视频解码 FFmpeg HEVC/H.265

关注者 10 被浏览 9,147

怎么解析HEVC/H265解码参数HEVCDecoderConfigurationRecord?

如何从HEVC的VPS,SPS,PPS中获取HEVCDecoderConfigurationRecord （ISO-IEC-14496-15）参数集？...显示全部

关注问题 写回答 邀请回答 好问题 添加评论 分享 ...

2 个回答 默认排序

天涯来客 + 关注

HEVCDecoderConfigurationRecord的定义参考:

ffmpeg.org/doxygen/trun ...

具体解析部分参考: ffmpeg.org/doxygen/trun ...

发布于 2018-03-04 19:15

▲ 赞同 ▼ 添加评论 分享 收藏 喜欢

后端开发老鸽 + 关注

H265出现的原因：

我们视频的分辨率 从 720p到1080p,而且电视的屏幕也越来越大

视频帧率从30帧 到60帧，再到120帧

这就会导致我们cpu在编解码的时候，会出现宏块个数爆发式增长，运动矢量α复杂度增加的后果，直接导致我们编码后的视频文件依旧很大，所以H264编码方式α已经不满足现在的需求了，H265也就应运而生。

下面是通过Elecard HEVCAnalyzer 软件分析α宏块的效果，可以看到我们的宏块大小在颜色差异大的地方越小，这样我们的画面在界面明显差异的地方就会越清晰。

- H265和H264编码数据比较：
- 编码后H265 I帧数据大小 比H264 I帧数据大
- 编码H265 P帧、B帧数据大小 比 H264P帧、B帧数据小
  所以H265的编码数据小的主要原因在于 P帧、B帧数据很小。

下图是类型的占用字节数α。（前面依旧是四个字节的分隔符）

7.3.1.2 NAL unit header syntax

nal_unit_header( ) {	Descriptor
forbidden_zero_bit	f(1)
nal_unit_type	u(6)
nuh_layer_id	u(6)
nuh_temporal_id_plus1	u(3)

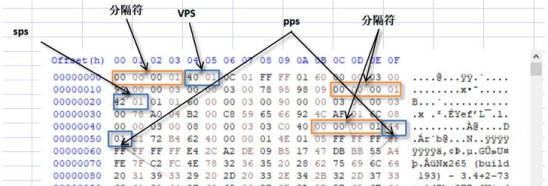
H265-NALU-Type

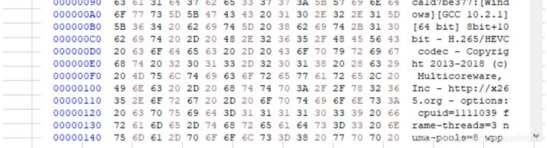
接下来看一下H265的NALU类型的种类。如下图。

Table 7-1 – NAL unit type codes and NAL unit type classes

nal_unit_type	Name of nal_unit_type	Content of NAL unit and RBSP syntax structure	NAL unit type class
0 1	TRAIL_N TRAIL_R	Coded slice segment of a non-TSA, non-STSA trailing picture slice_segment_layer_rbsp( )	VCL
2 3	TSA_N TSA_R	Coded slice segment of a TSA picture slice_segment_layer_rbsp( )	VCL
4 5	STSA_N STSA_R	Coded slice segment of an STSA picture slice_segment_layer_rbsp( )	VCL
6 7	RADL_N RADL_R	Coded slice segment of a RADL picture slice_segment_layer_rbsp( )	VCL
8 9	RASL_N RASL_R	Coded slice segment of a RASL picture slice_segment_layer_rbsp( )	VCL
10 12 14	RSV_VCL_N10 RSV_VCL_N12 RSV_VCL_N14	Reserved non-IRAP SLNR VCL NAL unit types	VCL
11 13 15	RSV_VCL_R11 RSV_VCL_R13 RSV_VCL_R15	Reserved non-IRAP sub-layer reference VCL NAL unit types	VCL
16 17 18	BLA_W_LP BLA_W_RADL BLA_N_LP	Coded slice segment of a BLA picture slice_segment_layer_rbsp( )	VCL
19 20	IDR_W_RADL IDR_N_LP	Coded slice segment of an IDR picture slice_segment_layer_rbsp( )	VCL
21	CRA_NUT	Coded slice segment of a CRA picture slice_segment_layer_rbsp( )	VCL
22 23	RSV_IRAP_VCL22 RSV_IRAP_VCL23	Reserved IRAP VCL NAL unit types	VCL
24..31	RSV_VCL24.. RSV_VCL31	Reserved non-IRAP VCL NAL unit types	VCL
32	VPS_NUT	Video parameter set video_parameter_set_rbsp( )	non-VCL
33	SPS_NUT	Sequence parameter set seq_parameter_set_rbsp( )	non-VCL
34	PPS_NUT	Picture parameter set pic_parameter_set_rbsp( )	non-VCL
35	AUD_NUT	Access unit delimiter access_unit_delimiter_rbsp( )	non-VCL
36	EOS_NUT	End of sequence end_of_seq_rbsp( )	non-VCL
37	EOB_NUT	End of bitstream end_of_bitstream_rbsp( )	non-VCL
38	FD_NUT	Filler data filler_data_rbsp( )	non-VCL
39 40	PREFIX_SEI_NUT SUFFIX_SEI_NUT	Supplemental enhancement information sei_rbsp( )	non-VCL
41..47	RSV_NVCL41.. RSV_NVCL47	Reserved	non-VCL
48..63	UNSPEC48.. UNSPEC63	Unspecified	non-VCL

接下来直接来分析H265编码数据中的 NALU_TYPE;





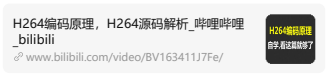
来举个例子，解释怎么计算的吧。

```
40 01
//根据上面的NAL_unit_header_type
16进制:01000000 000 00001
//第二位包括第二位往后6位都是nal_Type,所以计算40 里的 6位 就行了
01000000 & 0x7E << 1
计算结果:32
//查表得知
这个帧类型是 vps
```

上面是热身，接下来就进入实战，我们来解析sps里的高度和宽度。

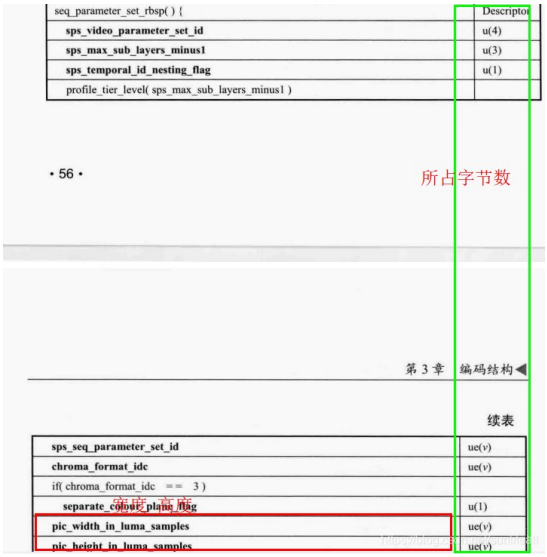
首先我们必须知道那两个字段是高度和宽度，这就要查找官方文档了。

首先我们需要看几个字段的含义，不然看文档的时候会一脸懵



名称	含义
ae(v)	基于上下文自适应的二元算术编码
b(8)	读进连续的 8 bit
f(n)	读进连续的n bit
se(v)	有符号指数 Golomb 熵编码
u(n)	读进连续的 n bit，且它们解码后的值为无符号整数
e(n)	无符号指数 Golomb熵编码

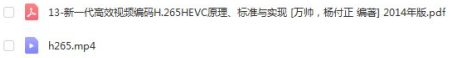
好的我们了解了这几个字符之后，就可以看懂点文档了，贴sps编码Q格式



从图中我们可以很容易的发现，宽度和高度在sps编码格式的所处位置和占用字节数，我们只需要按照这张表格的规范，逐个解析即可， 但是. 我们需要注意. 当我们解析的时候遇到 `00 00 03`的数值时，我们需要跳过`03`,接下来，看一个表格(sps模拟数据读取宽高表格分解图Q)

1	2	3	4	5	6	7	8	9
42	01	01	01	60	00	00	03	00
0100 0010	0000 0001	0000 0001	0000 0001	0110 0000	0000 0000	0000 0000	0000 0011	0000 0000
					general_level_idc			
10	11	12	13	14	15	16	17	18
90	00	00	03	00	00	03	00	78
1001 0000	0000 0000	0000 0000	0000 0011	0000 0000	0000 0000	0000 0011	0000 0000	0111 1000
19	20	21	22	23	24	25	26	27
A0	04	B2	00	C8	59	65	66	92
1010 0000	0000 0100	1011 001	0000 0000	1100 1000	01 1001	0110 0101	0110 0110	1001 0010
4C								
0100 1100								

上图就是我截取的sps数据(一段真实的视频编码Q)，图中灰色的区域，我们解析的时候需要跳过这个字段，不去解析，这个是转义符Q的意思。红色字体就是我们的宽度(pic_width_in_luma_samples)，而橘黄色字体就是我们的高度(pic_height_in_luma_samples),图中其他不同颜色的是解析的时候比较混淆的地方，我标出来，可以更好的区分。



解析代码如下:

```
#include <iostream>
#include <memory.h>
#include <bh.h>
typedef signed char int8;

typedef unsigned char   uint8;

typedef unsigned short uint16;

typedef unsigned long uint32;
typedef signed char int8;
typedef signed short int16;
typedef signed long int32;
struct vc_params_t
{
    long width,height;
    int8 profile, level;
    int8 nal_length_size;
    void clear()
    {
        memset(this, 0, sizeof(*this));
    }
};

class NALBitstream
{
public:
    int m_idx;
    NALBitstream() : m_data(NULL), m_len(0), m_idx(0), m_bits(0), m_byte(0), m_z
    {

    };
    NALBitstream(void * data, int len)
    {
        Init(data, len);
    };
    void Init(void * data, int len)
    {
        m_data = (LPBYTE)data;
        m_len = len;
        m_idx = 0;
        m_bits = 0;
        m_byte = 0;
        m_zeros = 0;
    };
    BYTE GetBYTE()
    {
        if ( m_idx >= m_len )
            return 0;
        BYTE b = m_data[m_idx++];
        if ( b == 0 )
        {
            m_zeros++;
            if ( (m_idx < m_len) && (m_zeros == 2) && (m_data[m_idx] == 0x03) )
            {
                m_idx++;
                m_zeros=0;
            }
        }
        else  m_zeros = 0;

        return b;
    };

    UINT32 GetBit()
    {
        if (m_bits == 0)
        {
            m_byte = GetBYTE();
            m_bits = 8;
        }
        m_bits--;
        return (m_byte >> m_bits) & 0x1;
    };

    UINT32 GetWord(int bits)
    {
        UINT32 u = 0;
        while ( bits > 0 )
        {
            u <<= 1;
            u |= GetBit();
            bits--;
        }
        printf("end index is :%d\n",m_idx);
        return u;
    };

    //返回哥伦布编码对应的实际值
    UINT32 GetUE()
    {
        int zeros = 0;
        while (m_idx < m_len && GetBit() == 0 ) zeros++;
        UINT32 value=GetWord(zeros) + ((1 << zeros) - 1);
        return value;
    };

    INT32 GetSE()
    {
        UINT32 UE = GetUE();
        bool positive = UE & 1;
        INT32 SE = (UE + 1) >> 1;
        if ( !positive )
```

```
        {
            SE = -SE;
        }
        return SE;
    };
private:
    LPBYTE m_data;
    int m_len;
    int m_bits;
    BYTE m_byte;
    int m_zeros;
};

bool  ParseSequenceParameterSet(BYTE* data,int size, vc_params_t& params)
{
    if (size < 20)
    {
        return false;
    }
    NALBitstream bs(data, size);
    // seq_parameter_set_rbsp()
    int  test =bs.GetWord(4);// sps_video_parameter_set_id
    int sps_max_sub_layers_minus1 = bs.GetWord(3);
    if (sps_max_sub_layers_minus1 > 6)
    {
        return false;
    }
    test =bs.GetWord(1);
    {
        test =bs.GetWord(2);
        test =bs.GetWord(1);
        test = params.profile = bs.GetWord(5);
        test = bs.GetWord(32);//6
        test =bs.GetWord(1);//
        test =bs.GetWord(1);//
        test = bs.GetWord(1);//
        test =bs.GetWord(1);//
        test = bs.GetWord(44);//
        test= params.level   = bs.GetWord(8);// general_level_idc
        uint8 sub_layer_profile_present_flag[6] = {0};
        uint8 sub_layer_level_present_flag[6]   = {0};
        for (int i = 0; i < sps_max_sub_layers_minus1; i++) {
            sub_layer_profile_present_flag[i]= bs.GetWord(1);
            sub_layer_level_present_flag[i]= bs.GetWord(1);
        }
        if (sps_max_sub_layers_minus1 > 0)
        {
            for (int i = sps_max_sub_layers_minus1; i < 8; i++) {
                uint8 reserved_zero_2bits = bs.GetWord(2);
                printf("------");
            }
        }
        for (int i = 0; i < sps_max_sub_layers_minus1; i++)
        {
            if (sub_layer_profile_present_flag[i]) {
                test =bs.GetWord(2);
                test = bs.GetWord(1);
                test = bs.GetWord(5);
                test =bs.GetWord(32);
                test = bs.GetWord(1);
                test = bs.GetWord(1);
                test = bs.GetWord(1);
                test = bs.GetWord(1);
                test = bs.GetWord(44);
            }
            if (sub_layer_level_present_flag[i]) {
                test = bs.GetWord(8);// sub_layer_level_idc[i]
            }
        }
    }
    uint32 sps_seq_parameter_set_id= bs.GetUE();
    if (sps_seq_parameter_set_id > 15) {
        return false;
    }
    uint32 chroma_format_idc = bs.GetUE();
    if (sps_seq_parameter_set_id > 3) {
        return false;
    }
    if (chroma_format_idc == 3) {
        test =  bs.GetWord(1);//
    }

    params.width  = bs.GetUE(); // pic_width_in_luma_samples
    params.height  = bs.GetUE(); // pic_height_in_luma_samples
    if (bs.GetWord(1)) {
        bs.GetUE();
        bs.GetUE();
        bs.GetUE();
        bs.GetUE();
    }
    uint32 bit_depth_luma_minus8= bs.GetUE();
    uint32 bit_depth_chroma_minus8= bs.GetUE();
    if (bit_depth_luma_minus8 != bit_depth_chroma_minus8) {
        return false;
    }
    return true;
}
int main() {
    vc_params_t params = {0};
    BYTE Sps[41] = {0X42 ,0X01 ,0X01 ,0X01 ,0X60 ,0X00 ,0X00 ,0X03 ,0X00 ,0X90 ,(
                    0X00 ,0X03 ,0X00 ,0X00 ,0X03 ,0X00 ,0X78,0XA0 ,0X04 ,0XB2 ,0
                    0XC8 ,0X59 ,0X65 ,0X66 ,0X92 ,0X4C ,0XAF ,0X01 ,0X6C ,0X08 ,(
                    0X00 ,0X03 ,0X00 ,0X08 ,0X00};
    ParseSequenceParameterSet(Sps,41,params);
    printf("%d-%d-%d\n",params.width,params.height,params.level);
    system("pause");
    return 0;
}
```

发布于 2022-05-23 20:23

写回答