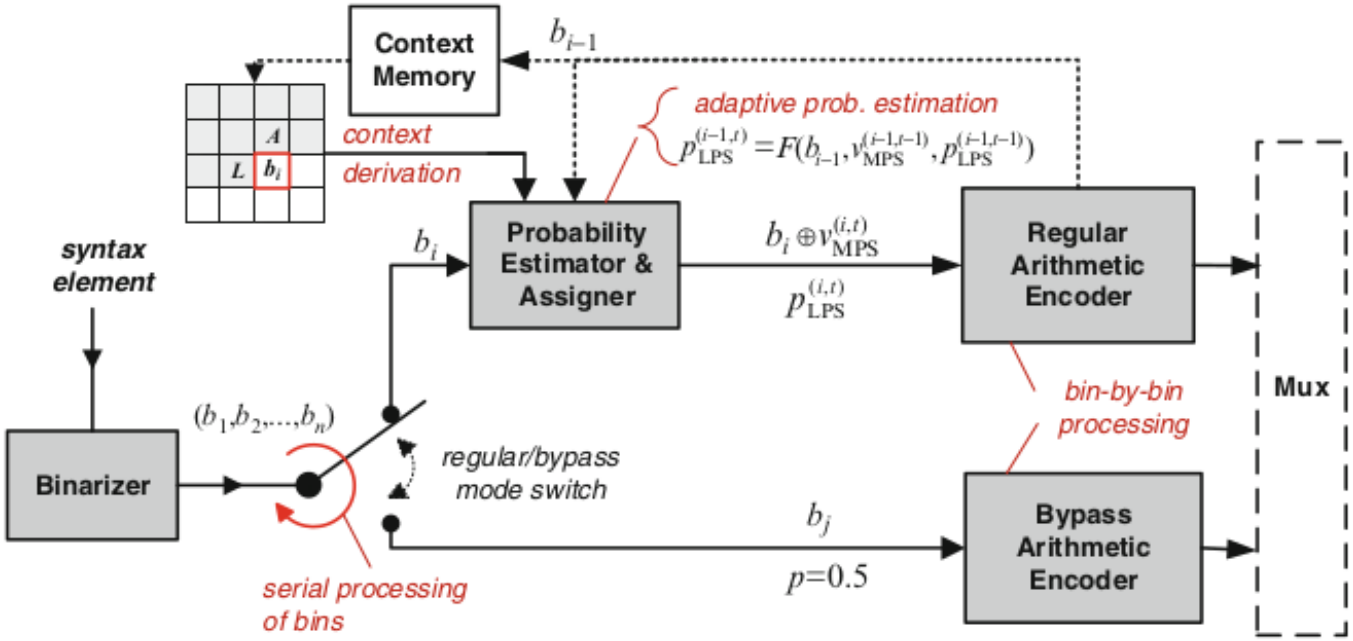

图 6.2: CABAC block diagram (from the encoder perspective): Binarization, context modeling (including probability estimation and assignment), and binary arithmetic coding. In red: Potential throughput bottlenecks, as further discussed from the decoder perspective in Sect. 8.3.2

3. 二进制算数编码。

图6.2为其具体实现框图。二进制化指将语法元素映射为二进制符号, 上下文建模预测每个常规编码符号概率, 二进制算数编码将符号编码为二进制比特流。

6.3.2 二进制化

CABAC 编码策略是找到一种对非二进制语法元素有效的编码方案, 像是运动矢量差与变换系数等可以根据先验知识设计出很有效的编码方案来预处理, 之后的上下文建模与算数编码基于这个来进行。H.264/AVC 与 HEVC 的二值化手段都是基于一些基础的可快速实现的概率模型设计的。

HEVC 二值化手段包括 k 阶截断 Rice 编码 (k-th order truncated Rice,TRk),k 阶指数 Golomb 编码 (k-th order Exp-Golomb,EGk), 定长二进制化 (fixed-length,FL) 方法, 其中部分方法也备用在了 H.264/AVC 上。表6.1表示了用这些方法编码 N 的结果。

- 一元码编码后前 N 个符号为 1, 最后一个符号为 0。当遇到 0 时代表编码结束。对 TrU(Truncated Unary), 当数值大于 cMax 则截断该码字。
- k 阶截断 Rice 码由前缀与后缀组成: 前缀为 N«k, 最大值为 cMax; 后缀为 N 的最后 k 个符号。
- k 阶 Golomb 码对于参数未知或可变的几何分布有接近最优的编码性能。每个码字由一个长为 $l_N + 1$ 的一元码字与长为 $l_N + k$ 的后缀组成, 其中 $l_N = \lfloor log_2((N >> k) + 1) \rfloor$
- 定长码使用长度为 $\lceil log_2(cMax + 1) \rceil$ 的码字表示。

表 6.1: Examples of different binarizations

		TrU	TRk	EG	FL
N	Unary(U)	cMax=7	k=1;cMax=7	k=0	cMax=7
0	0	0	00	1	000
1	10	10	01	010	001
2	110	110	100	011	010
3	1110	1110	101	00100	011
4	11110	11110	1100	00101	100
5	111110	111110	1101	00110	101
6	1111110	1111110	1110	00111	110
7	11111110	1111111	1111	0001000	111

不同的语法元素使用不同的二进制化方案, 有些还与以前编码过的元素以及参数有关。大多数语法元素均采用上述方案或者其中几种的组合, 还有些语法元素使用自定义的二进制化方案, 具体内容参见 [?]。

6.3.3 上下文建模

语法元素二进制化完成后, 其上下文建模方法依赖于编码模式。这包括两种模式: 常规编码模式与旁路编码模式。旁路编码模式不对概率模型进行自适应更新, 而是假设每个符号服从均匀分布。在常规编码模式中, 概率模型或根据语法元素种类, 其符号索引 (binIdx) 等固定选择, 或根据其相邻块的编码信息自适应选择。这种概率模型的选择被称为上下文建模。

作为一项重要的设计决策, 第二种情况通常只在最常见到的符号上使用, 而第一种则一般来说会使用联合零阶概率模型来处理。通过这种方式,CABAC 使得次符号等级的自适应概率建模变成了可能, 因此也能更好地利用起符号间冗余而不需很费力地为其建模。需要注意的是, 原则上由一种模型转换到另一种模型可发生在任何两个连续编码的符号中间。大体上 CABAC 中上下文模型的设计目的就是在避免没必要的建模消耗与充分利用符号间依赖关系中作出很好的权衡。

CABAC 中概率模型的参数也是自适应的, 即是说它以输入比特为单位自适应更新其模型参数, 这一过程被称为概率估计。为了这个目的,CABAC 中的每个概率模型都可取 126 种不同的状态, 其概率位于区间 $[0.01875, 0.98125]$。每个概率模型有两个参数, 被储存为 7-bit 的条目 (entry): 6bits 用来存储 63 种概率状态, 代表最小概率符号 (LPS) 的模型概率 p_{LPS}; 1bit 用来存储 v_{MPS}, 代表当前最大概率符号 (MPS)。CABAC 中概率估计依赖于被称为 “指数退化” 的模型, 在 t 时刻编码完一个符号 b 后概率更新如下:

$$p_{LPS}^{(t+1)} = \begin{cases} \alpha * p_{LPS}^{(t)} & if\ b = v_{MPS} \\ 1 - \alpha * (1 - p_{LPS}^{(t)}) & otherwise \end{cases} \tag{6.2}$$

该式中, 因子 α 决定了适应速度, 当 $\alpha \to 1$ 时代表最慢的适应速度,α 越小适应速度越快。在 CABAC

设计中, 等式6.2使用的因子 α 如下:

$$\alpha = \left(\frac{0.01875}{0.5}\right)^{\frac{1}{63}} with \min_t p_{LPS}^{(t)} = 0.01875 \tag{6.3}$$

且使用合理的量化将其分为 63 种状态, 来得到一有限状态机 (Finite-state Machine,FSM), 对应一转移概率表。在 HEVC 中这一方式不变, 尽管有其他的提案 [?, ?] 指出可以在跟高的运算复杂度下使平均比特率降低 $0.8 - 0.9\%$。

CABAC 中每一个概率模型都使用唯一的上下文索引 (ctxIdx) 指代, 其中该索引或由固定分配产生, 或由对应上下文逻辑计算得出。在 HEVC 标准化过程中花费了大量精力来在吞吐量与编码效率方面改进模型分配与上下文导出逻辑。

6.3.4 无乘法二进制算术编码

二进制算术编码, 或者算数编码笼统来说都基于递归区间划分原理。最初给定区间由其下界 (base)L 与其宽度 (range)R 表示, 接下来它被分为两个不相交子区间: 一个宽度为

$$R_{LPS} = p_{LPS} * R \tag{6.4}$$

的 LPS 区间, 与其对偶的长为 $R_{MPS} = R - R_{LPS}$ 的 MPS 区间。根据要编码的为 LPS 还是 MPS, 对应的子区间被选为新的编码区间, 通过这样的递归划分, 编码器最终输出区间 $[L^{(N)}, L^{(N)} + R^{(N)})$ 中最短的一个值作为编码结果。为了保证以有限精度表示 $R^{(j)}$ 与 $L^{(j)}$, 在区间划分时需要进行重归一化操作。每当 $R^{(j)}$ 低于一阈值就进行归一化操作, 同时输出其起始无歧义比特。

解码器处可以通过追踪区间来很快地恢复编码比特。因为 CABAC 只有两个符号, 根据式6.4仅需比较一次编码值与边界值就可判断输入比特为 0 还是 1。

从具体实现角度来说,CABAC 中最耗时的操作是式6.4中的乘法, 如果概率预测基于缩放累积频率计数方法的话, 运算过程中还会包含整数除法, 这会进一步降低性能, 实际上6.2中作者写的代码就是使用的这种方法。因此为了解决这个问题, 实际上在 H.264/AVC 标准化过程中已经提出了一族无乘法二进制算数编码方法, 这在之后被称为 *modulo coder*(M coder) [?, ?]。这种方法主要的特性是将基于查表的子区间划分法与上述的 FSM 概率预测法结合, 以及快速的旁路编码模式。

6.3.4.1 常规编码模式

M-coder 中区间划分的基本思想是将重归一化后的区间长度的可能区间长度量化为少量的 K 个单元。为了简化运算, 一般采用均匀量化, 其中 $K = 2^\kappa$, 最终可得到集合 $\mathbf{W} = W_0, W_1, \cdots, W_{K-1}$ 代表区间宽度。将其与 LPS 概率集 $\mathbf{P} = p_0, p_1, \cdots, p_{N-1}$ 分别相乘可以得到一大小为 $K \times N$ 的列表 $\{W_k * p_n | 0 \le k < K; 0 \le n < N\}$, 则可以通过选定精度来逼近式6.4所得结果。二维查找表 *TabRangeLPS* 中的条目以概率状态索引 n 与量化块索引 *k(R)* 表示。*k(R)* 的计算可以采用移位运算与位掩码来进行, 而掩码可被解释为取模运算, 这也是 M-coder 名称的由来。

在 H.264/AVC 中选择 $\kappa = 2$ 与 $N = 64$, 为了使表格最大大小 $2^\kappa * N \le 256$,HEVC 也采用了这种设计。若选定 $\kappa = 0$, 二维表 *TabRangeLPS* 退化为一维表, 等价于区间长度不变。这与 JBIG 中的 Q coder,JPEG 中的 QM coder,JPEG2000 中的 MQ coder 中的做法一致, 因此可以将 M-coder 视为更一般化的 Q-coder。相比于 QM/MQ coder,H.264/AVC 中的 M-coder 的吞吐量增加了 18%,

同时比特率降低了 $2-4\%$。有趣的是吞吐量的上升主要归功与它的旁路编码模式, 因为虽然查找表大小的增加可以提高编码效率, 同时它也会带来吞吐量的降低。

6.3.4.2 旁路编码模式

旁路编码模式中每个符号概率都被视为 0.5, 因此每次区间划分只需要一次比特移位, 比较, 以及可能有的减法便可实现。在 H.264/AVC 中旁路编码模式主要用在符号以及量化系数的最低有效位上, 然而在 HEVC 中绝大多数的二进制符号都采用旁路编码模式。如前文所述, 这也在很大程度上是 HEVC 为各个语法元素精心设计的二进制化方案的功劳, 在这种情况下二进制化后的语法元素已经接近了最优前缀码。

6.3.4.3 快速重归一化

任何算术编/解码吞吐量的主要瓶颈之一是重归一化过程。H.264/AVC 与 HEVC 中该过程需要以位为单位比较判断是否需要进一步归一化, 同时把结果输出到比特流, 也因此降低了吞吐量。文献 [?] 指出可以通过字节或字为单位来提高吞吐量, 具体细节见 [?, ?]。

6.3.4.4 结束标识

在解码时区间细分会不断进行下去, 因此为了表示解码结束需要有对应标识。在 M-coder 中有一个保留概率状态, 对应索引为 $n=63$, 此时 $R_{LPS}=2$。其结果是对 `end_of_slice_segment_flag`, `end_of_sub_stream_one_bit` 和 `pcm_flag` 等终止标识, 在重归一化过程中生成 7bits 输出, 然后再输出 2bits 来终止该码字。在编码器处最后写入的比特必为 1, 代表 `rbsp_stop_one_bit`。在对比特流进行封装前, 二进制码字以零值填充以按字节对齐。

6.3.5 CABAC 流程

以下为 HEVC 中 CABAC 常规编码具体实现流程: 对每一上下文索引均分配了一个初始值 $initValue$, 根据下式计算其状态 $ucState$, 包括其概率状态索引 n 与其 MPS,$ucState = n << 1 + MPS$:

$$
\begin{aligned}
qp &= clip3(0, 51, qp) \\
slope &= (initValue >> 4) * 5 - 45 \\
offset &= ((initValue \& 15) << 3) - 16 \\
initState &= clip3(1, 126, (((slope * qp) >> 4) + offset)) \\
mpState &= initState >= 64 \\
ucState &= ((mpState?(initState - 64) : (63 - initState)) << 1) + mpState
\end{aligned}
$$

其中 qp 为片层量化参数,`clip3` 为动态限幅函数, 原型为 `clip3(min,max,v)`, 仅当 `v` 在 `min` 与 `max` 间时输出 `v`, 其余时刻输出边界值。

概率模型自适应更新发生在每个二进制符号编码后, 其更新方法为:

```
if( binVal = = valMps )
```

表 6.2: State transition table

pStateIdx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
transIdxLps	0	0	1	2	2	4	4	5	6	7	8	9	9	11	11	12
transIdxMps	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
pStateIdx	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
transIdxLps	13	13	15	15	16	16	18	18	19	19	21	21	22	22	23	24
transIdxMps	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
pStateIdx	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
transIdxLps	24	25	26	26	27	27	28	29	29	30	30	30	31	32	32	33
transIdxMps	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
pStateIdx	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
transIdxLps	33	33	34	34	35	35	35	36	36	36	37	37	37	38	38	63
transIdxMps	49	50	51	52	53	54	55	56	57	58	59	60	61	62	62	63

```
  pStateIdx = transIdxMps( pStateIdx )
else {
  if( pStateIdx = = 0 )
    valMps = 1 - valMps
  pStateIdx = transIdxLps( pStateIdx )
}
```

状态转移表见??。

解码过程:

1. LPS 区间 ivlLpsRange 更新如下:
   - 由当前区间 ivlCurrRange, 得量化块索引 qRangeIdx 如下:
     `qRangeIdx =( ivlCurrRange >> 6 ) & 3`
   - 根据 pStateIdx 和 qRangeIdx 更新下界 ivlLpsRange 如下:
     `ivlLpsRange = rangeTabLps[ pStateIdx ][ qRangeIdx ]`
2. 首先将 ivlCurrRange 置为 ivlCurrRange-ivlLpsRange, 而后进行下一步:
   - 若 ivlOffset 不小于 ivlCurrRange,binVal 置为 1-valMps,ivlOffset 自减 ivlCurrange,ivlCurrRange 置为 ivlLpsRange
   - 否则 binVal 置为 valMps

rangeTabLps 见表??。

表 6.3: Specification of rangeTabLps depending on the values of pStateIdx and qRangeIdx

pStateIdx	qRangeIdx				pStateIdx	qRangeIdx			
	0	1	2	3		0	1	2	3
0	128	176	208	240	32	27	33	39	45
1	128	167	197	227	33	26	31	37	43
2	128	158	187	216	34	24	30	35	41
3	123	150	178	205	35	23	28	33	39
4	116	142	169	195	36	22	27	32	37
5	111	135	160	185	37	21	26	30	35
6	105	128	152	175	38	20	24	29	33
7	100	122	144	166	39	19	23	27	31
8	95	116	137	158	40	18	22	26	30
9	90	110	130	150	41	17	21	25	28
10	85	104	123	142	42	16	20	23	27
11	81	99	117	135	43	15	19	22	25
12	77	94	111	128	44	14	18	21	24
13	73	89	105	122	45	14	17	20	23
14	69	85	100	116	46	13	16	19	22
15	66	80	95	110	47	12	15	18	21
16	62	76	90	104	48	12	14	17	20
17	59	72	86	99	49	11	14	16	19
18	56	69	81	94	50	11	13	15	18
19	53	65	77	89	51	10	12	15	17
20	51	62	73	85	52	10	12	14	16
21	48	59	69	80	53	9	11	13	15
22	46	56	66	76	54	9	11	12	14
23	43	53	63	72	55	8	10	12	14
24	41	50	59	69	56	8	9	11	13
25	39	48	56	65	57	7	9	11	12
26	37	45	54	62	58	7	9	10	12
27	35	43	51	59	59	7	8	10	11
28	33	41	48	56	60	6	8	9	11
29	32	39	46	53	61	6	7	9	10
30	30	37	43	50	62	6	7	8	9
31	29	35	41	48	63	2	2	2	2

6.3.6 运行效率

为了验证 CABAC 算法执行效率, 本章作者以 HM 源代码为基础实现了常规编码模式下的熵编码器, 托管于`github`。因为要体现压缩效率的话需要对应语法元素支持, 这里只列出算法执行速度为 8.7MB/s。而旁路编码模式只会更快, 且还可以通过并行计算来实现, 因此实际效率会高于该值。

7 其他