讲到H.264除了前两篇文章提到的，I、P、B帧。**参看：** **图像和流媒体 -- I 帧,B帧,P帧,IDR帧的区别**

还有其他相关术语：

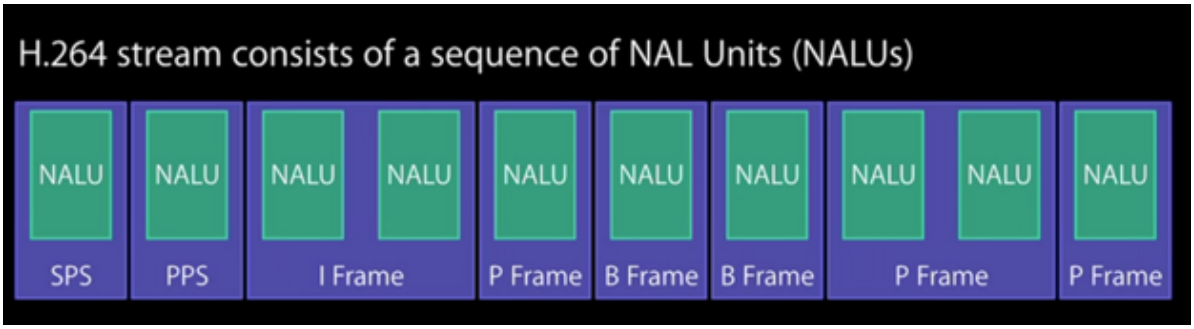
NALU：H264编码数据存储或传输的基本单元，一般H264码流最开始的两个NALU是SPS和PPS，第三个NALU是IDR。SPS、PPS、SEI这三种NALU不属于帧的范畴。
SPS(Sequence Parameter Sets)：序列参数集，作用于一系列连续的编码图像。
PPS(Picture Parameter Set)：图像参数集，作用于编码视频序列中一个或多个独立的图像。
SEI(Supplemental enhancement information)：附加增强信息，包含了视频画面定时等信息，一般放在主编码图像数据之前，在某些应用中，它可以被省略掉。
IDR(Instantaneous Decoding Refresh)：即时解码刷新。
HRD(Hypothetical Reference Decoder)：假想码流调度器。

上面这些知识我还是了解的。但还是思考了半晌，不知道从哪讲起？它们之间的关系又该怎么讲？

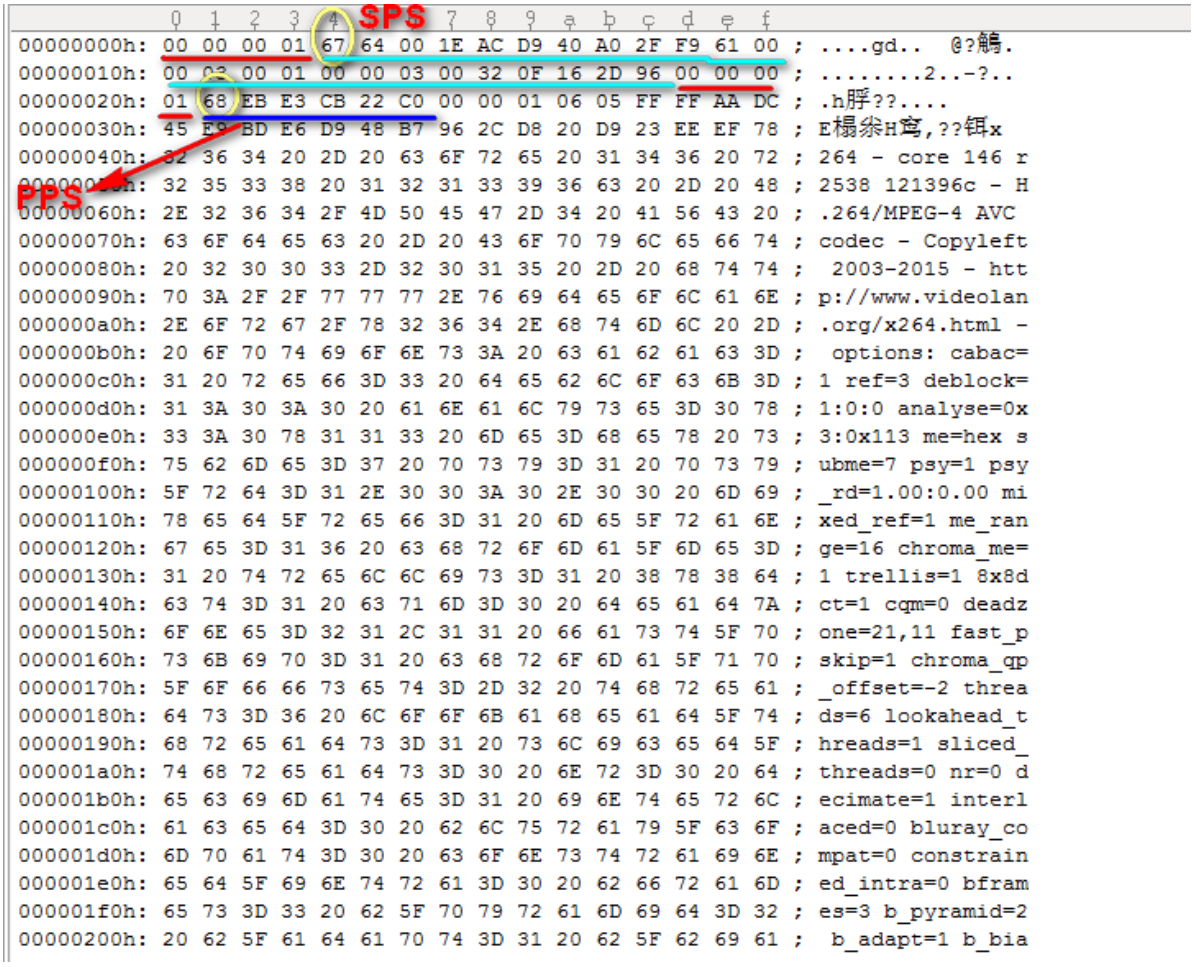
想了解更多内容，

一、H.264 NALU语法结构

在H.264/AVC视频编码标准中，整个系统框架被分为了两个层面：视频编码层面（VCL）和网络抽象层面（NAL）。其中，前者负责有效表示视频数据的内容，**而后者则负责格式化数据并提供头信息，以保证数据适合各种信道和存储介质上的传输。**因此我们平时的每帧数据就是一个NAL单元（SPS与PPS除外）。在实际的H264数据帧中，往往帧前面带有00 00 00 01 或 00 00 01分隔符，一般来说编码器编出的首帧数据为PPS与SPS，接着为I帧……



使用 UltraEdit 查看一个 h.264 文件信息



其中 SPS、PPS 文章开始也讲了。

SPS(Sequence Parameter Sets)：序列参数集，作用于一系列连续的编码图像。
PPS(Picture Parameter Set)：图像参数集，作用于编码视频序列中一个或多个独立的图像。

如上图，在H264码流中，都是以"0x00 0x00 0x01"或者"0x00 0x00 0x00 0x01"为开始码的，找到开始码之后，使用开始码之后的第一个字节的低 5 位判断是否为 7(sps)或者 8(pps), 及 data[4] & 0x1f == 7 || data[4] & 0x1f == 8。然后对获取的 nal 去掉开始码之后进行 base64 编码，得到的信息就可以用于 sdp。 sps和pps需要用逗号分隔开来。

上图中，00 00 00 01是一个nalu的起始标志。后面的第一个字节，0x67，是nalu的类型，type &0x1f==0x7表示这个nalu是sps，type &0x1f==0x8表示是pps。

接下来我们来讲解一下NALU语法结构：

H264基本码流由一些列的NALU组成。原始的NALU单元组成：

[start code] + [NALU header] + [NALU payload]；

start code	1字节	00 00 01 或 00 00 00 01	需要添加的
NALU header	1字节	如下3	通过mdat定位

H264基本码流结构分两层：视频编码层VCL和网络适配层NAL，这样使信号处理和网路传输分离

VCL	负责高效视频内容表示
NAL	以网络所要求的恰当方式对数据进行打包和发送

H.264码流在网络中传输时实际是以NALU的形式进行传输的.

每个`NALU`由**一个字节**的`Header`和`RBSP`组成.

NAL Header 的组成为：

forbidden_zero_bit(1bit) + nal_ref_idc(2bit) + nal_unit_type(5bit)

1、forbidden_zero_bit：

禁止位，初始为0，当网络发现NAL单元有比特错误时可设置该比特为1，以便接收方纠错或丢掉该单元。

2、nal_ref_idc：

nal重要性指示，标志该NAL单元的重要性，值越大，越重要，解码器在解码处理不过来的时候，可以丢掉重要性为0的NALU。

3、nal_unit_type：NALU类型取值如下表所示：

句法表中的 C 字段表示该句法元素的分类，这是为片区服务。

nal_unit_type	**NAL类型**	C
0	未使用	
1	非IDR图像中不采用数据划分的片段	2,3,4
2	非IDR图像中A类数据划分片段	2
3	非IDR图像中B类数据划分片段	3
4	非IDR图像中C类数据划分片段	4
5	IDR图像的片	2,3
6	补充增强信息单元（SEI）	5
7	序列参数集	0
8	图像参数集	1
9	分界符	6
10	序列结束	7
11	码流结束	8
12	填充	9
13..23	保留	
24..31	不保留（RTP打包时会用到）	

不过上面这张图，我实在没有找到出处啊。但是我在 x264 里看到了这个。

```
65 /****************************************************************************
66  * NAL structure and functions
67  ****************************************************************************/
68
69 enum nal_unit_type_e
70 {
71     NAL_UNKNOWN     = 0,
72     NAL_SLICE       = 1,
   工作区 SLICE_DPA   = 2,
   NAL_SLICE_DPB   = 3,
75     NAL_SLICE_DPC   = 4,
76     NAL_SLICE_IDR   = 5,    /* ref_idc != 0 */
77     NAL_SEI         = 6,    /* ref_idc == 0 */
78     NAL_SPS         = 7,
79     NAL_PPS         = 8,
80     NAL_AUD         = 9,
81     NAL_FILLER      = 12,
82     /* ref_idc == 0 for 6,9,10,11,12 */
83 };
"x264.h" 962 行 --5%--
```

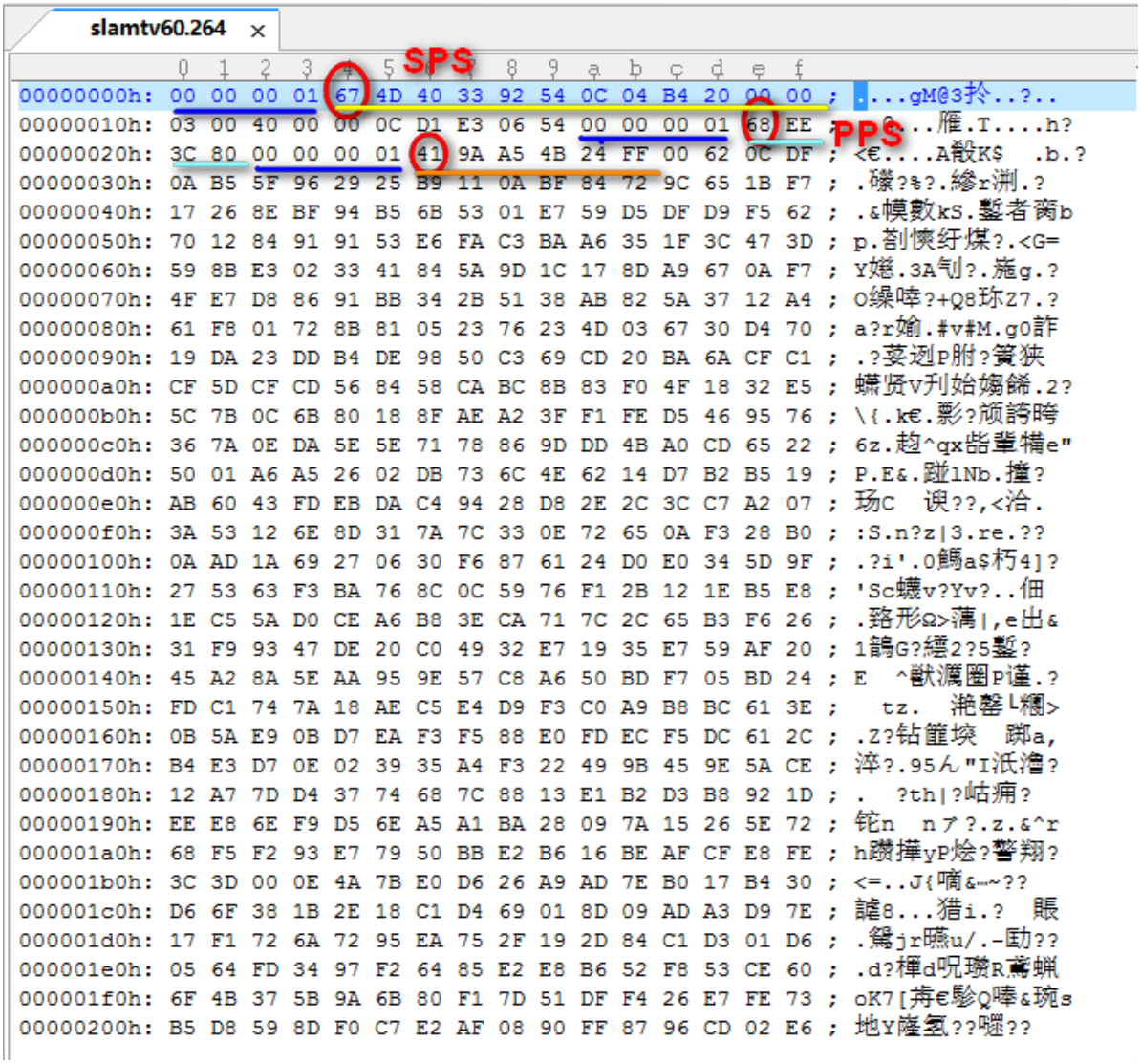
其中需要关注的是 SEI、SPS、PPS。我在 LIVE555 里又看到这个。

```
55   unsigned numSPropRecords;
56   SPropRecord* sPropRecords = parseSPropParameterSets(sPropParameterSetsStr, numSPropRecords);
57   for (unsigned i = 0; i < numSPropRecords; ++i) {
58     if (sPropRecords[i].sPropLength == 0) continue; // bad data
59     u_int8_t nal_unit_type = (sPropRecords[i].sPropBytes[0])&0x1F;
60     if (nal_unit_type == 7/*SPS*/) {
61       sps = sPropRecords[i].sPropBytes;
62       spsSize = sPropRecords[i].sPropLength;
63     } else if (nal_unit_type == 8/*PPS*/) {
64       pps = sPropRecords[i].sPropBytes;
65       ppsSize = sPropRecords[i].sPropLength;
66     }
67   }
"liveMedia/H264VideoRTPSink.cpp" [只读] 131 行 --51%--
```

这不就是上面我们讲到的，nalu的类型 type &0x1f==0x7表示这个nalu是sps，type &0x1f==0x8表示是pps。

接下来我们来举个例子，来讲解下：

该视频下载：[H.264 示例视频和工具]



00 00 00 01 为起始符，67 即 nal_unit_type。

0x67的二进制是 0110 0111

则 forbidden_zero_bit(1bit) = 0；

nal_ref_idc(2bit) = 3；

nal_unit_type(5bit) = 7；即 SPS 类型。

然后看另一部分 RBSP

这里提一下 SODB 和 RBSP 关系

SODB(String Of Data Bits)：最原始的编码数据RBSP, 长度不一定是8的倍数，此时需要对齐.
RBSP： 在SODB的后面填加了结尾比特（RBSP trailing bits 一个bit“1”）若干比特“0”,以便字节对齐。

RBSP

我们知道码流是由一个个的NAL Unit组成的，NALU是由NALU头和RBSP数据组成，而RBSP可能是SPS，PPS，Slice或SEI，目前我们这里SEI不会出现，而且SPS位于第一个NALU，PPS位于第二个NALU，其他就是Slice(严谨点区分的话可以把IDR等等再分出来)了。

而上面这个h.264文件，相当于包含两个 NALU吧，第一个是SPS，第二个是PPS。

我们先看第一个NALU（SPS）的 RBSP (10个字节)

67 4D 40 33 92 54 0C 04 B4 20

转换成二进制：

0110 0111
0100 1101
0100 0000
0011 0011
1001 0010
0101 0100
0000 1100
0000 0100
1011 0100
0010 0000

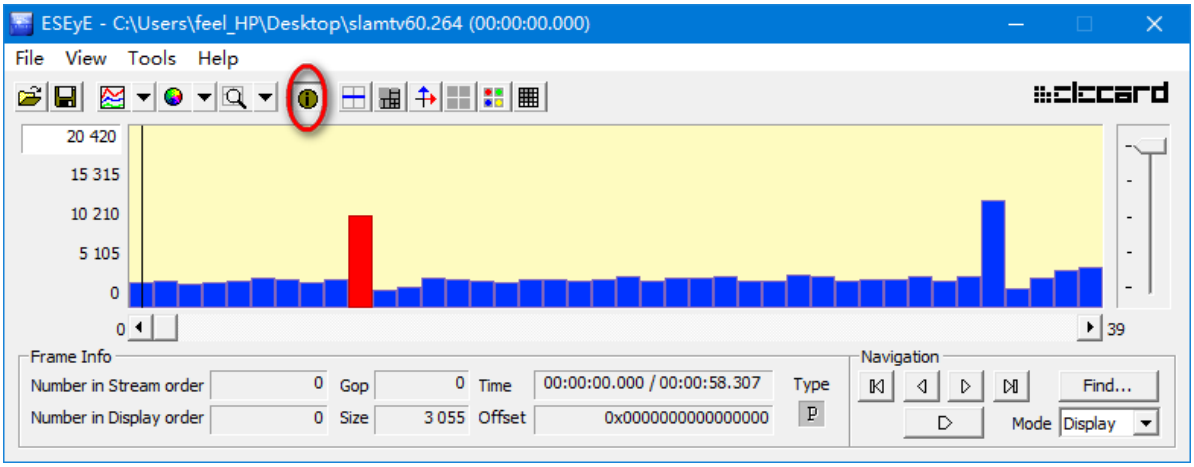
先看NALU头，解析结果如下：

forbidden_zero_bit = 0 // 0 u(1)
nal_ref_idc = 3 // 11 u(2)
nal_unit_type = 7 // 00111 u(5)
这就对了，看看 NAL_SPS = 7;

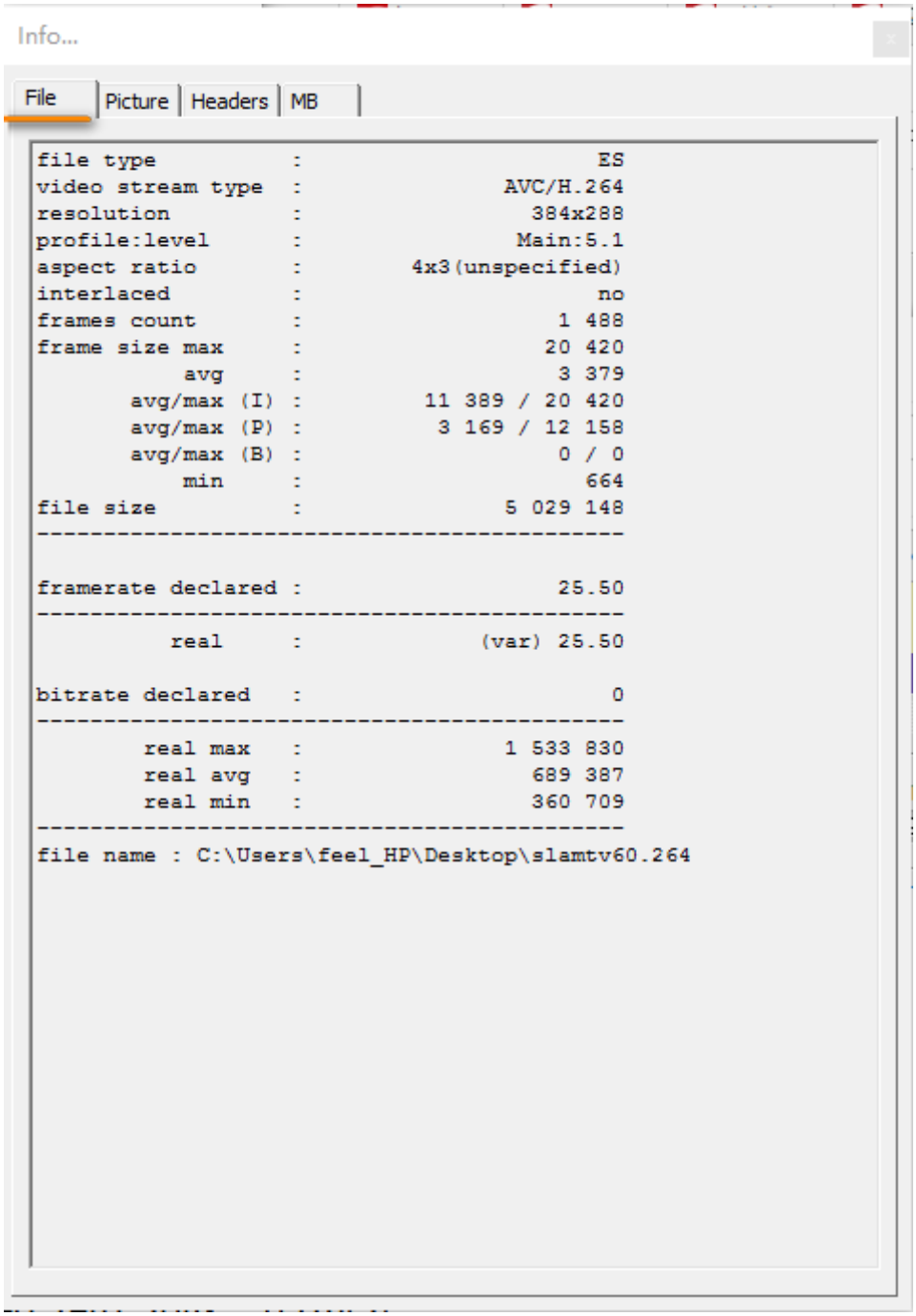
接下来进入 RBSP，先讲SPS的

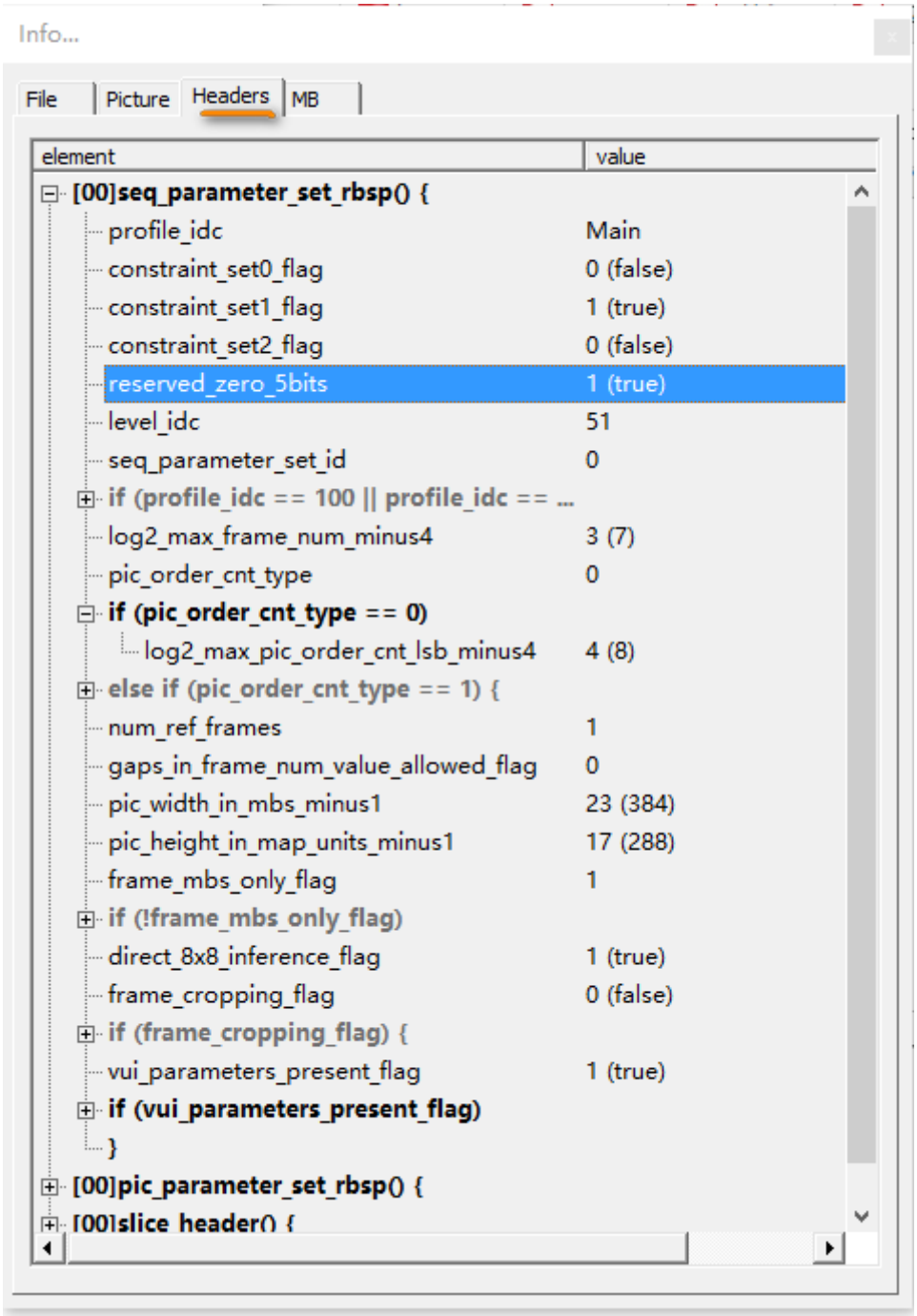
还记得视频编码数据工具 Elecard Stream Eye

参看：FFmpeg再学习 -- 视音频基础知识

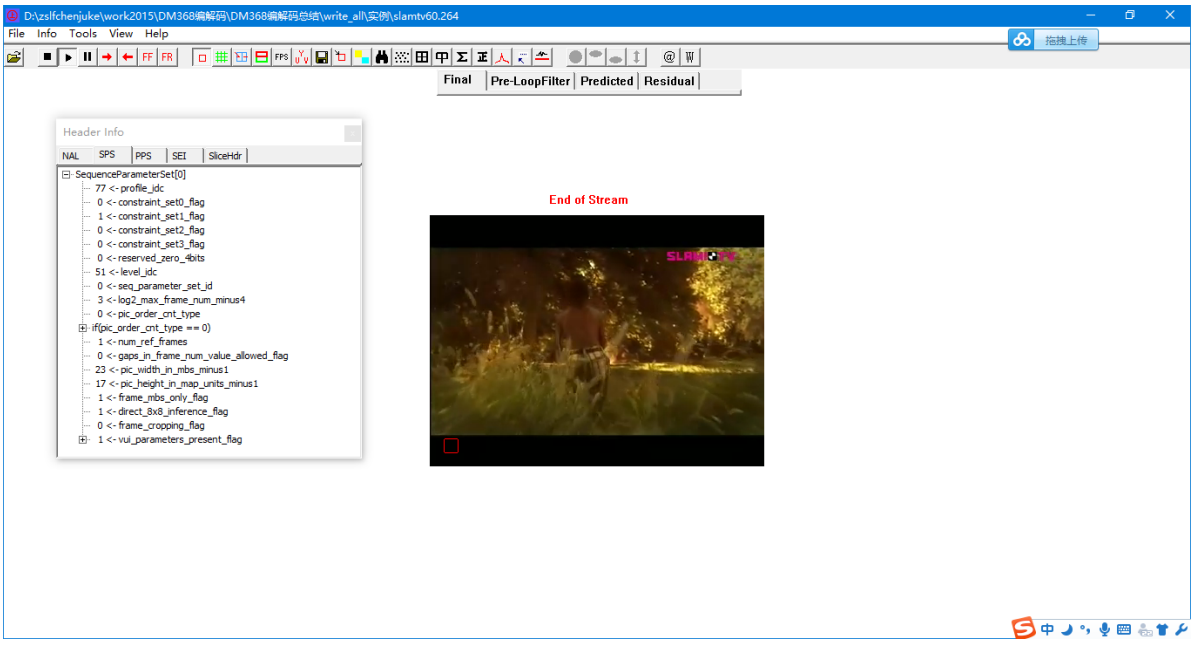


点击 show\hide info 可查看 File 和 Headers （重点）





这里在推荐一款软件 **下载：** [视频分析工具H264Visa]



我们就是根据上图里的内容来进行分析。

profile_idc = 77 // 0100 1101 u(8) Main **参看：H264 各profiles用途和特点**

constraint_set0_flag = 0 //0 u(1)

constraint_set1_flag = 1 //1 u(1)

constraint_set1_flag = 0 //0 u(1)

reserved_zero_5bits = 1 //1 u(5)

level_idc = 51 //0011 0011 u(8)

==========

对于 seq_parameter_set_id，我们看到它是ue(v)，这是一种指数哥伦布编码

扩展：

参看：指数哥伦布码

指数哥伦布编码是一种在编码技术中经常用到的编码，其是无损编码，在HEVC中以及之前的编码技术H.264/AVC中，由于其可以由编码直接解得码字的变长码，所以广受欢迎。HM源码中的SPS/PPS和每个片的头部分都是用哥伦布编码进行编码。
对于一个需要编码的数 x，按照以下的几步进行编码：
\1. 按照二进制形式写下 x+1，
\2. 根据写下的数字，计算出当前数值的位数，然后在该数的前面加上当前数值位数减一后得到的数值个数的零。

例如：编码“3”
\1. 该数加一后（即4）的二进制为100，
\2. 当前数值的位数是三位，3减去1后得到2，所以在“100”的前方加上两个零，得“00100”即为3的哥伦布码。

下面列出1-8的哥伦布码：
0=> 1=> 1
1=> 10=> 010
2=> 11=> 011
3=> 100=> 00100
4=> 101=> 00101
5=> 110=> 00110
6=> 111=> 00111
7=> 1000=> 0001000
8=> 1001=> 0001001

哥伦布码扩展到负数范围
每一个负数进行编码的时候，将其映射到其绝对值的两倍。即-4映射为8进行编码；正数的映射为其两倍减一进行编码，即4映射为7进行编码。
例如：
0 => 0 => 1 => 1
1 => 1 => 10 => 010
=>1 => 2 => 11 => 011
2 => 3 => 100 => 00100

=>2 => 4 => 101 => 00101
3 => 5 => 110 => 00110
=>3 => 6 => 111 => 00111
4 => 7 => 1000 => 0001000
=>4 => 8 => 1001 => 0001001

K阶指数哥伦布码
为了用更少的比特表示更大的数值，可以使用多阶指数哥伦布编码（代价是相比起之前的0阶哥伦布码来书，小的数值可能需要更多的比特去表示）
进行K阶哥伦布编码的步骤是
\1. 确定进行编码的阶数K
\2. 将原数映射到” X + (2^k) -1” （即如果在3阶条件下编码4，则其将被映射到4+2^3-1=11）
\3. 将上一步骤得到的数值进行0阶编码得到0阶哥伦布码（11->0001100）
\4. 去掉码的前部分k个前导零（0001100->1100）
在进行解码的时候，从bit stream中寻找第一个非零比特值，然后把之前遇到的零的个数存在leadingzerobit参数中，即可根据该参数去被编码值了。

上面讲到的只是 ue(v)，但是还有其他的像是 se(v)，又是什么?

参看：H264的句法和语义(二)

H264定义了如下几种描述子：

ae(v)	**基于上下文自适应的二进制算术熵编码；**
b(8)	读进连续的8个比特；
ce(v)	基于上下文自适应的可变长熵编码；
f(n)	读进连续的n个比特；
i(n)/i(v)	读进连续的若干比特，并把他们解释为有符号整数；
me(v)	映射指数Golomb熵编码；
se(v)	有符号指数Golomb熵编码；
te(v)	截断指数Golomb熵编码；
u(n)/u(v)	读进连续的若干比特，并将它们解释为无符号整数；
ue(v)	无符号指数Golomb熵编码。

我们看到，描述子都在括号中带有一个参数，这个参数表示需要提取的比特数。

当参数是n时，表明调用这个描述子的时候回指明n的值，也即该句法元素是定长编码。

当参数是v时，对应的句法元素是变成编码，这时有两种情况：

i(v)和u(v)两个描述子的v由以前的句法元素指定，也就是说在前面会有句法元素指定当前句法元素的比特长度；陈列这两个描述子外，其他描述子都是熵编码，他们的解码算术本身能够确定当前句法元素的比特长度。

==========

seq_parameter_set_id = 0 // 1 ue(v)

log2_max_frame_num_minus4 = 3 //00100 ue(v)

pic_order_cnt_type = 0 //1 ue(v)

log2_max_pic_order_cnt_lsb_minus4 = 4 //00101 ue(v)

num_ref_frames = 1//010 ue(v)

gaps_in_frame_num_value_allowed_flag = 0 //0

pic_width_in_mbs_minus1 = 23 // 000011000 ue(v) (23+1)*16 = 384

pic_height_in_map_units_minus1 = 17 //000010010 ue(v) (17+1)*16 = 288

frame_mbs_only_flag = 1 //1

direct_8x8_inference_flag = 1 //1

frame_cropping_flag = 0 //0

vui_parameters_present_flag = 1 //1

以上分析部分和视频分析工具 header info SPS 比较发现结果是一致的。

SPS部分讲完了，然后再看 PPS

00 00 00 01 68 EE 3C 80

首先起始符 00 00 00 01

然后 68 即 nal_unit_type。

0x68的二进制是 0110 1000

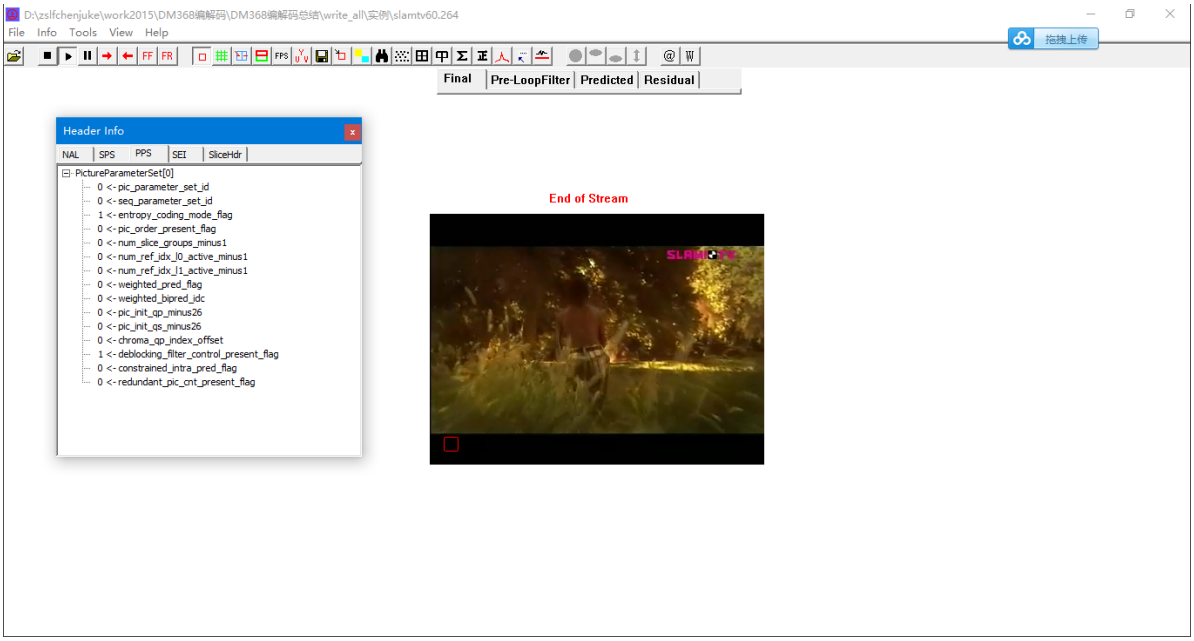
则 forbidden_zero_bit(1bit) = 0；

nal_ref_idc(2bit) = 3；

nal_unit_type(5bit) = 8；即 PPS 类型。

然后再看 RBSP 部分

用视频分析工具得出的结果如下图：



将 EE 3C 80

转换成二进制：

1110 1110
0011 1100
1000 0000

pic_parameter_set_id = 0 //1 ue(v)
seq_parameter_set_id = 0 //1 ue(v)
entropy_coding_mode_flag = 1 //1
pic_order_present_flag = 0 //0
num_slice_groups_minus1 = 0 //1 ue(v)
num_ref_idx_l0_default_active_minus1 = 0 //1 ue(v)
num_ref_idx_l1_default_active_minus1 = 0 //1 ue(v)
weighted_pred_flag = 0 //0
weighted_bipred_idc = 0 //00
pic_init_qp_minus26 = 0 //1 ue(v)
pic_init_qs_minus26 = 0 //1 ue(v)
chroma_qp_index_offset = 0 //1 ue(v)
deblocking_filter_control_present_flag = 1 //1
constrained_intra_pred_flag = 0 // 0
redundant_pic_cnt_present_flag = 0 // 0

以上分析部分和视频分析工具 header info PPS 比较发现结果是一致的。

上面部分 **参看：** [一步一步解析H.264码流的NALU(SPS,PSS,IDR)](#) 同理可能还有 slice 部分的分析，这里不做介绍了

想深入了解的

参看：H.264学习笔记之一（层次结构，NAL，SPS）

参看：H.264学习笔记之二（片及片头语法）

参看：H.264句法和语法总结 系列

记得一定要看哦，讲的是真好。还是贴出两张 NAL 句法，不然上面的讲的内容没一点解释，以后再看会有点懵。

句法	C	Desc
nal_nuit(NumBytesInNALunit){/* NumBytesInNALunit为统计出来的数据长度 */		
forbidden_zero_bit /* 等于0 */	All	f(1)
nal_ref_idc/* 当前NAL的优先级，取值范围0-3 */	All	u(2)
nal_unit_type /* NAL类型，见表2描述 */	All	u(5)
NumBytesInRBSP=0		
for(i=1;i<NumBytesInNALunit;i++){		
if(i+2<NumBytesInNALunit && next_bits(24)==0x000003{		
/* 0x000003伪起始码，需要删除0x03这个字节 */		
rbsp_byte[NumBytesInRBSP++]	All	b(8)
rbsp_byte[NumBytesInRBSP++]	All	b(8)
i+=2/* 取出前两个0x00后，跳过0x03 */		
emulation_prevention_three_byte/* equal to 0x03 */	All	f(8)
}else{		
rbsp_byte[NumBytesInRBSP++] /* 继续读取后面的字节 */	All	b(8)
}		
}		

表3

2.2序列参数集（SPS）

句法	C	Desc
seq_parameter_set_rbsp(){		
profile_idc/* 指明所用的Profile */	0	u(8)
constraint_set0_flag	0	u(1)
constraint_set1_flag	0	u(1)
constraint_set1_flag	0	u(1)
reserved_zero_5bits /* equal to 0 */	0	u(5)
level_idc /* 指明所用的Level */	0	u(8)
seq_parameter_set_id /* 指明本序列参数集的id号，0-31，被图像集引用，编码需要产生新的序列集时，使用新的id，而不是改变原来参数集的内容 */	0	ue(v)
log2_max_frame_num_minus4/* 为读取元素frame_num服务，frame_num标识图像的解码顺序，frame_num的解码函数是ue(v)，其中v=log2_max_frame_num_minus4+4，该元素同时指明frame_num的最大值MaxFrameNum=2(log2_max_frame_num_minus4+4)*/	0	ue(v)
pic_order_cnt_type /* 指明poc的编码方法，poc标识图像的播放顺序，poc	0	ue(v)

可以由frame_num计算，也可以显示传送。poc共三种计算方式 */		
if(pic_order_cnt_type==0)		
log2_max_pic_order_cnt_lsb_minus4 /* 指明变量MaxPicOrderCntLsb的值， MaxPicOrderCntLsb = 2(log2_max_pic_order_cnt_lsb_minus4+4) */	0	ue(v)
else if(pic_order_cnt_type==1){		
delta_pic_order_always_zero_flag /* 等于1时，元素delta_pic_order_cnt[0]和delta_pic_order_cnt[1]不在片头中出现，并且它们的默认值是0，等于0时，上述两元素出现的片头中 */	0	u(1)
offset_for_non_ref_pic /* 用来计算非参考帧或场的poc，$[-2^{31},2^{31}-1]$ */	0	se(v)
offset_for_top_to_bottom_field/* 计算帧的底场的poc */	0	se(v)
num_ref_frames_inpic_order_cnt_cycle /* 用来解码poc,[0.255] */	0	ue(v)
for(i=0;i<num_ref_frames_inpic_order_cnt_cycle;i++)		
offset_for_ref_frame[i]/* 用来解码poc，对于循环中的每个元素指定一个偏移 */	0	se(v)
}		
num_ref_frames /* 参考帧队列可达到的最大长度，[0,16] */	0	ue(v)
gaps_in_frame_num_value_allowed_flag /* 为1，允许slice header中的frame_num不连续 */	0	u(1)
pic_width_inmbs_minus1 /* 本元素加1，指明以宏块为单位的图像宽度 PicWidthInMbs=pic_width_in_mbs_minus1+1 */	0	ue(v)
pic_height_in_map_units_minus1 /* 本元素加1，指明以宏块为单位的图像高宽度 PicHeightInMapUnitsMbs=pic_height_in_map_units_minus1+1 */	0	ue(v)
frame_mbs_only_flag /* 等于0表示本序列中所有图像均为帧编码；等于1，表示可能是帧，也可能场或帧场自适应，具体编码方式由其它元素决定。结合前一元素：FrameHeightInMbs=(2-frame_mbs_only_flag)*PicHeightInMapUnits */	0	ue(v)
if(frame_mbs_only_flag)		
mb_adaptiv_frame_field_flag /* 指明本序列是否是帧场自适应模式： frame_mbs_only_flag=1，全部是帧 frame_mbs_only_flag=0， mb_adaptiv_frame_field_flag=0，帧场共存 frame_mbs_only_flag=0， mb_adaptiv_frame_field_flag=1，帧场自适应和场共存*/	0	u(1)
direct_8x8_inference_flag /* 用于指明B片的直接和skip模式下的运动矢量的计算方式 */	0	u(1)
frame_cropping_flag /* 解码器是否要将图像裁剪后输出，如果是，后面为裁剪的左右上下的宽度 */	0	u(1)
if(frame_cropping_flag){		
frame_crop_left_offset	0	ue(1)
frame_crop_right_offset	0	ue(1)
frame_crop_top_offset	0	ue(1)
frame_crop_bottom_offset	0	ue(1)
}		

}		
vui_parameters_present_flag /* 指明vui子结构是否出现在码流中，vui子结构在附录中指明，用于表征视频格式的信息 */	0	u(1)
if(vui_parameters_present_flag)		
vui_parameters()	0	
rbsp_trailing_bits()	0	
}		

表4

到此，NALU语法结构大致讲完了。