


## 【H.264/AVC视频编解码技术详解】十二、解析H.264码流的宏块结构（上）\_Workshop of Wenjie.Yin-CSDN博客

---

 [blog.csdn.net/shagoneal/article/details/53471976](https://blog.csdn.net/shagoneal/article/details/53471976)

《H.264/AVC视频编解码技术详解》视频教程已经在“CSDN学院”上线，视频中详述了H.264的背景、标准协议和实现，并通过一个实战工程的形式对H.264的标准进行解析和实现，欢迎观看！

---

“纸上得来终觉浅，绝知此事要躬行”，只有自己按照标准文档以代码的形式操作一遍，才能对视频压缩编码标准的思想和方法有足够深刻的理解和体会！

---

链接地址：[H.264/AVC视频编解码技术详解](#)

---

GitHub代码地址：[点击这里](#)

---

### H.264中Slice Body——宏块结构(Macroblock)的解析

---

#### 1.Slice Data结构的定义

---

在已经实现了一个slice的header部分之后，下面的工作将是研究如何解析一个slice的主体，即Slice Body部分。一个Slice的body部分主要是一个个的宏块结构Macroblock组成，此外还存在一些辅助的信息。标准文档中规定的slice\_data()结构如下图：

slice_data( ) {	C	Descriptor
if( entropy_coding_mode_flag )		
while( !byte_aligned( ) )		
<b>cabac_alignment_one_bit</b>	2	f(1)
CurrMbAddr = first_mb_in_slice * ( 1 + MbaffFrameFlag )		
moreDataFlag = 1		
prevMbSkipped = 0		
do {		
if( slice_type != I && slice_type != SI )		
if( !entropy_coding_mode_flag ) {		
<b>mb_skip_run</b>	2	ue(v)
prevMbSkipped = ( mb_skip_run > 0 )		
for( i=0; i<mb_skip_run; i++ )		
CurrMbAddr = NextMbAddress( CurrMbAddr )		
if( mb_skip_run > 0 )		
moreDataFlag = more_rbsp_data( )		
} else {		
<b>mb_skip_flag</b>	2	ae(v)
moreDataFlag = !mb_skip_flag		
}		
if( moreDataFlag ) {		
if( MbaffFrameFlag && ( CurrMbAddr % 2 == 0    ( CurrMbAddr % 2 == 1 && prevMbSkipped ) ) )		
<b>mb_field_decoding_flag</b>	2	u(1)   ae(v)
macroblock_layer( )	2   3   4	
}		
if( !entropy_coding_mode_flag )		
moreDataFlag = more_rbsp_data( )		
else {		
if( slice_type != I && slice_type != SI )		
prevMbSkipped = mb_skip_flag		
if( MbaffFrameFlag && CurrMbAddr % 2 == 0 )		
moreDataFlag = 1		
else {		
<b>end_of_slice_flag</b>	2	ae(v)
moreDataFlag = !end_of_slice_flag		
}		
}		
CurrMbAddr = NextMbAddress( CurrMbAddr )		
} while( moreDataFlag )		
}		

从文档中我们可以看出，Slice\_data结构中独立的语法元素并不多，主要只有以下几个：

1. **cabac\_alignment\_one\_bit**：表示如果码流启用了CABAC算法，那么码流在这里必须使用若干个比特1实现字节对齐。
2. **mb\_skip\_run**和**mb\_skip\_flag**：这两个语法元素都用于表示宏块结构是否可以被跳过。“跳过”的宏块指的是，在帧间预测的slice中，当图像区域平坦时，码流中跳过这个宏块的所有数据，不进行传输，只通过这两个语法元素进行标记。在解码端，跳过的宏块通过周围已经重建的宏块来进行恢复。mb\_skip\_run用于熵编码使用CAVLC时，用一个语法元素表示连续跳过的宏块的个数；mb\_skip\_flag用于熵编码使用CABAC时，表示每一个宏块是否被跳过。

3. **mb\_field\_decoding\_flag**：标识位，用于在帧场自适应的码流中标识某个宏块是帧模式还是场模式。
4. **end\_of\_slice\_flag**：在CABAC模式下的一个标识位，表示是否到了slice的末尾。

上述的几个语法元素毫无疑问仅仅占用了全部数据很少的一部分，其他大部分的数据都包含在宏块结构中，即上表中的\*\*macroblock\_layer()\*\*结构。

## 2. 宏块(Macroblock)结构

从上表中我们可以看出，一个Slice结构中宏块实际上占据了绝大部分。在标准中一个宏块的结构定义为下表：

macroblock_layer() {	C	Descriptor
<b>mb_type</b>	2	ue(v)   ae(v)
if( mb_type == I_PCM ) {		
while( !byte_aligned() )		
<b>pcm_alignment_zero_bit</b>	3	f(1)
for( i = 0; i < 256; i++ )		
<b>pcm_sample_luma[ i ]</b>	3	u(v)
for( i = 0; i < 2 * MbWidthC * MbHeightC; i++ )		
<b>pcm_sample_chroma[ i ]</b>	3	u(v)
} else {		
noSubMbPartSizeLessThan8x8Flag = 1		
if( mb_type != I_NxN && MbPartPredMode( mb_type, 0 ) != Intra_16x16 && NumMbPart( mb_type ) == 4 ) {		
sub_mb_pred( mb_type )	2	
for( mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++ )		
if( sub_mb_type[ mbPartIdx ] != B_Direct_8x8 ) {		
if( NumSubMbPart( sub_mb_type[ mbPartIdx ] ) > 1 )		
noSubMbPartSizeLessThan8x8Flag = 0		
} else if( !direct_8x8_inference_flag )		
noSubMbPartSizeLessThan8x8Flag = 0		
} else {		
if( transform_8x8_mode_flag && mb_type == I_NxN )		
<b>transform_size_8x8_flag</b>	2	u(1)   ae(v)
mb_pred( mb_type )	2	
}		
if( MbPartPredMode( mb_type, 0 ) != Intra_16x16 ) {		
<b>coded_block_pattern</b>	2	me(v)   ae(v)
if( CodedBlockPatternLuma > 0 && transform_8x8_mode_flag && mb_type != I_NxN && noSubMbPartSizeLessThan8x8Flag && ( mb_type != B_Direct_16x16    direct_8x8_inference_flag ) )		
<b>transform_size_8x8_flag</b>	2	u(1)   ae(v)
}		
if( CodedBlockPatternLuma > 0    CodedBlockPatternChroma > 0    MbPartPredMode( mb_type, 0 ) == Intra_16x16 ) {		
<b>mb_qp_delta</b>	2	se(v)   ae(v)
residual( 0, 15 )	3   4	
}		
}		
}		

<https://blog.csdn.net/shaqoneal>

## (1). mb\_type:

在一个宏块中，最开始的语法元素为宏块的类型：**mb\_type**。从表中我们可以看出，根据**mb\_type**的值是否等于I\_PCM，整个解析方法分为两大类：PCM类型和非PCM类型，判断依据是当**mb\_type**为25时为I\_PCM模式，否则为非I\_PCM模式。

当这个宏块为I\_PCM模式时，宏块中以差分编码的形式保存宏块原始的像素值。此时存在如下几个语法元素：

- **pcm\_alignment\_zero\_bit**：填充位，用比特0来填充直到按字节对齐；
- **pcm\_sample\_luma**：256个亮度分量的差分像素值；
- **pcm\_sample\_chroma**：若干个色度分量的差分像素值，实际数量由码流的颜色格式指定。例如对于最常用的4:2:0格式的视频，共有128个色度像素值。

除了**mb\_type**等于25时可以确定为I\_PCM格式之外，其他的**mb\_type**值可能根据帧类型（或slice类型）的不同而不同。比如对于I slice，**mb\_type**的非PCM模式可以选择0~24这些值之一；对于P slice，**mb\_type**只能取0~4这5个值；对于B slice，**mb\_type**可以取0~22这些值之一。目前我们所处理的码流全部由I帧构成，因此我们暂时只考虑I slice的情况。下图是标准文档中规定的I slice的**mb\_type**列表的一部分，完整列表在协议文档的表7-11中：

Table 7-11 – Macroblock types for I slices

mb_type	Name of mb_type	transform_size_8x8_flag	MbPartPredMode (mb_type, 0)	Intra16x16PredMode	CodedBlockPatternChroma	CodedBlockPatternLuma
0	I_NxN	0	Intra_4x4	na	Equation 7-35	Equation 7-35
0	I_NxN	1	Intra_8x8	na	Equation 7-35	Equation 7-35
1	I_16x16_0_0_0	na	Intra_16x16	0	0	0
2	I_16x16_1_0_0	na	Intra_16x16	1	0	0
3	I_16x16_2_0_0	na	Intra_16x16	2	0	0
4	I_16x16_3_0_0	na	Intra_16x16	3	0	0
5	I_16x16_0_1_0	na	Intra_16x16	0	1	0
6	I_16x16_1_1_0	na	Intra_16x16	1	1	0
7	I_16x16_2_1_0	na	Intra_16x16	2	1	0
8	I_16x16_3_1_0	na	Intra_16x16	3	1	0
9	I_16x16_0_2_0	na	Intra_16x16	0	2	0
10	I_16x16_1_2_0	na	Intra_16x16	1	2	0
11	I_16x16_2_2_0	na	Intra_16x16	2	2	0
12	I_16x16_3_2_0	na	Intra_16x16	3	2	0
13	I_16x16_0_0_1	na	Intra_16x16	0	0	15
14	I_16x16_1_0_1	na	Intra_16x16	1	0	15
15	I_16x16_2_0_1	na	Intra_16x16	2	0	15
16	I_16x16_3_0_1	na	Intra_16x16	3	0	15

从上表中我们可以看出，mb\_type不仅仅表示了宏块的分割方式，还包含了一些其他的附加信息，如帧内预测模式、亮度和色度分量的coded\_block\_pattern。当帧内预测使用16×16模式时，宏块整个宏块的预测信息相同，因此不需要为各个子宏块分别指定预测模式，这样可以有效减少消耗的码流。

## (2). transform\_size\_8x8\_flag

该语法元素为一个标识位，用于表示在环路滤波之前，预测残差的变换系数解码时依照的尺寸。当该标识位为1时，预测残差按照8×8像素块进行解码；当该标志位不存在或者为0时，预测残差按照4×4像素块进行解码。

## (3). coded\_block\_pattern



coded\_block\_pattern语法元素常简称做cbp，用于表示当前宏块内的4个8×8子块编码对其中的哪个的残差系数进行编码。值得注意的是该语法元素仅仅在宏块为非I\_16x16模式时才存在，因为在I\_16x16模式时cbp的有关信息已经在mb\_type中体现。

#### (4). mb\_qp\_delta

mb\_qp\_delta表示宏块层的量化参数偏移值，取值范围为[-26, 25]。我们在前面已经在PPS中获取了整个序列的量化参数初始值（由pic\_init\_qp\_minus26计算），在slice header中获取slice层的量化参数偏移slice\_qp\_delta，因此每一个slice第一个宏块的量化参数可通过下面的公式计算：

$QP_0 = \text{pic\_init\_qp} + 26 + \text{slice\_qp\_delta} + \text{mb\_qp\_delta}$

$QP_0 = \text{pic\_init\_qp} + 26 + \text{slice\_qp\_delta} +$

$\text{mb\_qp\_delta}$

从第二个宏块开始，每个宏块实际量化参数的计算方法为：

$QP_n = (QP_m + \text{mb\_qp\_delta} + 52) \% 52$

$QP_n = (QP_m + \text{mb\_qp\_delta} + 52) \% 52$