# HEVC 关键技术

随着数字视频应用产业链的快速发展，**视频应用**向以下几个方向**发展**的**趋势**愈加明显：

(1) **高清晰度**(HigherDefinition)：数字视频的应用格式从 720 P 向 1080 P 全面升级，在一些视频应用领域甚至出现了 4K x 2K、8K x 4K 的数字视频格式；

(2) **高帧率**(Higherframe rate )：数字视频帧率从 30fps 向 60fps、120fps 甚至 240fps 的应用场景升级；

(3) **高压缩率**(HigherCompression rate )：传输带宽和存储空间一直是视频应用中最为关键的资源，因此，在有限的空间和管道中获得最佳的视频体验一直是用户的不懈追求。

由于数字视频应用在发展中面临上述趋势，如果继续采用 **H.264 编码**就出现的如下一些**局限性**：

(1) **宏块个数的爆发式增长**，会导致用于编码宏块的预测模式、运动矢量、参考帧索引和量化级等宏块级参数信息所占用的码字过多，用于编码残差部分的码字明显减少。

(2) 由于分辨率的大大增加，**单个宏块所表示的图像内容的信息大大减少**，这将导致相邻的 4 x 4 或 8 x 8 块变换后的低频系数相似程度也大大提高，导致出现大量的冗余。

(3) 由于分辨率的大大增加，表示同一个运动的**运动矢量的幅值将大大增加**，H.264 中采用一个运动矢量预测值，对运动矢量差编码使用的是哥伦布指数编码，该编码方式的特点是数值越小使用的比特数越少。因此，随着运动矢量幅值的大幅增加，H.264 中用来对运动矢量进行预测以及编码的方法压缩率将逐渐降低。

(4) H.264 的一些关键算法例如采用 CAVLC 和 CABAC 两种基于上下文的**熵编码方法、deblock 滤波**等都要求**串行编码，并行度**比较**低**。针对 GPU/DSP/FPGA/ASIC 等并行化程度非常高的 CPU，H.264 的这种串行化处理越来越成为制约运算性能的瓶颈。

为了面对以上发展趋势，2010 年 1 月，ITU-T VCEG 和 ISO/IEC MPEG 联合成立 JCT-VC 了联合组织，统一制定下一代编码标准：HEVC(High Efficiency Video Coding)。

HEVC 协议标准计划于 2013 年 2 月份正式在业界发布，目前整个框架结构已基本确定。与他的前辈 H.264 相比，H.265 具有更高的（接近于两倍）压缩比率，在同码率下具有更佳的视频质量，而且支持 8k UHD 超高清视频。

## 1. HEVC 的新特征

### ● 高压缩率

HEVC 仍属于基于块的，预测加变换的混合编码框架，如图 1 所示。然而其采用更加灵活的编码结构来提高编码效率，包括编码单元（CU）、预测单元（PU）和变换单元（TU）。其中编码单元类似于 H.264 中的宏块的概念，预测单元是进行预测的基本单元，变换单元是进行变换和量化的基本单元。在此混合编码框架下，HEVC 进行了大量的技术创新，其中具有代表性的技术方案有：基于超大尺寸四叉树的分割结构和残差编码结构，多角度帧内预测技术，运动估计融合技术，高精度运动补偿技术，自适应环路滤波技术以及基于语义的熵编码技术。

HEVC 特别为逐行扫描视频信号设计，并未专门研究隔行扫描信息。为此，HEVC 发送“元数据”来解释隔行信息如何传输：或者两场数据分别作为一帧编码，或者将两场数据合并成一帧进行编码传输。这样 HEVC 就不需要专门设计隔行扫描的解码器。

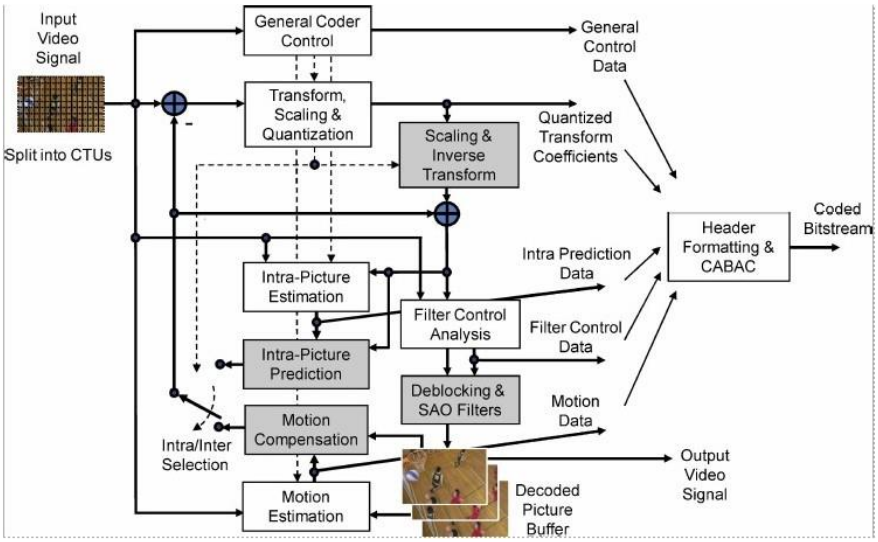



图 1. 典型的 HEVC 视频编码器框架 (灰色高亮解码器模块).

### ● 可并行性

当前芯片架构已经从单核向同构多核并行方向发展，因此为了适应并行化程度高的芯片实现，HEVC 引入了很多并行运算的优化思路，主要包括以下几个方面：

**1) Tile**

如图 2 所示，在 slice 划分的基础上用垂直的边界将图像划分为一些列，划分出的矩形区域为一个 Tile，每一个 Tile 包含整数个 LCU（Largest CU），Tile 之间可以互相独立，以此实现并行处理。

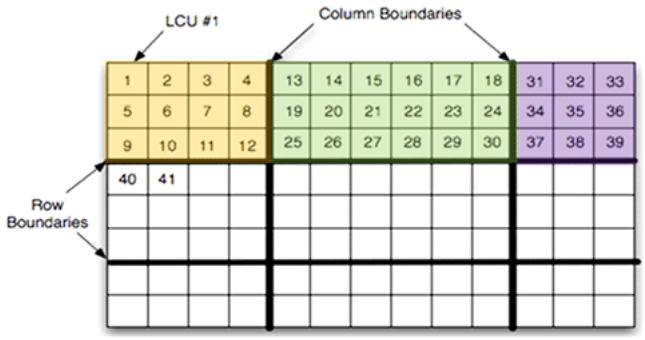



图 2. Tile 划分示意图

**2) Dependent Slice Segments**

Dependent slice 允许在一个 slice 内部再切分成多个 Dependent Slice Segments，每个 Dependent Slice Segment 可以独立的编码和解码，从而提高了编解码器的并行处理能力；

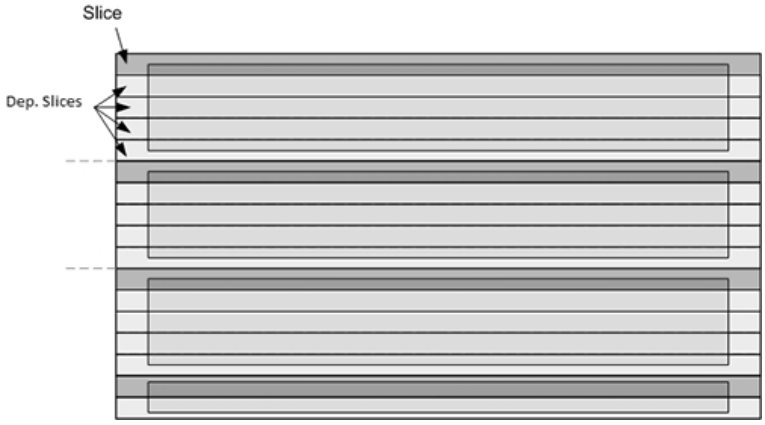



图 3. Dependent Slice 示意图

**3) WPP**(Wave-front Parallel Processing)

熵编码环节中，编码单元上下文存在依赖关系。在 HEVC 中，上一行的第二个 LCU 处理完毕，即对当前行的第一个 LCU 的熵编码概率状态参数进行初始化，如图 4 所示。因此，只需要上一行的第二个 LCU 编解码完毕，即可以开始当前行的编解码，以此提高编解码器的并行处理能力。

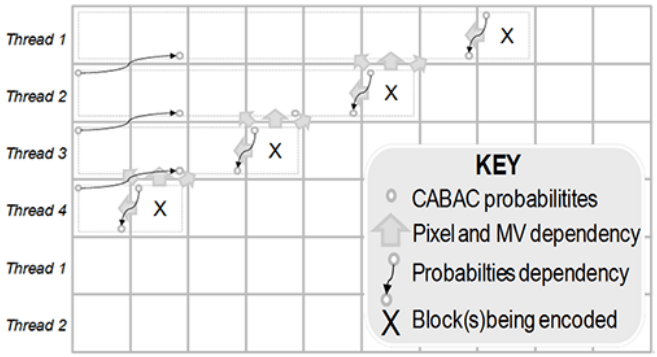



图 4. WPP 示意图

语法：

| | |
|---|---|
| tiles_enabled_flag | -unavailable |
| entropy_coding_sync_enabled_flag | -available in x265 |
| dependent_slice_segments_enabled_flag | -unavailable |

## 2. HEVC 编码工具

- **基于四叉树结构的分割技术**

HEVC 放弃了当前各种编码标准中“宏块”的概念，而是采用了超大尺寸四叉树编码结构。该结构使用编码单元(Coding Unit, CU)，预测单元(Prediction Unit, PU)和变换单元(Transform unit, TU) 三个概念描述整个编码过程。这三个单元的分离，使得变换、预测和编码，各个处理环节更加灵活，也有利于各环节的划分更加符合视频图像的纹理特征，有利于各个单元更优化地完成各自的功能。

**1) 编码树单元（CTU）与编码树块（CTB）**

一帧图像分割成编码树单元 CTUs，CTU 包含一个亮度和两个色度分量 CTBs，CTU 是编解码处理的基本单元。亮度分量 CTB 为边长为 16、32 或者 64 的**方块**，色度 CTB 的边长为亮度的一半，有关 CTB 大小 CtbSizeY 的语法元素在 SPS 中定义。更大尺寸的分块可以更好地处理高清视频信号，其编码效率也越高。

| coding_tree_unit( ) { | Descriptor |
|---|---|
| xCtb = ( CtbAddrInRs % PicWidthInCtbsY ) << CtbLog2SizeY | |
| yCtb = ( CtbAddrInRs / PicWidthInCtbsY ) << CtbLog2SizeY | |
| if( slice_sao_luma_flag \|\| slice_sao_chroma_flag ) | |
| sao( xCtb >> CtbLog2SizeY, yCtb >> CtbLog2SizeY ) | |
| coding_quadtree( xCtb, yCtb, CtbLog2SizeY, 0 ) | |
| } | |

SPEC 附录 A 规定，HEVC Level 5 及以上等级要求 CTB 大小至少为 32×32 或 64×64。

**2) 编码单元（CU）和编码块（CB）**

亮度和色度 CTB 可以按照四叉树结构进一步分解为 CBs。四叉树结构可以根据视频信号的区域特征将 CTB 分割成合适大小的 CB 块。分割过程可反复迭代，直到亮度 CB 的大小达到最小可允许尺寸 MinCbSizeY，通常是 8x8 或更大；相关语法元素在 SPS 中定义。如图 5 所示。在图像的边界处，CB 为最小支持尺寸。解码器对可能“越界”的 CTB 会做专门处理。

log2_min_luma_coding_block_size_minus3

log2_diff_max_min_luma_coding_block_size

MinCbLog2SizeY = log2_min_luma_coding_block_size_minus3 + 3 (7-10)

CtbLog2SizeY = MinCbLog2SizeY + log2_diff_max_min_luma_coding_block_size (7-11)

**MinCbSizeY** = 1 << MinCbLog2SizeY (7-12)

**CtbSizeY** = 1 << CtbLog2SizeY (7-13)

PicWidthInMinCbsY = pic_width_in_luma_samples / MinCbSizeY (7-14)

PicWidthInCtbsY = Ceil( pic_width_in_luma_samples ÷ CtbSizeY ) (7-15)

PicHeightInMinCbsY = pic_height_in_luma_samples / MinCbSizeY (7-16)

PicHeightInCtbsY = Ceil( pic_height_in_luma_samples ÷ CtbSizeY ) (7-17)

PicSizeInMinCbsY = PicWidthInMinCbsY * PicHeightInMinCbsY (7-18)

PicSizeInCtbsY = PicWidthInCtbsY * PicHeightInCtbsY (7-19)

PicSizeInSamplesY = pic_width_in_luma_samples * pic_height_in_luma_samples (7-20)

PicWidthInSamplesC = pic_width_in_luma_samples / SubWidthC (7-21)

PicHeightInSamplesC = pic_height_in_luma_samples / SubHeightC (7-22)

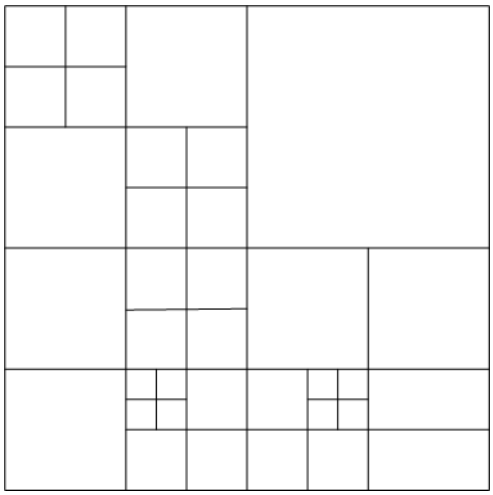


图 5. CTB 的树形划分结构示意图

| coding_quadtree( x0, y0, log2CbSize, cqtDepth ) { | Descriptor |
|---|---|
| if( x0 + ( 1 << log2CbSize ) <= pic_width_in_luma_samples && y0 + ( 1 << log2CbSize ) <= pic_height_in_luma_samples && log2CbSize > MinCbLog2SizeY ) | |
| split_cu_flag[ x0 ][ y0 ] | ae(v) |
| if( cu_qp_delta_enabled_flag && log2CbSize >= Log2MinCuQpDeltaSize ) { | |
| IsCuQpDeltaCoded = 0 | |
| CuQpDeltaVal = 0 | |
| } | |
| if( split_cu_flag[ x0 ][ y0 ] ) { | |
| x1 = x0 + ( 1 << ( log2CbSize • 1 ) ) | |
| y1 = y0 + ( 1 << ( log2CbSize • 1 ) ) | |
| coding_quadtree( x0, y0, log2CbSize − 1, cqtDepth + 1 ) | |
| if( x1 < pic_width_in_luma_samples ) | |
| coding_quadtree( x1, y0, log2CbSize − 1, cqtDepth + 1 ) | |
| if( y1 < pic_height_in_luma_samples ) | |
| coding_quadtree( x0, y1, log2CbSize − 1, cqtDepth + 1 ) | |
| if( x1 < pic_width_in_luma_samples && y1 < pic_height_in_luma_samples ) | |
| coding_quadtree( x1, y1, log2CbSize − 1, cqtDepth + 1 ) | |
| } else | |
| coding_unit( x0, y0, log2CbSize ) | |
| } | |

一个 CU 由一个亮度 CB、两个色度 CBs 和关联的语法元素构成。一个 CTB 可以只包含一个 CU，也可以包含好几个 CUs，每一个 CU 都有一个分区关联的预测单元（PU）和一个变换树单元（TU）。

```
coding_unit( x0, y0, log2CbSize ) {
    if( transquant_bypass_enabled_flag )
        cu_transquant_bypass_flag
    if( slice_type != I )
        cu_skip_flag[ x0 ][ y0 ]
    nCbS = ( 1 << log2CbSize )
    if( cu_skip_flag[ x0 ][ y0 ] )
        prediction_unit( x0, y0, nCbS, nCbS )
    else {
        if( slice_type != I )
            pred_mode_flag
        if( CuPredMode[ x0 ][ y0 ] != MODE_INTRA | | log2CbSize = = MinCbLog2SizeY )
            part_mode
        if( CuPredMode[ x0 ][ y0 ] = = MODE_INTRA ) {
            if( PartMode = = PART_2Nx2N && pcm_enabled_flag &&
              log2CbSize >= Log2MinIpcmCbSizeY &&
              log2CbSize <= Log2MaxIpcmCbSizeY )
                pcm_flag[ x0 ][ y0 ]
            if( pcm_flag[ x0 ][ y0 ] ) {
                while( !byte_aligned( ) )
                    pcm_alignment_zero_bit
                pcm_sample( x0, y0, log2CbSize )
            } else {
                pbOffset = ( PartMode = = PART_NxN ) ? ( nCbS / 2 ) : nCbS
                for( j = 0; j < nCbS; j = j + pbOffset )
                    for( i = 0; i < nCbS; i = i + pbOffset )
                        prev_intra_luma_pred_flag[ x0 + i ][ y0 + j ]
                for( j = 0; j < nCbS; j = j + pbOffset )
                    for( i = 0; i < nCbS; i = i + pbOffset )
                        if( prev_intra_luma_pred_flag[ x0 + i ][ y0 + j ] )
                            mpm_idx[ x0 + i ][ y0 + j ]
                        else
                            rem_intra_luma_pred_mode[ x0 + i ][ y0 + j ]
                intra_chroma_pred_mode[ x0 ][ y0 ]
            }
        } else {
            if( PartMode = = PART_2Nx2N )
                prediction_unit( x0, y0, nCbS, nCbS )
```

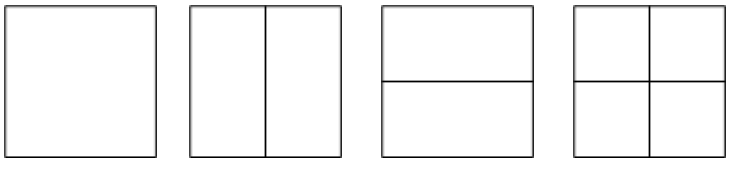
```
                else if( PartMode = = PART_2NxN ) {
                    prediction_unit( x0, y0, nCbS, nCbS / 2 )
                    prediction_unit( x0, y0 + ( nCbS / 2 ), nCbS, nCbS / 2 )
                } else if( PartMode = = PART_Nx2N ) {
                    prediction_unit( x0, y0, nCbS / 2, nCbS )
                    prediction_unit( x0 + ( nCbS / 2 ), y0, nCbS / 2, nCbS )
                } else if( PartMode = = PART_2NxnU ) {
                    prediction_unit( x0, y0, nCbS, nCbS / 4 )
                    prediction_unit( x0, y0 + ( nCbS / 4 ), nCbS, nCbS * 3 / 4 )
                } else if( PartMode = = PART_2NxnD ) {
                    prediction_unit( x0, y0, nCbS, nCbS * 3 / 4 )
                    prediction_unit( x0, y0 + ( nCbS * 3 / 4 ), nCbS, nCbS / 4 )
                } else if( PartMode = = PART_nLx2N ) {
                    prediction_unit( x0, y0, nCbS / 4, nCbS )
                    prediction_unit( x0 + ( nCbS / 4 ), y0, nCbS * 3 / 4, nCbS )
                } else if( PartMode = = PART_nRx2N ) {
                    prediction_unit( x0, y0, nCbS * 3 / 4, nCbS )
                    prediction_unit( x0 + ( nCbS * 3 / 4 ), y0, nCbS / 4, nCbS )
                } else { /* PART_NxN */
                    prediction_unit( x0, y0, nCbS / 2, nCbS / 2 )
                    prediction_unit( x0 + ( nCbS / 2 ), y0, nCbS / 2, nCbS / 2 )
                    prediction_unit( x0, y0 + ( nCbS / 2 ), nCbS / 2, nCbS / 2 )
                    prediction_unit( x0 + ( nCbS / 2 ), y0 + ( nCbS / 2 ), nCbS / 2, nCbS / 2 )
                }
            }
            if( !pcm_flag[ x0 ][ y0 ] ) {
                if( CuPredMode[ x0 ][ y0 ] != MODE_INTRA &&
                    !( PartMode = = PART_2Nx2N && merge_flag[ x0 ][ y0 ] ) )
                    rqt_root_cbf
                if( rqt_root_cbf ) {
                    MaxTrafoDepth = ( CuPredMode[ x0 ][ y0 ] = = MODE_INTRA ?
                        ( max_transform_hierarchy_depth_intra + IntraSplitFlag ) :
                        max_transform_hierarchy_depth_inter )
                    transform_tree( x0, y0, x0, y0, log2CbSize, 0, 0 )
                }
            }
        }
    }
```

**3) 预测块（PB）和预测单元（PU）**

CU 的预测模式分为帧内预测或帧间预测；

若预测模式为帧内预测，除了最小尺寸模式外，PB 的尺寸等于 CB。若是最小尺寸模式，则用标志位 part_mode 表示是否将 CB 分割成四个 PB 块。这样最小可以支持到 4×4 大小的帧内预测块（此时色度分量 PB 也为 4×4）。

若预测模式为帧间预测，亮度和色度 CB 可能分割为 1/2/4 个 PB。只有 CB 为最小尺寸模式（且 log2CbSize>3）才能分割为 4 个 PB 块。CB 分割为四个 PB 时，每个 PB 占据 CB 的 1/4；CB 分为两个 PB 时，可选择 6 种分割模式的一种，其中包含 4 种**非对称运动分割模式** AMP（log2cbSize>MinCbLog2SizeY 且 amp_enabled_flag==1）。每个帧间预测 PB 包含一个或者两个运动矢量和参考帧索引。为了控制编码带宽，帧间预测不允许使用 4×4 块，8×4 块和 4×8 块最多只允许使用于单向预测编码。



(a) 2N×2N　(b) N×2N　(c) 2N×N　(d) N×N

图 6. 4 种 PU 分割类型

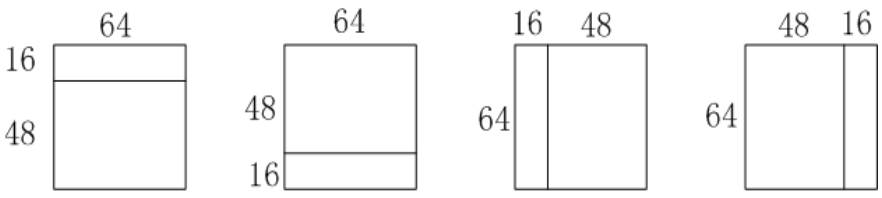



图 7. 64×64CU 所支持的 4 种 AMP 分割形态

PU 由亮度和色度 PBs，以及相应的预测语法组成；

amp_enabled_flag

| prediction_unit( x0, y0, nPbW, nPbH ) { | Descriptor |
|---|---|
| if( cu_skip_flag[ x0 ][ y0 ] ) { | |
| if( MaxNumMergeCand > 1 ) | |
| **merge_idx**[ x0 ][ y0 ] | ae(v) |
| } else { /* MODE_INTER */ | |
| **merge_flag**[ x0 ][ y0 ] | ae(v) |
| if( merge_flag[ x0 ][ y0 ] ) { | |
| if( MaxNumMergeCand > 1 ) | |
| **merge_idx**[ x0 ][ y0 ] | ae(v) |
| } else { | |
| if( slice_type == B ) | |
| **inter_pred_idc**[ x0 ][ y0 ] | ae(v) |
| if( inter_pred_idc[ x0 ][ y0 ] != PRED_L1 ) { | |
| if( num_ref_idx_l0_active_minus1 > 0 ) | |
| **ref_idx_l0**[ x0 ][ y0 ] | ae(v) |
| mvd_coding( x0, y0, 0 ) | |
| **mvp_l0_flag**[ x0 ][ y0 ] | ae(v) |
| } | |
| if( inter_pred_idc[ x0 ][ y0 ] != PRED_L0 ) { | |
| if( num_ref_idx_l1_active_minus1 > 0 ) | |
| **ref_idx_l1**[ x0 ][ y0 ] | ae(v) |
| if( mvd_l1_zero_flag && | |
| inter_pred_idc[ x0 ][ y0 ] == PRED_BI ) { | |
| MvdL1[ x0 ][ y0 ][ 0 ] = 0 | |
| MvdL1[ x0 ][ y0 ][ 1 ] = 0 | |
| } else | |
| mvd_coding( x0, y0, 1 ) | |
| **mvp_l1_flag**[ x0 ][ y0 ] | ae(v) |
| } | |
| } | |
| } | |
| } | |

**4) 变换单元（TU）和变换块（TB）**

CB 的残差信号可按照残差四叉树分解为变换块 TBs；如图 8 中所示。只有方形的 TB 划分是被允许的，SPS 指明了将会使用的最大亮度 TB 尺寸 MaxTrafoSize 和最小亮度 TB 尺寸 MinTrafoSize；除了 4×4 之外，色度 TB 的尺寸为亮度 TB 的一半。

| transform_tree( x0, y0, xBase, yBase, log2TrafoSize, trafoDepth, blkIdx ) { | Descriptor |
|---|---|
| if( log2TrafoSize <= Log2MaxTrafoSize && log2TrafoSize > Log2MinTrafoSize && trafoDepth < MaxTrafoDepth && !( IntraSplitFlag && ( trafoDepth == 0 ) ) ) | |
| **split_transform_flag**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| if( log2TrafoSize > 2 ) { | |
| if( trafoDepth == 0 \|\| cbf_cb[ xBase ][ yBase ][ trafoDepth − 1 ] ) | |
| **cbf_cb**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| if( trafoDepth == 0 \|\| cbf_cr[ xBase ][ yBase ][ trafoDepth − 1 ] ) | |
| **cbf_cr**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| } | |
| if( split_transform_flag[ x0 ][ y0 ][ trafoDepth ] ) { | |
| x1 = x0 + ( 1 << ( log2TrafoSize • 1 ) ) | |
| y1 = y0 + ( 1 << ( log2TrafoSize • 1 ) ) | |
| transform_tree( x0, y0, x0, y0, log2TrafoSize − 1, trafoDepth + 1, 0 ) | |
| transform_tree( x1, y0, x0, y0, log2TrafoSize − 1, trafoDepth + 1, 1 ) | |
| transform_tree( x0, y1, x0, y0, log2TrafoSize − 1, trafoDepth + 1, 2 ) | |
| transform_tree( x1, y1, x0, y0, log2TrafoSize − 1, trafoDepth + 1, 3 ) | |
| } else { | |
| if( CuPredMode[ x0 ][ y0 ] == MODE_INTRA \|\| trafoDepth != 0 \|\| cbf_cb[ x0 ][ y0 ][ trafoDepth ] \|\| cbf_cr[ x0 ][ y0 ][ trafoDepth ] ) | |
| **cbf_luma**[ x0 ][ y0 ][ trafoDepth ] | ae(v) |
| transform_unit( x0, y0, xBase, yBase, log2TrafoSize, trafoDepth, blkIdx ) | |
| } | |
| } | |

对于尺寸为 MxM 的亮度 CB，使用了 split_transform_flag 来指示它是否被切分成四个尺寸为 M/2xM/2 的 TBs 块；如果更进一步的划分是被允许的，那么需要在 SPS 中指定残差四叉树的最大深度 MaxTrafoDepth；在 CU 帧内预测时，最近相邻 TBs 的解码像素被用作帧内预测的参考数据；

max_transform_hierarchy_depth_intra

max_transform_hierarchy_depth_inter

MaxTrafoDepth = ( CuPredMode[ x0 ][ y0 ] == MODE_INTRA ?
( max_transform_hierarchy_depth_intra + IntraSplitFlag ) :
max_transform_hierarchy_depth_inter )

log2_min_transform_block_size_minus2

log2_diff_max_min_transform_block_size

log2MinTrafoSize = log2_min_transform_block_size_minus2 + 2

log2MaxTrafoSize = log2MinTrafoSize + log2_diff_max_min_transform_block_size

(且 Log2MaxTrafoSize <= Min(CtbLog2SizeY, 5))

相对于以前的标准，为了使帧间预测 CU 在四叉树结构的 TB 划分上获得最大的编码效率，HEVC 允许一个 TB 跨越多个 PB；

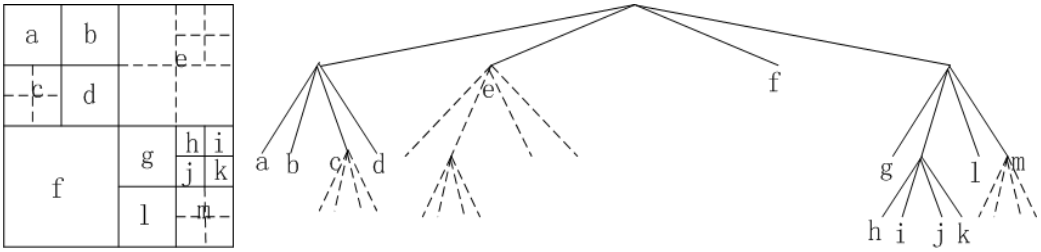



图 8. 编码单元、变换单元的四叉树结构关系图

- **帧内预测**

帧内预测是以 TB 尺寸进行操作的；在空域上相邻的、已重建的边界像素将被用作预测参考信号；对于从 4×4 到 32×32 大小的 TB 块而言，定义了 33 种预测方向以及 DC 和 planar 两种特殊模式。所有可能的预测方向如图 9 中所示；对于色度分量，可以显式指定水平、垂直、planar 和 DC 模式，也可以采用跟亮度预测一致的模式。同 H.264 类似，HEVC 的帧内预测在所有类型的 slice 中都可支持。

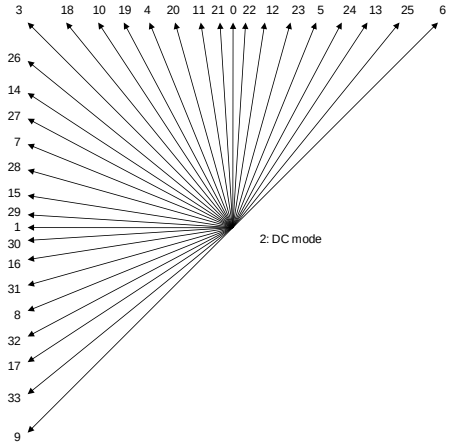



图 9. 34 种帧内预测方式

下面将对帧内预测技术做进一步的解析；

1) **预测块(PB)的划分**

帧内预测 PB 划分类型 PartMode：PART_2Nx2N、PART_NxN；

PART_2Nx2N，表示这个 CB 不能再被划分；

PART_NxN，表示这个 CB 可以再被划分成四个相同尺寸(N = M/2)的 PB；

注意，帧内预测 PB 块定义了帧内预测的语法元素，但它并不等价于帧内预测单元；此时，也没有真正意义上的 PU。

2) **Intra_Angular**

帧内预测在 H.264 中得到了成功的应用；在 HEVC 中，随着 TB 尺寸的增加，提供更多可选的预测方向；

相对于 H.264 中的 8 个帧内预测方向，HEVC 支持 33 个预测方向，总称为 Intra_Angular[k]， 其中 k 为 2 到 34；这些角度是对相邻的水平和垂直边界在统计学上的信号预测处理；

当使用 Intra_Angular 模式时，每个 TB 都是使用空域上相邻的已重建的像素来预测；如图 9 中所示，对于一个尺寸为 NxN 的 TB， 它一共有 4N+1 个相邻像素用来预测；在 HEVC 中，当先前 TB 解码操作有效时，左下角相邻 TB，左相邻 TB，上和右上相邻 TB 的相邻边界像素均可用作预测；Intra_Angular 模式的预测处理可以从给定的方向的像素位置中推断得到；

为了统一预测算法，对于 Intra_Angular[k]中，k 为 2-17 时，位于上行的像素被投影成位于左列的额外像素；k 为 18-34 时，位于左列的像素被投影成位于上行的像素；为了提高帧内预测的精度，被投影的参考像素精度位置使用 1/32 像素精度；为了获得被投影的参考像素，需要对位于整像素位置间的两个最近的参考像素使用线性插值；

Intra_Angular 模式的预测处理可以用于所有块尺寸和预测方向；相对于 H.264 中，不同的块尺寸(4x4，8x8，16x16)使用不同的预测模式，这个设计更适合大的 TB 尺寸预测；

3) **Intra_Planar 和 Intra_DC**

除了 Intra_Angular 预测外，HEVC 还和 H.264 一样，支持 Intra_Planar，Intra_DC 预测模式；

Intra_DC，使用参考像素的均值进行预测；

Intra_Planar，使用四个角的参考像素得到的两个线性预测的均值；

在 HEVC 中，Intra_Planar 预测模式可用于所有块尺寸，不像在 H.264 中，planar 预测模式只能用在 16x16 的亮度块，且它们的方法也不一样；

**4) 参考像素平滑**

在 HEVC 中，用于帧内预测的参考像素可以使用三阶([1 2 1]/4)平滑滤波器来平滑，这个和 H.264 中的 8x8 帧内预测模式一样；HEVC 可以根据预测方向、不连续的数量和块大小对平滑操作灵活使用；而在 H.264 中，平滑操作不能用于 4x4 块，对于 8x8 块，也只能用于对角线预测方向；HEVC 则可以对 Intra_Angular[k]，k=2，18，34，使用参考像素平滑；对于 16x16 块，除了近水平和近垂直方向外，可以对 k 为 9-11 和 25-27 预测方向使用参考像素平滑滤波；对于 32x32 块，除了水平(k=10)和垂直(k=26)方向外，可以对所有剩下的预测方向使用参考像素平滑滤波；当不连续侦测的数量超过阈值时，可以对三个相邻区域像素使用线性插值来做平滑预测；当 TB 尺寸等于或大于 8x8 时，Intra_Planar 模式也可以使用平滑滤波；

strong_intra_smoothing_enabled_flag

**5) 边界值平滑**

为了去掉块边界的不连续性，对于这三种模式，

Intra_DC(mode 1)，

Intra_Angular[k]， k = 10，26.

当 TB 的尺寸小于 32x32 时，TB 内的边界像素将被滤波后的值替代；

对于 Intra_DC 模式，TB 内的第一行和第一列的像素，使用以原像素和相邻像素作为二阶([3 1]/4)滤波器输入的输出值代替；

对于 Intra_Angular[10]，水平方向，TB 的第一列边界像素，使用相邻参考像素的差值，再加上左上角参考像素除二来代替；

对于 Intra_Angular[26]，垂直方向，TB 的第一行边界像素， 和水平的处理类似；

**6) 参考像素替换**

在 slice 和 tile 的边界，相邻参考像素是无效的；另外，当具有差错快速恢复功能的受限帧内预测 constrained_intra_pred_flag 开启后，为了避免由前面的解码图像造成差错扩散，所有帧间预测 PB 的相邻参考像素也是无效的；在 H.264 中只有 Intra_DC 预测模式能使用这个功能；HEVC 则允许用相邻有效的参考像素值替换无效的参考像素值后再进行其它帧内预测；

**7) 模式编码**

HEVC 对所有块尺寸的亮度预测支持 33 种 Intra_Angular，Intra_Planar，Intra_DC 预测模式；由于方向的增加，HEVC 提供对亮度帧内预测模式的三种最可能模式(MPM)的预测；不同于 H.264 只提供一种最可能模式的预测；在这三个最可能模式中，如果相邻的上方和左方的 PB 都有效且是帧内预测编码模式，则前两个使用上方和左方的 PB 的亮度帧内预测模式来初始化；所有无效的预测模式都被认为是 Intra_DC；为了避免需要存储相邻亮度预测模式的一行像素值，亮度 CTB 之上的 PB 都认为是无效的；

当前两个最可能模式不相等时，则依据 Intra_Planar，Intra_DC，或 Intra_Angular[26](垂直)这三种模式，将第三个最可能预测模式设为在顺序上不和前两种模式重复的模式；

当前两个最可能模式是相等时，且第一个模式的值为 Intra_Planar 或 Intra_DC，则依据 Intra_Planar，Intra_DC，或 Intra_Angular[26](垂直)这三种模式，将第二个和第三个模式依据顺序上的不重复设为这三个模式中的两个；

当前两个最可能模式是相等时，且第一个模式的值为 Intra_Angular，则将第二个和第三个最可能模式设为和第一个模式在方向最近的预测方向值；

如果当前的亮度预测模式是三个最可能模式(MPM)中的一种，则只有 MPM 的索引号被传输给解码器；否则，除了三个最可能模式外，当前亮度预测模式的索引也要使用 5 比特的定长码字传输给解码器；

对于色度帧内预测模式，HEVC 允许其选择下面五种模式：

Intra_Planar，

Intra_Angular[26](垂直)，

Intra_Angular[10](水平)，

Intra_DC，

Intra_Derived，

中的一种，其中 Intra_Derived 是将色度预测模式设定为对应亮度块的帧内方向预测模式；对于这样一种设计，在原则上，HEVC 支持的所有亮度方向预测模式在色度上也同样支持；这样的设计很好地取得了预测精度和信号传输开销间的平衡；被选择的色度预测模式是直接编码的(不使用对预测模式进行预测的机制)；

## ● 帧间预测

**1) 预测块(PB)的划分**

相对于帧内预测 CB，HEVC 对帧间预测 CB 提供了更多的 PB 划分形状；

下面四种模式对应的 CB 划分形状如下:

PART_2N×2N，CB 不划分；

PART_2N×N， CB 水平划分成两个相等尺寸的 PB；

PART_N×2N， CB 垂直划分成两个相等尺寸的 PB；

PART_N×N， CB 划分成四个相等尺寸的 PB，

但是，PART_N×N 模式只有当 CB 尺寸等于最小可允许尺寸 MinCbSizeY（且 log2CbSize>3）时才支持；

另外，还有四种划分类型将 CB 划分成两个不同尺寸的 PB：

PART_2N×nU，

PART_2N×nD，

PART_nL×2N，

PART_nR×2N。

它们被称作**非对称运动分割模式**(AMP)，只有当 log2cbSize>MinCbLog2SizeY 且 amp_enabled_flag==1 时才支持；

**2) 分像素插值**

对于帧间预测编码块(CB)的预测块(PB)像素是从参考图像--以参考图像索引标记--的对应区域得到，这个位置表示为运动矢量的水平和垂直分量；

除了使用整像素 MV 外，对于非整像素位置，分像素插值被用来生成此处的预测像素值。和 H.264 一样，HEVC 也支持四分之一亮度像素的 MV；对于色度像素来说，MV 的精度依据色度像素格式来确定，对于 4:2:0 像素格式，MV 的精度为八分之一像素；

在 HEVC 中，亮度像素的分像素插值应用了两种方法：

对半像素使用**八阶滤波器**；

对四分之一像素使用**七阶滤波器**；

这一点和 H.264 是不一样的；H.264 是用的两步插值处理：

先使用六阶滤波器，舍入均值，在半像素位置生成一个或两个相邻像素的值；

然后在整像素和半像素位置取两个值的平均；

HEVC 对所有分像素位置使用了独立的插值处理，而不用中间的舍入操作，这种方式提高了精度并简化了分像素插值的架构；而且，在 HEVC 中，使用更长的滤波器，如七阶和八阶滤波器来提高插值精度，而不是像在 H.264 中用的六阶滤波器；对四分之一像素位置使用七阶滤波器插值，而不像半像素位置使用八阶滤波器插值，是因为四分之一像素位置更接近整像素位置，因此，在八阶插值中，最远的像素相比半像素情况会更远；在半像素中，相对于整像素的位置是非对称的；实际上，插值滤波器内核的滤波系统部分是从 DCT 基本函数等式中推导出来的；

在图 10 中，标记为大写字母的位置 $A_{i,j}$，表示在整像素位置有效的亮度像素；因此，其它的标记为小写字母的位置表示非整像素位置的像素，它们是需要插值生成的；

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $A_{-1,-1}$ | | | | $A_{0,-1}$ | $a_{0,-1}$ | $b_{0,-1}$ | $c_{0,-1}$ | $A_{1,-1}$ | | | | $A_{2,-1}$ |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| $A_{-1,0}$ | | | | $A_{0,0}$ | $a_{0,0}$ | $b_{0,0}$ | $c_{0,0}$ | $A_{1,0}$ | | | | $A_{2,0}$ |
| $d_{-1,0}$ | | | | $d_{0,0}$ | $e_{0,0}$ | $f_{0,0}$ | $g_{0,0}$ | $d_{1,0}$ | | | | $d_{2,0}$ |
| $h_{-1,0}$ | | | | $h_{0,0}$ | $i_{0,0}$ | $j_{0,0}$ | $k_{0,0}$ | $h_{1,0}$ | | | | $h_{2,0}$ |
| $n_{-1,0}$ | | | | $n_{0,0}$ | $p_{0,0}$ | $q_{0,0}$ | $r_{0,0}$ | $n_{1,0}$ | | | | $n_{2,0}$ |
| $A_{-1,1}$ | | | | $A_{0,1}$ | $a_{0,1}$ | $b_{0,1}$ | $c_{0,1}$ | $A_{1,1}$ | | | | $A_{2,1}$ |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| $A_{-1,2}$ | | | | $A_{0,2}$ | $a_{0,2}$ | $b_{0,2}$ | $c_{0,2}$ | $A_{1,2}$ | | | | $A_{2,2}$ |

图 10. 亮度插值的整像素和分像素位置

$a_{0,0}$，$b_{0,0}$，$c_{0,0}$，$d_{0,0}$，$h_{0,0}$，and $n_{0,0}$ 像素都是对 $A_{i,j}$ 像素，在半像素位置时用八阶滤波器，在四分之一像素位置用七阶滤波器，推导等式如下：

$$a_{0,0} = ( -A_{-3,0} + 4 * A_{-2,0} - 10 * A_{-1,0} + 58 * A_{0,0} + 17 * A_{1,0} - 5 * A_{2,0} + A_{3,0} ) >> shift1 \quad (8\text{-}199)$$

$$b_{0,0} = ( -A_{-3,0} + 4 * A_{-2,0} - 11 * A_{-1,0} + 40 * A_{0,0} + 40 * A_{1,0} - 11 * A_{2,0} + 4 * A_{3,0} - A_{4,0} ) >> shift1 \quad (8\text{-}200)$$

$$c_{0,0} = ( A_{-2,0} - 5 * A_{-1,0} + 17 * A_{0,0} + 58 * A_{1,0} - 10 * A_{2,0} + 4 * A_{3,0} - A_{4,0} ) >> shift1 \quad (8\text{-}201)$$

$$d_{0,0} = ( -A_{0,-3} + 4 * A_{0,-2} - 10 * A_{0,-1} + 58 * A_{0,0} + 17 * A_{0,1} - 5 * A_{0,2} + A_{0,3} ) >> shift1 \quad (8\text{-}202)$$

$$h_{0,0} = ( -A_{0,-3} + 4 * A_{0,-2} - 11 * A_{0,-1} + 40 * A_{0,0} + 40 * A_{0,1} - 11 * A_{0,2} + 4 * A_{0,3} - A_{0,4} ) >> shift1 \quad (8\text{-}203)$$

$$n_{0,0} = ( A_{0,-2} - 5 * A_{0,-1} + 17 * A_{0,0} + 58 * A_{0,1} - 10 * A_{0,2} + 4 * A_{0,3} - A_{0,4} ) >> shift1 \quad (8\text{-}204)$$

像素 $e_{0,0}$，$f_{0,0}$，$g_{0,0}$，$i_{0,0}$，$j_{0,0}$，$k_{0,0}$，$p_{0,0}$，$q_{0,0}$，and $r_{0,0}$的值是对垂直相邻的像素位置 $a_{0,j}$，$b_{0,j}$ and $c_{0,j}$ 使用如下等式得到的：

$$e_{0,0} = ( -a_{0,-3} + 4 * a_{0,-2} - 10 * a_{0,-1} + 58 * a_{0,0} + 17 * a_{0,1} - 5 * a_{0,2} + a_{0,3} ) >> shift2 \quad (8\text{-}205)$$

$$i_{0,0} = ( -a_{0,-3} + 4 * a_{0,-2} - 11 * a_{0,-1} + 40 * a_{0,0} + 40 * a_{0,1} - 11 * a_{0,2} + 4 * a_{0,3} - a_{0,4} ) >> shift2 \quad (8\text{-}206)$$

$$p_{0,0} = ( a_{0,-2} - 5 * a_{0,-1} + 17 * a_{0,0} + 58 * a_{0,1} - 10 * a_{0,2} + 4 * a_{0,3} - a_{0,4} ) >> shift2 \quad (8\text{-}207)$$

$$f_{0,0} = ( -b_{0,-3} + 4 * b_{0,-2} - 10 * b_{0,-1} + 58 * b_{0,0} + 17 * b_{0,1} - 5 * b_{0,2} + b_{0,3} ) >> shift2 \quad (8\text{-}208)$$

$$j_{0,0} = ( -b_{0,-3} + 4 * b_{0,-2} - 11 * b_{0,-1} + 40 * b_{0,0} + 40 * b_{0,1} - 11 * b_{0,2} + 4 * b_{0,3} - b_{0,4} ) >> shift2 \quad (8\text{-}209)$$

$$q_{0,0} = ( b_{0,-2} - 5 * b_{0,-1} + 17 * b_{0,0} + 58 * b_{0,1} - 10 * b_{0,2} + 4 * b_{0,3} - b_{0,4} ) >> shift2 \quad (8\text{-}210)$$

$$g_{0,0} = ( -c_{0,-3} + 4 * c_{0,-2} - 10 * c_{0,-1} + 58 * c_{0,0} + 17 * c_{0,1} - 5 * c_{0,2} + c_{0,3} ) >> shift2 \quad (8\text{-}211)$$

$$k_{0,0} = ( -c_{0,-3} + 4 * c_{0,-2} - 11 * c_{0,-1} + 40 * c_{0,0} + 40 * c_{0,1} - 11 * c_{0,2} + 4 * c_{0,3} - c_{0,4} ) >> shift2 \quad (8\text{-}212)$$

$$r_{0,0} = ( c_{0,-2} - 5 * c_{0,-1} + 17 * c_{0,0} + 58 * c_{0,1} - 10 * c_{0,2} + 4 * c_{0,3} - c_{0,4} ) >> shift2 \quad (8\text{-}213)$$

HEVC 对权重预测编码工具也是可选的；H.264 支持隐含和显示的权重预测；而在 HEVC 中，只能使用显示的权重预测；需要通过对缩放和位移预测值编码在 slice header 中实现；然后，预测的比特深度调整到参考像素原始比特深度；在单向预测中，插值预测值被舍入，右移，并切断到原始比特深度；在双向预测中，从两个 PB 中得到的插值预测值先被相加，然后舍入，右移和切断；

在 H.264 中，需要对每个预测像素(位于四分之一像素位置的像素)进行三步的舍入操作；而如果是双向预测，则在最坏的情况下，需要最多可能到七步的舍入操作；在 HEVC 中，最多需要两步舍入操作来得到每个位于四分之一像素位置的像素；因此，对于双向预测，最多只需要五步的舍入操作；而且，对于最通常的情况，8bits 像素，在最坏情况下整个舍入操作也只需要三步；由于舍入操作步骤的减少，累积的舍入错误会增加，但对于解码器来说，有了更多的灵活性；

对于色度分量的分像素插值处理和亮度分量是相似的；只是在 4：2：0 颜色空间下，分像素的精度为 1/8，并且使用四阶滤波器；

**3) 合并模式**

运动信息通常由水平和垂直运动矢量位移值，一个或两个(对于 B slice，每个参考图像列表都有一个索引)参考图像索引组成；HEVC 允许使用一个合并模式来从空域或时域相邻块来推导运动矢量；所有的运动信息候选者形成了一个合并区域；

合并模式在概念上和 H.264 中的 direct 和 skip 模式相似；然而，这两者有两个很大的不同点:

首先，它是从多个有效候选中选择一个出来作为索引信息传输，这是一种 MV 竞争方案；

其次，它显式地标识了参考图像列表和参考图像索引，而 direct 模式假定这个的值是相同的；

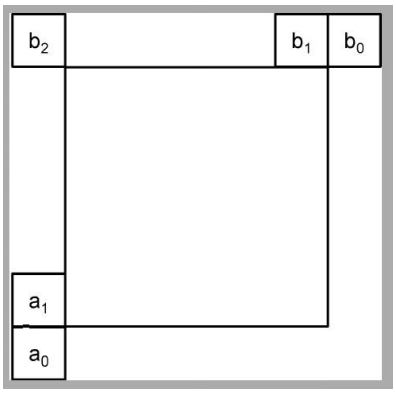



图 11. 运动矢量空域候选者的位置.

合并模式中的可能候选者由空域相邻候选者、时域相邻候选者和生成的候选者组成。图 11 显示了 5 个空域候选者的位置；对于每个候选者的位置，依据{a1，b1，b0，a0，b2}这个顺序来检查有效性；如果此位置是帧内预测块，或是超出了当前 slice 或 tile 边界，就认为它是无效的；

在对空域候选者验证完成后，下面两种类型的冗余被移除：

对于当前 PU，如果候选者的位置是同一个 CU 中的第一个 PU，这个位置的候选者被排除；因为同样的合并可以通过不对预测单元进行划分来实现；有着完全相同运动信息的候选都也要被移除；

对于时域候选者，参考图像对应 PU 外的右下位置，如果有效，则可以用作候选者；否则，使用中心位置来代替；这种选择对应位置 PU 的方法在以前的编码标准中也多有应用；而 HEVC 只通过传输一个索引来说明哪个参考图像列表被用作对应参考图像，这样更灵活；

合并候选者的最大数目 MaxNumMergeCand 在 slice header 中定义；如果发现合并候选者的数目大于 MaxNumMergeCand，则只有前 MaxNumMergeCand -1 个空域候选者和时域候选者有效；否则，如果合并候选者的数目小于 MaxNumMergeCand，需要生成额外的候选者直到数目等于 MaxNumMergeCand；这种方式简化的解析，并且使其更健壮；因为解析编码数据的能力不依赖于合并候选者的有效性；

five_minus_max_num_merge_cand

MaxNumMergeCand = 5 − five_minus_max_num_merge_cand

对于 B slice，额外的合并候选者，依据参考图像列表 0 和列表 1 预定义的顺序，选取两个存在的候选者来得到；例如，第一个生成的候选者使用列表 0 第一个合并候选者；而第二个生成的候选者使用列表 1 第一个合并候选者；HEVC 以下面的顺序定义了 12 个已重建的合并候选者组成预定义对的候选者：

(0， 1)， (1， 0)，

(0， 2)， (2， 0)，

(1， 2)， (2， 1)，

(0， 3)， (3， 0)，

(1， 3)， (3， 1)，

(2， 3)， (3， 2).

在它们中间，在移除冗余后，最多可以用五个候选者；

当 slice 为 P slice 或合并候选者的数量仍小于 MaxNumMergeCand 时，参考索引从 0 到索引数减 1 的相应的零运动矢量被用于填充合并候选者列表；

在 HEVC 中，当所有的编码块标志等于零时，skip 模式被处理成特别的合并模式；在这种特殊情况中，只有 skip 标志和对应的合并索引传输给解码器；H.264 中 B-direct 模式也被合并模式代替，因为合并模式允许所有的运动信息从相邻块空域和时域运动信息中推导得到；

**4) 非合并模式的运动矢量预测**

当帧间预测 CB 块不能被编码成 skip 或合并模式时，MV 就使用运动矢量预测值与 MV 的差来编码；和合并模式类似，HEVC 允许编码器在多个预测值候选中选择 MV 预测值；预测值和实际 MV 间的差值以及候选者的索引被一起传输给解码器；

在图 11 中，依据其有效性，在五个候选中间只有两个空域运动候选者；依据它们的有效性， 第一个空域运动候选者从左位置{a0，a1}集中产生；第二个空域运动候选者从上位置{b0，b1，b2}集中产生，并且是以这个顺序来搜索；

在非合并模式情况下，HEVC 允许用于 MV 预测处理的候选者个数更少，因为编码端可以通过发送编码后的差值来进行运动估计；而且， 因为编码器需要大量计算资源的进行运动估计，更少的候选者能降低计算复杂度；

如果相邻 PU 的参考索引不等于当前的 PU 时，需要对 MV 进行缩放，依据当前图像和由相邻 PU 的参考索引指示的参考图像时域距离，对相邻 MV 进行缩放；当两个空域候选者有相同的 MV 分量时，需要移除一个冗余的空域候选者；

当 MV 预测值个数不等于 2 且时域 MV 预测被有被显示关闭时，就可以使用时域 MV 预测候选者；也可以说，当两个空域候选者都有效时，就不能用时域候选者；最后，零运动矢量可以重复使用直到 MV 的预测候选者为 2，这样就保证了 MV 预测值的个数为 2；因此，在 MV 预测在非合并模式中只需要用一个标志来标识；

## ● 变换与量化

和以前的标准一样，HEVC 对预测残差使用变换编码；依据标准中的规定，可以将残差块划分成多个方形的 TB；支持的变换块尺寸为 4×4，8×8，16×16 和 32×32.

**1) 变换核**

二维变换是通过计算水平和垂直方向的一维变换实现的；核变换矩阵的元素是通过缩放 DCT 基本函数导致得到的，在矩阵系数使用整数值时，需要考虑的是限制变换计算的动态范围，并最大化精度和正交性；

为了简单化，对于 32 个点的长度，只有一个整系数变换矩阵，而子样本用于其它尺寸；

例如，

16 点变换的矩阵；

8 点变换矩阵可以对行 0，2，4，...前 8 个系数推导得到；

4 点变换矩阵可以对行 0，4，8，...前 4 个系数推导得到；

尽管标准在矩阵值是做的变换的简化，但是我们选择的矩阵的系数值有一个对称的特性，这可以减少算术操作并提高因子计算的速度；并且大的变换可以由小的变换组成；

由于支持的变换的尺寸增加，在变换的第一步就对中间值的动态范围作限制是很重要的；HEVC 显示地在第一个 1 维反变换阶段插入一个 7 比特的右移和一个 16 比特的截断操作以确保所有的中间值都在 16 比特范围内；

**2) 可选的 4x4 变换**

对于尺寸为 4x4 的变换块，HEVC 使用了从 DST 推导得到提整数变换，以用于帧内预测模式的亮度残差块，变换矩阵如下：

```
transMatrix =
    {
    {29  55  74  84}
    {74  74   0 -74}
    {84 -29 -74  55}
    {55 -84  74 -29}
    }
```

DST 的基本函数在统计上更适合于残差振幅，并增加了用于预测的边界像素的距离；在复杂度上，4x4 DST 变换和 4x4DCT 变换差不多，但对帧内预测预测减少了约 1%的码率；

DST 变换只用在 4x4 亮度变换块，因为在其它情况下产生不了额外的编码效率改善；

**3) 量化**

因为变换矩阵的行是正交 DCT 变换统一归约数，因此，不需要使用像 H.264 中的预缩放；这可以减少存储开销，特别是变换尺寸为 32x32 时；

HEVC 使用的和 H.264 一样的 URQ 方式来控制量化参数(QP)；QP 的范围为 0-51，并增加到了 6 倍的量化步骤尺寸，以映射算术 QP 值；量化系数矩阵同样支持；

为了减少的在存储开销，标准规定量化系数矩阵只在尺寸为 4x4 和 8x8 时使用；对于更大的 16x16，32x32 变换，使用 8x8 的量化系数矩阵，并且被 2x2，4x4 共享使用；

coefNum = Min(64, (1<<(4+(sizeId<<1))))

## ● 环路滤波

HEVC 指定了两种环路滤波器，即去块滤波器 DBF 和采样点自适应偏移 SAO，均在帧间预测环路中进行。

pps_loop_filter_across_slices_enabled_flag

deblocking_filter_control_present_flag

sample_adaptive_offset_enabled_flag

1) **去块滤波器 DBF**：

HEVC 中的去块滤波器与 H.264 中的 DBF 类似，但是设计更简单，对并行设计支持更好。HEVC 只采用 8*8 结构，而 H.264 采用 4*4 结构。HEVC 的 DBF 滤波强度范围为 0~2 三级，要求先对垂直边缘进行水平滤波，然后再对水平边缘进行垂直滤波。

2) **采样点自适应偏移 SAO**：

采样点自适应偏移 SAO 在编解码环路内，位于 DBF 之后，通过对重建图像的分类，对每一类图像像素值加减一个偏移，达到减少失真的目的，从而提高压缩率，减少码流。若使用 SAO 技术，重构图像将按照递归的方式分裂成 4 个子区域，每个子区域将根据其图像像素特征选择一种像素偏移方式，以减少源图像与重构图像之间的失真。目前采样点自适应偏移方式分为带状偏移(BO)和边缘偏移(EO)两大类。

带状偏移将像素值强度等级划分为若干个条带，每个条带内的像素拥有相同的偏移值。进行偏移时根据重构像素点所处的条带，选择相应的带状偏移值进行偏移。现有的 x265 编码器将像素值强度从 0 到最大值 255，划分为 32 个等级，如图 12 所示。同时这 32 个等级条带还分为两类，第一类是位于中间的 16 个条带，剩余的 16 个条带是第二类。编码时只将其中一类具有较大补偿值的条带偏移信息写入片头；另一类条带信息则不传送。这样的方式编码将具有较小偏移值的一类条带忽略不计，从而节省了编码比特数。

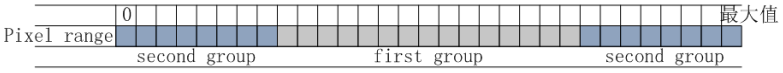



图 12. 32 级像素值条带分割示意图

边缘偏移主要用于对图像的轮廓进行偏移。它将当前像素点值与相邻的 2 个像素值进行对比，用于比较的 2 个相邻像素可以在图 13 中所示的 4 种模板中选择，从而得到该像素点的类型：局部最大、局部最小或者图像边缘。解码器根据码流中标示的像素点的类型信息进行相应的偏移校正。

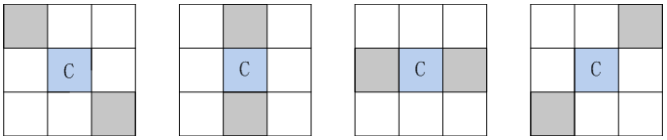



图 13. 4 种边缘样点偏移模板

采用 SAO 后，平均可以减少 2%~6%的码流，而编码器和解码器的性能消耗仅仅增加了约 2%。

- **熵编码**

相比于 H.264，HEVC 只指定使用 CABAC 编码；其核心算法并没有改变；

1) **二值化**

2) **上下文模型**

合适的上下文模型选择对于 CABAC 编码的效率改善是关键因素；在 HEVC 中，除了使用 H.264 中使用空域相邻语法因子外，还使用编码树和变换树的划分深度来推导各种语法元素的上下文模型；

例如，语法元素 cu_skip_flag 指示了当前 CB 是否是帧间预测 skip 模式编码，而语法元素 split_cu_flag 指示了当前 CB 是否要更一步的划分；这些信息都会用作基于空间相邻信息的上下文模型；语法元素 split_transform_flag 指示了当前 TB 是否需要更进一步划分，并且这三个语法元素指明了每个颜色分量的非零变换系数 cbf_luma，cbf_cb，cbf_cr；它们都基于变换树的深度进行编码；尽管 HEVC 中，上下文个数少于 H.264，但它提供了更好的压缩效果；而且，HEVC 使用了更多的 CABAC 操作中的 bypass 模式，这增加了数据吞吐量；

3) **自适应系数扫描**

对于所有的 4x4TB 子块，都要执行系数扫描操作，HEVC 定义了三种系数扫描方法：朝右上的对角线扫描、水平扫描和垂直扫描。

对于在帧内预测区域的 4×4 和 8×8TB 块的变换系数编码来说，这三种扫描方式的选择是隐示的；这个系数扫描的顺序要根据帧内预测方向来选择；

当预测方向接近水平时就使用垂直扫描；

当预测方向接近垂直时就使用水平扫描；

对于其它预测方向，使用右上方向的对角线扫描；

对于所有块尺寸的帧间预测的变换系数，以及 16×16 和 32×32 帧内预测的变换系数，不能对变换系数的子块使用 4x4 的右上方向的对角线扫描；

4) **残差系数编码**

和 H.264 一样，HEVC 最后传输一个非零残差系数的位置、重要性表、符号位及系数值；然而，HEVC 对这些都做了改善，以更好地适应于 TB 尺寸的增加；

## 3. 高层语法

包含类似 H.264 中网络提取层的元素，用于联系视频编码层和网络传输（如数据包封装等），支持的新功能描述如下：

- **随机读取和任意点码流拼接**

H.264 规定**码流**，必须从一个包含关键帧的 IDR 单元开始，它必须不依赖 NAL 中的前置的包就可以独立解码。IDR 是 close GOP 的标志性组成部分。

**IRAP** (Intra Random Access Point)：随机读取点

随机位置读取的支持对频道切换、拖动操作和动态流服务是十分关键的。

HEVC 中新的图像定义：

**CRA** (clean random access) picture，它类似于 open GOP 的起始关键帧。

CRA_NUT

**BLA** (broken link access) pictures，断点连接帧。

BLA_W_LP，可以跟随 RASL 和 RADL pictures

BLA_W_RADL，只跟随 RADL pictures

BLA_N_LP，不会跟随 LPs

**IDR** (Instantaneous Decoding Refresh) picture，close GOP 的起始关键帧。

IDR_W_RADL，可以跟随 RADL pictures

IDR_N_LP，不会跟随 LPs

**RASL** (random access skipped leading) pictures，某些解码顺序在 CRA 帧之后，显示顺序在 CRA 帧之前的帧可能会参考 DPB 中还不存在的帧，于是这些解码器无法解码的帧就只能被丢弃。基于这种情况，这些帧被定义为拖动可跳过的前置帧（RASL）。

RASL_R，

RASL_N，

**RADL** (random access decodable leading) picture，在随机读取点 IRAP 帧后，可以按顺序解码的图像。

RADL_R，

RADL_N，

**LPs** (leading pictures)，前置帧，是 RASL 和 RADL 的总称。

trailing pictures：解码和显示顺序都在随机读取点 IRAP 帧之后的图像，它们不可以将 LPs 作为参考帧。

不同的码流之间切换可以通过断点连接帧（BLA）来拼接。简单的把需要切换的码流的 IRAP 帧标记为 BLA，并放到当前帧的下一个 CRA 帧的地方，然后传输切换码流就可以完成码流拼接的工作。随机读取点 IRAP 帧可以是 IDR、CRA、BLA 图像。CRA 和 BLA 的后面都可能跟随着 RASL 帧（BLA 的 NAL 单元的标记可定）。BLA 帧之后的 RASL 帧解码器必须抛弃，因为它们可能参考了拼接前源码流的帧导致无法解码。

- **支持时域分级扩展**

HEVC 可以在 NAL 单元头中定义一个时域分级扩展的层次。这样只解析到 NAL 层面就可以实现时域可伸缩性。子层接入图像是指在解码过程中，某一子层与其它子层图像没有解码依赖的图像。

在 HEVC 中，设置 SPS 参数 sps_temporal_id_nesting_flag 为 1，即意味着参数 TemporalId 大于 0 的图像属于子层接入图像。其中包含两种类型，即 TSA 和 STSA。

时域子层接入 TSA (temporal sublayer access)；

步进式时域子层接入 STSA (stepwise TSA)，比特流中可以由此开始解码更高时域层的位置。

- **扩展参数集**

作为元数据用以描述编码视频序列的整体特征，包括时域子层之间的依赖性。

- **参考图像集合和参考图像表**

为了管理解码多参考帧，已解码帧将被放在解码图像缓冲区（DPB）中并被详细标记，以供码流后续的解码帧作参考。每个片的头部都会包含一个 POC 以定位那些图像。

RPS 定义为在 DPB 中用作参考帧的图像集合。HEVC 借助 RPS 概念提出了全新的 DPB 管理方法。

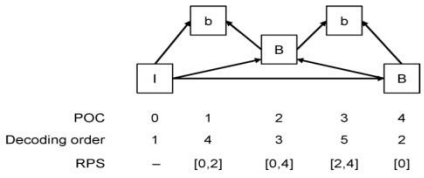



图 15. RPS 示意图

在 H.264 中，当前帧解码完成后，进行参考帧标记和 DPB 操作（从缓存中输出和移出图像）；在 HEVC 中，首先从当前帧的 slice header 中解码出 RPS，并在解码当前帧之前进行参考帧标记和 DPB 操作。

HEVC 中每一个 slice header 都要包含当前帧的 RPS 数据（IDR 条带除外，其 RPS 隐含为空）；除此之外，即使是 I 帧的 slice header 可能也包含 RPS 数据，因为按照解码顺序在它后面的帧也可能会参考它之前的数据；RPS 中图像的数量不能超过 DPB 的大小。

每个图像的 RPS 包含 5 个不同的参考图像表：

| | |
|---|---|
| RefPicSetStCurrBefore | 解码和输出顺序均在当前帧之前的短线参考帧，可用于当前帧的帧间预测； |
| RefPicSetStCurrAfter | 解码顺序在当前帧之前，输出顺序在当前帧之后的短线参考帧，可用于当前帧的帧间预测； |
| RefPicSetStFoll | 按解码顺序在当前帧之后的图像可用的短线参考帧，不能用于当前帧的帧间预测； |
| RefPicSetLtCurr | 可用于当前帧的帧间预测的长线参考帧； |
| RefPicSetLtFoll | 按解码顺序在当前帧之后的图像可用的长线参考帧，不能用于当前帧的帧间预测； |

RPS 根据参考图像类型的不同，分别使用三个循环迭代来传递数据：

POC 比当前帧更低的短线参考帧，

POC 比当前帧更高的短线参考帧，

长线参考帧。

除此之外，还将为每一帧传递一个标识 used_by_curr_pic_flag 来说明是否用作了当前帧的参考帧；

RPS 语法对丢包的兼容性更好，在拖动和其它播放模式下（快进、快退、动态码流切换等）也能更好地工作。这种设计让语法更加明确可展现，避免了解码器对解码过程中的中间状态和临时值的依赖。而且还比 H.264 中的语法更加简化了。

**1．Profile、Level 和 Tier 的概念**

这三个概念主要是为多种不同应用提供兼容性。

Profile 指出码流中使用了哪些编码工具和算法。

Level 指出一些对解码端的负载和内存占用影响较大的关键参数约束。主要包括采样率、分辨率、最大码率，最小压缩率，DPB 容量，CPB（解码缓冲区）大小。

在 HEVC 的设计中，应用可以只依据最大的码率和 CPB 大小就可以区分。为了达成这个效果，有些 Level 定义了两个 Tier——Main Tier 用于大多数应用，High Tier 用于那些最苛刻的应用。

遵守某 Level 和 tier 的解码器可以解码所有等于或低于这个 Level 和 Tier 的码流。

支持某 Profile 的解码器必须支持此 Profile 中的所有特性。

编码器不必实现 Profile 中所有的特性，但产出的码流必须是遵守标准的，比如说要遵守与之兼容的解码器的约束。

**2．HEVC 中的 Profile 和 level**

目前 HEVC 包含 Main， Main10 和 Main Still Picture 三个 profile(详见 spec 文档: T-REC-H.265-201304-I!!PDF-E.pdf 附录 A)。这三个 profile 具有如下限制条件：

* 仅支持 4:2:0 的色度采样；

* WPP 和 tiles 结构可选。若选用了 tiles 结构，则 tile 的亮度分辨率至少要为 256×64。

* **Main Profile** 仅支持 8 位像素（网络视频，移动应用领域）。

* Main 10 Profile 支持 10 位像素（广电，消费电子领域）；10 比特图像将改善图像的质量，并支持超高清电视 UHDTV 采用的 Rec.2020 颜色空间。

* Main Still Picture Profile 允许静态图像按照 main profile 的规定进行编码，但禁止帧间预测（图像领域）。

**Comparison of standards for still image compression based on equal PSNR and MOS**[82]

| Still image coding standard (test method) | Average bit rate reduction compared to | |
|---|---|---|
| | JPEG 2000 | JPEG |
| HEVC (PSNR) | 20.26% | 61.63% |
| HEVC (MOS) | 30.96% | 43.10% |

今后还会有多种附加档次。未来的扩展讨论主要集中在颜色 bit 位深扩展、4:2:2/4:4:4 色度采样视频、多视点编码和可分级编码等方面。

目前，HEVC 定义了 13 个不同的级别，它们的分辨率从 176×144（QCIF）到 7680×4320（8kx4k）。图像的宽和高均需小于等于 8 倍的 MaxLumaPS 再开方。MaxLumaPS 是下图中的最大亮度帧尺寸（避免极端尺寸时解码器产生错误）。

**Table A.1 – General tier and level limits**

| Level | Max luma picture size MaxLumaPs (samples) | Max CPB size MaxCPB (1000 bits) | | Max slice segments per picture MaxSliceSegmentsPerPicture | Max # of tile rows MaxTileRows | Max # of tile columns MaxTileCols |
|---|---|---|---|---|---|---|
| | | Main tier | High tier | | | |
| 1 | 36 864 | 350 | - | 16 | 1 | 1 |
| 2 | 122 880 | 1 500 | - | 16 | 1 | 1 |
| 2.1 | 245 760 | 3 000 | - | 20 | 1 | 1 |
| 3 | 552 960 | 6 000 | - | 30 | 2 | 2 |
| 3.1 | 983 040 | 10 000 | - | 40 | 3 | 3 |
| 4 | 2 228 224 | 12 000 | 30 000 | 75 | 5 | 5 |
| 4.1 | 2 228 224 | 20 000 | 50 000 | 75 | 5 | 5 |
| 5 | 8 912 896 | 25 000 | 100 000 | 200 | 11 | 10 |
| 5.1 | 8 912 896 | 40 000 | 160 000 | 200 | 11 | 10 |
| 5.2 | 8 912 896 | 60 000 | 240 000 | 200 | 11 | 10 |
| 6 | 35 651 584 | 60 000 | 240 000 | 600 | 22 | 20 |
| 6.1 | 35 651 584 | 120 000 | 480 000 | 600 | 22 | 20 |
| 6.2 | 35 651 584 | 240 000 | 800 000 | 600 | 22 | 20 |

**Table A.2 – Tier and level limits for the Main and Main 10 profiles**

| Level | Max luma sample rate MaxLumaSr (samples/sec) | Max bit rate MaxBR (1000 bits/s) | | Min Compression Ratio MinCr |
|---|---|---|---|---|
| | | Main tier | High tier | |
| **1** | 552 960 | 128 | - | 2 |
| **2** | 3 686 400 | 1 500 | - | 2 |
| **2.1** | 7 372 800 | 3 000 | - | 2 |
| **3** | 16 588 800 | 6 000 | - | 2 |
| **3.1** | 33 177 600 | 10 000 | - | 2 |
| **4** | 66 846 720 | 12 000 | 30 000 | 4 |
| **4.1** | 133 693 440 | 20 000 | 50 000 | 4 |
| **5** | 267 386 880 | 25 000 | 100 000 | 6 |
| **5.1** | 534 773 760 | 40 000 | 160 000 | 8 |
| **5.2** | 1 069 547 520 | 60 000 | 240 000 | 8 |
| **6** | 1 069 547 520 | 60 000 | 240 000 | 8 |
| **6.1** | 2 139 095 040 | 120 000 | 480 000 | 8 |
| **6.2** | 4 278 190 080 | 240 000 | 800 000 | 6 |

有 8 个 Level 支持 2 个 Tier（Level4 及以上）。除了 Level1 偏高（要求 350，000b）之外，CPB 容量均等于最大码率的 1 秒容量。当使用 Level 最大的分辨率时，CPB 最大容量为 6 帧图像（包括当前帧、用于参考的帧和准备输出的帧）。如果降低分辨率的话，CPB 可以容纳 16 帧图像（取决与具体采用的分辨率）。

Level 还约束了每帧中垂直和水平方向 tile 的最大数量，以及每秒最大的 tile 数量。

参考文档：

T-REC-H.265-201304-I!!PDF-E.pdf

## X265:

```
Executable Options:
-h/--h                  Show this help text and exit
-V/--version            Show version info and exit
   --cpuid              Limit SIMD capability bitmap 0:auto 1:None. Default:0
   --threads            Number of threads for thread pool (0: detect CPU core count, default)
-p/--preset             ultrafast, veryfast, faster, fast, medium, slow, slower, veryslow, or placebo
-t/--tune               Tune the settings for a particular type of source or situation
-F/--frame-threads      Number of concurrently encoded frames. 0: auto-determined by core count
   --log                Logging level 0:ERROR 1:WARNING 2:INFO 3:DEBUG -1:NONE. Default 2
   --csv                Comma separated log file, log level >= 3 frame log, else one line per run
   --y4m                Parse input stream as YUV4MPEG2 regardless of file extension
   --no-progress        Disable CLI progress reports
-o/--output             Bitstream output file name

Input Options:
   --input              Raw YUV or Y4M input file name
   --input-depth        Bit-depth of input file and internal encoder bit depth. Default 8
   --input-res          Source picture size [w x h], auto-detected if Y4M
   --input-csp          Source color space parameter, auto-detected if Y4M
   --fps                Source frame rate, auto-detected if Y4M
   --frame-skip         Number of frames to skip at start of input file
-f/--frames             Number of frames to be encoded. Default all

Quad-Tree analysis:
   --[no-]wpp           Enable Wavefront Parallel Processing. Default enabled
```

-s/--**ctu** Maximum CU size (default: 64x64). Default 64
--**tu-intra-depth** Max TU recursive depth for intra CUs. Default 1
--**tu-inter-depth** Max TU recursive depth for inter CUs. Default 1

Temporal / motion search options:
--me Motion search method 0:dia 1:hex 2:umh 3:star 4:full. Default 1
-m/--subme Amount of subpel refinement to perform (0:least .. 7:most). Default 2
--merange Motion search range. Default 60
--[no-]**rect** Enable rectangular motion partitions Nx2N and 2NxN. Default enabled
--[no-]**amp** Enable asymmetric motion partitions, requires --rect. Default enabled
--**max-merge** Maximum number of merge candidates. Default 2
--[no-]early-skip Enable early SKIP detection. Default disabled
--[no-]fast-cbf Enable Cbf fast mode. Default disabled

Spatial / intra options:
--rdpenalty penalty for 32x32 intra TU in non-I slices. 0:disabled 1:RD-penalty 2:maximum. Default 0
--[no-]**tskip** Enable intra transform skipping. Default disabled
--[no-]tskip-fast Enable fast intra transform skipping. Default disabled
--[no-]**strong-intra-smoothing** Enable strong intra smoothing for 32x32 blocks. Default enabled
--[no-]**constrained-intra** Constrained intra prediction (use only intra coded reference pixels) Default disabled

Slice decision options:
--**refresh** Intra refresh type - 0:none, 1:CDR, 2:IDR (default: CDR) Default 1
-i/--keyint Max intra period in frames. Default 250
--rc-lookahead Number of frames for frame-type lookahead (determines encoder latency) Default 20
--bframes Maximum number of consecutive b-frames (now it only enables B GOP structure) Default 4
--bframe-bias Bias towards B frame decisions. Default 0
--b-adapt 0 - none, 1 - fast, 2 - full (trellis) adaptive B frame scheduling. Default 2
--[no-]b-pyramid Use B-frames as references. Default enabled
--ref max number of L0 references to be allowed (1 .. 16) Default 3
-w/--[no-]**weightp** Enable weighted prediction in P slices. Default enabled

QP, rate control and rate distortion options:
--bitrate Target bitrate (kbps), implies ABR. Default 0
--crf Quality-based VBR (0-51). Default 28.000000
--vbv-maxrate Max local bitrate (kbit/s). Default 0
--vbv-bufsize Set size of the VBV buffer (kbit). Default 0
--vbv-init Initial VBV buffer occupancy. Default 0.900000
-q/--qp Base QP for CQP mode. Default 32
--aq-mode Mode for Adaptive Quantization - 0:none 1:aqVariance Default 0
--aq-strength Reduces blocking and blurring in flat and textured areas.(0 to 3.0)<double> . Default 1.000000
--cbqpoffs Chroma Cb QP Offset. Default 0
--crqpoffs Chroma Cr QP Offset. Default 0
--rd Level of RD in mode decision 0:least....2:full RDO. Default 0
--[no-]signhide Hide sign bit of one coeff per TU (rdo). Default enabled

Loop filter:
--[no-]**lft** Enable Loop Filter. Default enabled

Sample Adaptive Offset loop filter:
--[no-]**sao** Enable Sample Adaptive Offset. Default enabled
--sao-lcu-bounds 0: right/bottom boundary areas skipped 1: non-deblocked pixels are used. Default 0
--sao-lcu-opt 0: SAO picture-based optimization, 1: SAO LCU-based optimization. Default 1

Quality reporting metrics:
--[no-]ssim Enable reporting SSIM metric scores. Default disabled
--[no-]psnr Enable reporting PSNR metric scores. Default enabled

Reconstructed video options (debugging):

| | |
|---|---|
| -r/--recon | Reconstructed raw image YUV or Y4M output file name |
| --recon-depth | Bit-depth of reconstructed raw image file. Defaults to input bit depth |

SEI options:

| | |
|---|---|
| --hash | Decoded Picture Hash SEI 0: disabled, 1: MD5, 2: CRC, 3: Checksum. Default 0 |