FFmpeg的H.264解码器源代码简单分析:解码器主干部分

2015年04月14日 16:04:50 阅读数:15858

H.264源代码分析文章列表:

【编码 - x264】

x264源代码简单分析:概述

x264源代码简单分析:x264命令行工具(x264.exe)

x264源代码简单分析:编码器主干部分-1

x264源代码简单分析:编码器主干部分-2

x264源代码简单分析:x264_slice_write()

x264源代码简单分析:滤波(Filter)部分

x264源代码简单分析:宏块分析(Analysis)部分-帧内宏块(Intra)

x264源代码简单分析:宏块分析(Analysis)部分-帧间宏块(Inter)

x264源代码简单分析:宏块编码(Encode)部分

x264源代码简单分析:熵编码(Entropy Encoding)部分

FFmpeg与libx264接口源代码简单分析

【解码 - libavcodec H.264 解码器】

FFmpeg的H.264解码器源代码简单分析:概述

FFmpeg的H.264解码器源代码简单分析:解析器(Parser)部分

FFmpeg的H.264解码器源代码简单分析:解码器主干部分

FFmpeg的H.264解码器源代码简单分析:熵解码(EntropyDecoding)部分

FFmpeg的H.264解码器源代码简单分析:宏块解码(Decode)部分-帧内宏块(Intra)

FFmpeg的H.264解码器源代码简单分析:宏块解码(Decode)部分-帧间宏块(Inter)

FFmpeg的H.264解码器源代码简单分析:环路滤波(Loop Filter)部分

本文分析FFmpeg的H.264解码器的主干部分。"主干部分"是相对于"熵解码"、"宏块解码"、"环路滤波"这些细节部分而言的。它包含了H.264解码器 直到decode_slice()前面的函数调用关系(decode_slice()后面就是H.264解码器的细节部分,主要包含了"熵解码"、"宏块解码"、"环路滤波"3个部分)。

函数调用关系图

解码器主干部分的源代码在整个H.264解码器中的位置如下图所示。

单击查看更清晰的图片

解码器主干部分的源代码的调用关系如下图所示。

单击查看更清晰的图片

从图中可以看出,H.264解码器(Decoder)在初始化的时候调用了ff_h264_decode_init(),ff_h264_decode_init()又调用了下面几个函数进行解码器汇编函数的初始化工作(仅举了几个例子):

ff_h264dsp_init():初始化DSP相关的汇编函数。包含了IDCT、环路滤波函数等。

```
ff h264qpel init():初始化四分之一像素运动补偿相关的汇编函数。
```

ff_h264_pred_init():初始化帧内预测相关的汇编函数。

H.264解码器在关闭的时候调用了h264_decode_end(),h264_decode_end()又调用了ff_h264_remove_all_refs(),ff_h264_free_context()等几个函数进行清理工作。H.264解码器像帧的时候调用了h264_decode_frame(),h264_decode_frame()调用了decode_nal_units(),decode_nal_units()调用了两类函数——解析函数和解码函数,如下所示。

```
(1) 解析函数(获取信息):
```

```
ff_h264_decode_nal():解析NALU Header。
ff_h264_decode_seq_parameter_set():解析SPS。
ff_h264_decode_picture_parameter_set():解析PPS。
ff_h264_decode_sei():解析SEI。
ff_h264_decode_slice_header():解析Slice Header。
```

(2) 解码函数 (解码获得图像):

ff_h264_execute_decode_slices():解码Slice。

其中ff_h264_execute_decode_slices()调用了decode_slice(),而decode_slice()中调用了解码器中细节处理的函数(暂不详细分析):

ff_h264_decode_mb_cabac():CABAC熵解码函数。 ff_h264_decode_mb_cavlc():CAVLC熵解码函数。 ff_h264_hl_decode_mb():宏块解码函数。 loop_filter():环路滤波函数。

本文针对H.264解码器decode_slice()前面的函数调用关系进行分析。

ff_h264_decoder

ff_h264_decoder是FFmpeg的H.264解码器对应的AVCodec结构体。它的定义位于libavcodec\h264.c,如下所示。

```
[cpp] 📳 📑
     AVCodec ff h264 decoder = {
 1.
2.
         .name
                               = NULL_IF_CONFIG_SMALL("H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10"),
3.
          .long name
                           = AVMEDIA_TYPE_VIDEO,
      .type
4.
5.
          .id
                               = AV CODEC ID H264.
        .priv_data_size = sizeof(H264Context),
6.
7.
         .init
                               = ff h264 decode init,
     .close
                             = h264_decode_end,
8.
9.
          .decode
                                = h264_decode_frame
10.
         .capabilities
                             = /*CODEC_CAP_DRAW_HORIZ_BAND |*/ CODEC_CAP_DR1 |
11.
                                 CODEC CAP DELAY | CODEC CAP SLICE THREADS |
12.
                               CODEC_CAP_FRAME_THREADS,
13.
          .flush
                               = flush_dpb,
         .init_thread_copy = ONLY_IF_THREADS_ENABLED(decode_init_thread_copy),
14.
          .update_thread_context = ONLY_IF_THREADS_ENABLED(ff_h264_update_thread_context),
15.
16.
         .profiles
                        = NULL_IF_CONFIG_SMALL(profiles),
17.
                               = &h264_class,
          .priv class
     };
18.
```

从ff_h264_decoder的定义可以看出:解码器初始化的函数指针init()指向ff_h264_decode_init()函数,解码的函数指针decode()指向h264_decode_frame()函数,解码器 关闭的函数指针close()指向h264_decode_end()函数。

ff_h264_decode_init()

ff_h264_decode_init()用于FFmpeg H.264解码器的初始化。该函数的定义位于libavcodec\h264.c,如下所示。

```
[cpp] 📳 🛅
1.
     //H.264解码器初始化函数
2.
     av cold int ff h264 decode init(AVCodecContext *avctx)
3.
4.
         H264Context *h = avctx->priv data;
5.
          int i:
     int ret;
6.
8.
     h->avctx = avctx;
9.
          //8颜色位深8bit
10.
     h->bit depth luma
11.
          //1代表是YUV420P
     h->chroma_format_idc = 1;
12.
13.
14.
         h->avctx->bits_per_raw_sample = 8;
15.
          h->cur_chroma_format_idc = 1;
16.
         //初始化DSP相关的汇编函数。包含了IDCT、环路滤波函数等
17.
          ff h264dsp init(&h->h264dsp, 8, 1);
18.
         av assert0(h->sps.bit depth chroma == 0);
          ff_h264chroma_init(&h->h264chroma, h->sps.bit_depth_chroma);
19.
20.
         //初始化四分之一像素运动补偿相关的汇编函数
21.
          ff h264gpel init(&h->h264gpel, 8);
         //初始化帧内预测相关的汇编函数
22.
23.
          \label{linear_h264_pred_init(h->hpc, h->avctx->codec_id, 8, 1);} \\
24.
25
          h->dequant_coeff_pps = -1;
26
         h->current_sps_id = -1;
```

```
28.
         /* needed so that IDCT permutation is known early */
29.
          if (CONFIG ERROR RESILIENCE)
30.
             ff me cmp init(&h->mecc. h->avctx):
31.
          ff videodsp init(&h->vdsp, 8);
32.
          memset(h->pps.scaling_matrix4, 16, 6 * 16 * sizeof(uint8_t));
33.
         memset(h->pps.scaling_matrix8, 16, 2 * 64 * sizeof(uint8_t));
34.
35.
36.
     h->picture_structure = PICT_FRAME;
37.
          h->slice_context_count = 1;
38.
     h->workaround_bugs = avctx->workaround_bugs;
39.
          h->flags
                                = avctx->flags;
40.
41.
          /* set defaults */
      // s->decode mb = ff h263 decode mb;
42.
43.
          if (!avctx->has_b_frames)
44.
      h \rightarrow low delay = 1;
45.
46.
     avctx->chroma_sample_location = AVCHROMA_LOC_LEFT;
          //初始化熵解码器
47.
      //CAVLC
48.
49.
          ff_h264_decode_init_vlc();
50.
      //CABAC
51.
          ff_init_cabac_states();
52.
       //8-bit H264取0, 大于 8-bit H264取1
53.
          h->pixel shift
                               = 0;
      h->sps.bit_depth_luma = avctx->bits_per_raw_sample = 8;
54.
55.
56.
     h->thread context[0] = h;
                              = h->next_outputed_poc = INT_MIN;
57.
          h->outputed poc
      for (i = 0; i < MAX DELAYED PIC COUNT; i++)</pre>
58.
59.
             h->last pocs[i] = INT MIN:
     h->prev poc msb = 1 << 16;
60.
61.
          h->prev_frame_num = -1;
62.
      h \rightarrow x264_build = -1;
63.
          h->sei_fpa.frame_packing_arrangement_cancel_flag = -1;
      ff_h264_reset_sei(h);
64.
65.
          if (avctx->codec_id == AV_CODEC_ID_H264) {
66.
          if (avctx->ticks_per_frame == 1) {
67.
                  if(h->avctx->time_base.den < INT_MAX/2) {</pre>
68.
                     h->avctx->time_base.den *= 2;
69.
70.
                 h->avctx->time base.num /= 2;
71.
72.
             avctx->ticks_per_frame = 2;
73.
74.
        //AVCodecContext中是否包含extradata?包含的话,则解析之
75.
          if (avctx->extradata size > 0 && avctx->extradata) {
76.
             ret = ff h264 decode extradata(h, avctx->extradata, avctx->extradata size);
77.
              if (ret < 0) {
78.
              ff h264 free context(h);
79.
                  return ret;
80.
81.
82.
83.
          if (h->sps.bitstream restriction flag &&
84.
              h->avctx->has_b_frames < h->sps.num_reorder_frames) {
              h->avctx->has_b_frames = h->sps.num_reorder_frames;
85.
86.
             h->low_delay
                             = 0;
87.
88.
89.
          avctx->internal->allocate progress = 1;
90.
91.
          ff h264 flush change(h);
92.
93.
          return 0:
94.
```

从函数定义中可以看出,ff_h264_decode_init()一方面给H.264 解码器中一些变量(例如bit_depth_luma、chroma_format_idc等)设定了初始值,另一方面调用了一系列汇编函数的初始化函数(初始化函数的具体内容在后续文章中完成)。初始化汇编函数的的步骤是:首先将C语言版本函数赋值给相应模块的函数指针;然后检测平台的特性,如果不支持汇编优化(ARM、X86等),则不再做任何处理,如果支持汇编优化,则将相应的汇编优化函数赋值给相应模块的函数指针(替换掉C语言版本的效率较低的函数)。下面几个函数初始化了几个不同模块的汇编优化函数:

```
ff_h264dsp_init():初始化DSP相关的汇编函数。包含了IDCT、环路滤波函数等。
```

ff h264qpel init():初始化四分之一像素运动补偿相关的汇编函数。

ff_h264_pred_init():初始化帧内预测相关的汇编函数。

可以举例看一下个ff_h264_pred_init()的代码。

ff_h264_pred_init()

函数用于初始化帧内预测相关的汇编函数,定位于libavcodec\h264pred.c,如下所示。

```
* Set the intra prediction function pointers.
3.
4.
      //初始化帧内预测相关的汇编函数
      av cold void ff h264 pred init(H264PredContext *h, int codec id,
5.
                                     const int bit depth,
6.
                                     int chroma format idc)
8.
9.
      #undef FUNC
      #undef FUNCC
10.
11.
      #define FUNC(a, depth) a ## \_ ## depth
12.
      #define FUNCC(a, depth) a ## \_ ## depth ## \_
13
      #define FUNCD(a) a ## _c
14.
      //好长的宏定义...(这种很长的宏定义在H.264解码器中似乎很普遍
15.
      //该宏用干给帧内预测模块的函数指针赋值
      //注意参数为颜色位深度
16
17
      #define H264_PRED(depth)
18.
       if(codec_id != AV_CODEC_ID_RV40){\
              if (codec_id == AV_CODEC_ID_VP7 || codec_id == AV_CODEC_ID_VP8) {\
19.
20.
                  h->pred4x4[VERT PRED ]= FUNCD(pred4x4 vertical vp8);\
21.
                  h->pred4x4[HOR_PRED
                                             ]= FUNCD(pred4x4_horizontal_vp8);\
22.
              } else {\
23.
                  h->pred4x4[VERT PRED
                                             l= FUNCC(pred4x4 vertical
                                                                                 . depth):\
                                             ]= FUNCC(pred4x4 horizontal
24.
                  h->pred4x4[HOR PRED
                                                                                 , depth);\
25.
              }\
26.
              h->pred4x4[DC PRED
                                             ]= FUNCC(pred4x4_dc
                                                                                 , depth);\
27.
              if(codec id == AV CODEC ID SVQ3)\
28
                  h->pred4x4[DIAG_DOWN_LEFT_PRED ]= FUNCD(pred4x4_down_left_svq3);\
29.
30.
                 h->pred4x4[DIAG DOWN LEFT PRED ]= FUNCC(pred4x4 down left
31.
              h->pred4x4[DIAG_DOWN_RIGHT_PRED]= FUNCC(pred4x4_down_right
32
              h->pred4x4[VERT RIGHT PRED
                                            ]= FUNCC(pred4x4_vertical_right
                                                                                , depth);\
33.
              h->pred4x4[HOR_DOWN_PRED
                                             ]= FUNCC(pred4x4_horizontal_down
                                                                                , depth);\
34.
              if (codec id == AV CODEC ID VP7 || codec id == AV CODEC ID VP8) {\
35.
                  h->pred4x4[VERT_LEFT_PRED ]= FUNCD(pred4x4_vertical_left_vp8);\
36.
              } else\
37.
                  h\text{->pred4x4[VERT\_LEFT\_PRED} \quad ]\text{= } \text{FUNCC(pred4x4\_vertical\_left}
                                                                                 , depth);\
                                             ]= FUNCC(pred4x4_horizontal_up
              h->pred4x4[HOR UP PRED
38.
                                                                                 , depth);\
39.
              if (codec id != AV CODEC ID VP7 && codec id != AV CODEC ID VP8) {\
                                                                             , depth);∖
40.
                  h->pred4x4[LEFT_DC_PRED ]= FUNCC(pred4x4_left_dc
41.
                  h->pred4x4[TOP_DC_PRED
                                             ]= FUNCC(pred4x4 top dc
                                                                                 , depth);\
                else {\
42.
43
                  h->pred4x4[TM_VP8_PRED
                                              ]= FUNCD(pred4x4_tm_vp8);\
44.
                  h->pred4x4[DC_127_PRED
                                             ]= FUNCC(pred4x4_127_dc
                                                                                 , depth);\
45.
                  h->pred4x4[DC_129_PRED
                                              ]= FUNCC(pred4x4_129_dc
                                                                                 , depth);\
46
                  h->pred4x4[VERT VP8 PRED
                                             ]= FUNCC(pred4x4 vertical
                                                                                 , depth);\
47.
                  h->pred4x4[HOR_VP8_PRED
                                             ]= FUNCC(pred4x4_horizontal
                                                                                 , depth);\
48.
49.
              if (codec_id != AV_CODEC_ID_VP8)\
50.
                  h->pred4x4[DC_128_PRED
                                           ]= FUNCC(pred4x4_128_dc
                                                                                 , depth);\
51.
          }else{\
             h->pred4x4[VERT PRED
                                             l= FUNCC(pred4x4 vertical
52.
                                                                                 . depth):\
                                                                                 , depth);∖
53.
              h->pred4x4[HOR PRED
                                             ]= FUNCC(pred4x4 horizontal
54.
              h->pred4x4[DC PRED
                                             1= FUNCC(pred4x4 dc
                                                                                 , depth);\
55.
              h->pred4x4[DIAG DOWN LEFT PRED ]= FUNCD(pred4x4 down left rv40);\
                                                                              , depth);∖
56.
              h->pred4x4[DIAG DOWN RIGHT PRED]= FUNCC(pred4x4 down right
57.
              h->pred4x4[VERT_RIGHT_PRED
                                             ]= FUNCC(pred4x4_vertical_right
                                                                                , depth);∖
58.
              h->pred4x4[HOR_DOWN_PRED
                                             ]= FUNCC(pred4x4_horizontal_down , depth);\
59
              h->pred4x4[VERT_LEFT_PRED
                                              ]= FUNCD(pred4x4_vertical_left_rv40);\
                                             ]= FUNCD(pred4x4_horizontal_up_rv40);\
60
              h->pred4x4[HOR_UP_PRED
61.
              h->pred4x4[LEFT DC PRED
                                              ]= FUNCC(pred4x4 left dc
62.
              h->pred4x4[TOP_DC_PRED
                                             l= FUNCC(pred4x4 top dc
                                                                                 . depth):\
63.
              h->pred4x4[DC 128 PRED
                                             ]= FUNCC(pred4x4 128 dc
                                                                                 , depth);\
64.
              h->pred4x4[DIAG_DOWN_LEFT_PRED_RV40_NODOWN]= FUNCD(pred4x4_down_left_rv40_nodown);
              h->pred4x4[HOR UP PRED RV40 NODOWN]= FUNCD(pred4x4 horizontal up rv40 nodown);\
65.
              h->pred4x4[VERT LEFT PRED RV40 NODOWN]= FUNCD(pred4x4 vertical left rv40 nodown);\
66.
67.
          }\
68.
      \
                                                                                 , depth);\
69.
          h->pred8x8l[VERT PRED
                                          ]= FUNCC(pred8x8l vertical
70.
          h->pred8x8l[HOR PRED
                                          ]= FUNCC(pred8x8l horizontal
                                                                                  , depth);\
          h->pred8x8l[DC PRED
71
                                          ]= FUNCC(pred8x8l dc
                                                                                 , depth);\
72.
          h->pred8x8l[DIAG_DOWN_LEFT_PRED ]= FUNCC(pred8x8l_down_left
                                                                                 , depth);\
73.
          h->pred8x8l[DIAG_DOWN_RIGHT_PRED]= FUNCC(pred8x8l_down_right
                                                                                   depth);\
74.
          h->pred8x8l[VERT_RIGHT_PRED
                                          ]= FUNCC(pred8x8l_vertical_right
                                                                                 , depth);\
          h->pred8x8l[HOR DOWN PRED
                                           ]= FUNCC(pred8x8l horizontal down
                                                                                 , depth);\
75
76.
          h->pred8x8l[VERT_LEFT_PRED
                                          ]= FUNCC(pred8x8l_vertical_left
                                                                                 . depth):\
77.
          h->pred8x8l[HOR UP PRED
                                           ]= FUNCC(pred8x8l horizontal up
                                                                                 , depth);\
78.
          h->pred8x8l[LEFT_DC_PRED
                                          ]= FUNCC(pred8x8l_left_dc
                                                                                 , depth);\
          h->pred8x8l[TOP DC PRED
                                           ]= FUNCC(pred8x8l top dc
79.
                                                                                 , depth);\
80.
          h->pred8x8l[DC 128 PRED
                                          l= FUNCC(pred8x8l 128 dc
                                                                                 . depth):\
81.
82.
          if (chroma format idc <= 1) {\</pre>
                                                                                 , depth);\
83.
              h->pred8x8[VERT PRED8x8
                                        ]= FUNCC(pred8x8 vertical
84.
              h->pred8x8[HOR PRED8x8
                                        ]= FUNCC(pred8x8 horizontal
                                                                                 , depth);\
85.
          } else {\
                                                                                 , depth);\
86
              h->pred8x8[VERT PRED8x8 ]= FUNCC(pred8x16 vertical
87.
              h->pred8x8[HOR_PRED8x8
                                        ]= FUNCC(pred8x16_horizontal
                                                                                 , depth);\
88
          if (codec_id != AV_CODEC_ID_VP7 && codec_id != AV_CODEC_ID_VP8) {\
89
90
              if (chroma format idc <= 1) {\</pre>
```

```
h->pred8x8[PLANE PRED8x8]= FUNCC(pred8x8 plane
 91.
                                                                               . depth):\
 92.
        } else {\
 93.
                  h->pred8x8[PLANE PRED8x8]= FUNCC(pred8x16 plane
                                                                               , depth);\
 94.
              }\
 95.
          } else\
 96.
              h->pred8x8[PLANE_PRED8x8]= FUNCD(pred8x8_tm_vp8);\
 97.
          if (codec_id != AV_CODEC_ID_RV40 && codec_id != AV_CODEC_ID_VP7 && \
 98.
              codec_id != AV_CODEC_ID_VP8) {\
 99.
               if (chroma_format_idc <= 1) {\</pre>
                  h->pred8x8[DC PRED8x8
                                          ]= FUNCC(pred8x8 dc
100.
                   h->pred8x8[LEFT_DC_PRED8x8]= FUNCC(pred8x8_left_dc
                                                                                  , depth);\
101.
                  h->pred8x8[TOP_DC_PRED8x8 ]= FUNCC(pred8x8_top_dc
102.
103.
                   h->pred8x8[ALZHEIMER_DC_LOT_PRED8x8 ]= FUNC(pred8x8_mad_cow_dc_l0t, depth);\
                  h->pred8x8[ALZHEIMER DC 0LT PRED8x8 ]= FUNC(pred8x8 mad cow dc 0lt, depth);\
104.
                  h->pred8x8[ALZHEIMER DC L00 PRED8x8 ]= FUNC(pred8x8 mad cow dc l00, depth);\
105.
                  h->pred8x8[ALZHEIMER DC 0L0 PRED8x8 ]= FUNC(pred8x8 mad cow dc 0l0, depth);\
106.
107.
               } else {\
                                                                                   , depth);\
108.
                  h->pred8x8[DC PRED8x8
                                         ]= FUNCC(pred8x16 dc
                   h->pred8x8[LEFT_DC_PRED8x8]= FUNCC(pred8x16_left_dc
                                                                                  , depth);\
109.
                  h->pred8x8[TOP_DC_PRED8x8 ]= FUNCC(pred8x16_top_dc
110.
111.
                  h->pred8x8[ALZHEIMER_DC_L0T_PRED8x8 ]= FUNC(pred8x16_mad_cow_dc_l0t, depth);\
112
                  113.
                  h->pred8x8[ALZHEIMER_DC_L00_PRED8x8 ]= FUNC(pred8x16_mad_cow_dc_l00, depth);\
114
                  h->pred8x8[ALZHEIMER_DC_0L0_PRED8x8 ]= FUNC(pred8x16_mad_cow_dc_0l0, depth);\
115.
116.
           }else{\
117.
              h->pred8x8[DC PRED8x8
                                        ]= FUNCD(pred8x8_dc_rv40);\
118.
              h->pred8x8[LEFT_DC_PRED8x8]= FUNCD(pred8x8_left_dc_rv40);\
119.
               h->pred8x8[TOP DC PRED8x8 ]= FUNCD(pred8x8 top dc rv40);\
              if (codec id == AV CODEC ID VP7 || codec id == AV CODEC ID VP8) {\
120.
                  h->pred8x8[DC 127 PRED8x8]= FUNCC(pred8x8 127 dc
                                                                              , depth);\
121.
                  h->pred8x8[DC_129_PRED8x8]= FUNCC(pred8x8 129 dc
122.
                                                                              , depth);\
123.
              }\
124.
125.
           if (chroma format idc <= 1) {\</pre>
126.
              h\text{->}pred8x8[DC\_128\_PRED8x8 ] = FUNCC(pred8x8\_128\_dc
127.
           } else {\
128.
              h->pred8x8[DC_128_PRED8x8 ]= FUNCC(pred8x16_128_dc
129.
130.
131.
           h->pred16x16[DC_PRED8x8
                                      ]= FUNCC(pred16x16_dc
                                                                               , depth);\
          h->pred16x16[VERT_PRED8x8 ]= FUNCC(pred16x16_vertical
132.
                                                                               , depth);\
           h->pred16x16[HOR PRED8x8
                                     ]= FUNCC(pred16x16 horizontal
133.
                                                                                depth);\
134.
          switch(codec id){\
           case AV CODEC ID SVQ3:\
135.
136.
          h->pred16x16[PLANE_PRED8x8 ]= FUNCD(pred16x16_plane_svq3);\
137.
             break:\
           case AV CODEC ID RV40:\
138.
139.
             h->pred16x16[PLANE_PRED8x8 ]= FUNCD(pred16x16_plane_rv40);\
140.
            break;\
141.
           case AV_CODEC_ID_VP7:\
142.
           case AV_CODEC_ID_VP8:\
              h->pred16x16[PLANE PRED8x8 ]= FUNCD(pred16x16 tm vp8);\
143.
             h->pred16x16[DC_127_PRED8x8]= FUNCC(pred16x16_127_dc
144.
                                                                               , depth);\
145.
              h->pred16x16[DC_129_PRED8x8]= FUNCC(pred16x16_129_dc
                                                                              , depth);∖
146.
147.
           default:\
148.
             h->pred16x16[PLANE PRED8x8 ]= FUNCC(pred16x16 plane
                                                                               , depth);\
149.
              break; \
       }\
150.
           h->pred16x16[LEFT DC PRED8x8]= FUNCC(pred16x16 left dc
151.
                                                                               , depth);\
          h->pred16x16[TOP_DC_PRED8x8_]= FUNCC(pred16x16_top_dc_
                                                                               , depth):\
152.
           h->pred16x16[DC 128 PRED8x8 ]= FUNCC(pred16x16 128 dc
                                                                               , depth);\
153.
154.
155.
           /* special lossless h/v prediction for h264 */ \
156.
          h->pred4x4_add [VERT_PRED ]= FUNCC(pred4x4_vertical_add
                                                                               , depth);\
157
           h->pred4x4_add [ HOR_PRED
                                       ]= FUNCC(pred4x4_horizontal_add
                                                                               , depth);\
           h->pred8x8l_add [VERT_PRED ]= FUNCC(pred8x8l_vertical_add
158.
                                                                               , depth);\
           h->pred8x8l_add [ HOR_PRED
                                       ]= FUNCC(pred8x8l_horizontal_add
159.
                                                                               , depth);\
          h->pred8x8l_filter_add [VERT_PRED ]= FUNCC(pred8x8l_vertical_filter_add
160.
                                                                                            , depth);\
161.
           h->pred8x8l_filter_add [ HOR_PRED
                                             ]= FUNCC(pred8x8l_horizontal_filter_add
                                                                                            , depth);\
162.
       if (chroma format idc <= 1) {\</pre>
163.
           h->pred8x8_add [VERT_PRED8x8]= FUNCC(pred8x8_vertical_add
                                                                               , depth);\
164.
          h->pred8x8_add [ HOR_PRED8x8]= FUNCC(pred8x8_horizontal_add
                                                                              , depth);\
165.
           } else {\
            h->pred8x8 add [VERT PRED8x8]= FUNCC(pred8x16 vertical add
                                                                                , depth);\
166.
                                                                                   , depth);\
167.
              h->pred8x8 add [ HOR PRED8x8] = FUNCC(pred8x16 horizontal add
168.
                                                                               , depth);\
169.
           h->pred16x16_add[VERT_PRED8x8]= FUNCC(pred16x16_vertical_add
170.
          h->pred16x16_add[ HOR_PRED8x8]= FUNCC(pred16x16_horizontal_add
                                                                               , depth);\
           //注意这里使用了前面那个很长的宏定义
171.
          //根据颜色位深的不同,初始化不同的函数
172
173.
           //颜色位深默认值为8,所以一般情况下调用H264_PRED(8)
174.
          switch (bit_depth) {
              case 9:
175.
176.
               H264 PRED(9)
177.
                  break;
178.
               case 10:
179.
                  H264 PRED(10)
180.
                  break:
181
              case 12:
```

```
H264 PRED(12)
182.
183
                   break
184.
                case 14:
185.
                   H264 PRED(14)
186.
                   break:
187.
                default:
188.
                   av_assert0(bit_depth<=8);</pre>
                   H264 PRED(8)
189.
190.
                   break:
191.
        ·
//如果支持汇编优化,则会调用相应的汇编优化函数
192.
193.
           //neon这些的
194.
           if (ARCH ARM) ff h264 pred init arm(h, codec id, bit depth, chroma format idc);
           //mmx这些的
195.
196.
           if (ARCH_X86) ff_h264_pred_init_x86(h, codec_id, bit_depth, chroma_format_idc);
197. }
```

初看一眼ff_h264_pred_init()定义会给人一种很奇怪的感觉:前面的这个H264_PRED(depth)的宏定义怎么这么长?!实际上在FFmpeg的H.264解码器中这种很长的宏定义是很常见的。我个人觉得这么做主要是为了方便为不同的颜色位深的码流初始化不同的功能函数。例如,对于常见的8bit码流,调用H264_PRED(8)就可以初始化相应的函数;对于比较新的10bit码流,调用H264_PRED(10)就可以初始化相应的函数。

ff_h264_pred_init()的代码是开始于switch()语句的,可以看出该函数根据不同的bit_depth(颜色位深)调用了不同的H264_PRED(bit_depth)宏。 我们不妨展开一个H264_PRED()宏看看里面的代码究竟是什么。在这里我们选择最常见的8bit为例,看看H264_PRED(8)宏展开后的结果。

H264 PRED(8)

H264_PRED(8)用于初始化8bit颜色位深C语言版本的帧内预测的函数。该宏定义展开后的结果如下所示。

```
1.
      if(codec id != AV CODEC ID RV40){
2.
         if (codec_id == AV_CODEC_ID_VP7 || codec_id == AV_CODEC_ID_VP8) {
3.
              h->pred4x4[0
                             ]= pred4x4_vertical_vp8_c;
4.
              h->pred4x4[1
                               ]= pred4x4_horizontal_vp8_c;
 5.
          } else {
 6.
             //帧内4x4的Vertical预测方式
              h->pred4x4[0
                                ]= pred4x4 vertical 8 c;
 7.
8.
              //帧内4x4的Horizontal预测方式
9.
              h->pred4x4[1
                                 ]= pred4x4 horizontal 8 c;
10.
          //帧内4x4的DC预测方式
11.
      h->pred4x4[2
                              ]= pred4x4_dc_8_c;
12.
13.
          if(codec id == AV CODEC ID SVQ3)
             h->pred4x4[3 ]= pred4x4_down_left_svq3_c;
14.
15.
          else
16.
            h->pred4x4[3 ]= pred4x4_down_left_8_c;
17.
          h->pred4x4[4]= pred4x4_down_right_8_c;
18.
         h->pred4x4[5 ]= pred4x4_vertical_right_8_c;
19.
          h->pred4x4[6
                             ]= pred4x4_horizontal_down_8_c;
      if (codec_id == AV_CODEC_ID_VP7 || codec_id == AV_CODEC_ID_VP8)
20.
21.
              h->pred4x4[7 ]= pred4x4_vertical_left_vp8_c;
22.
23.
             h->pred4x4[7 ]= pred4x4_vertical_left_8_c;
24.
          h->pred4x4[8
                              ]= pred4x4 horizontal up 8 c;
          if (codec id != AV CODEC ID VP7 && codec id != AV CODEC ID VP8) {
25.
             h->pred4x4[9 ]= pred4x4_left_dc_8_c;
26.
27.
              h->pred4x4[10
                              ]= pred4x4_top_dc_8_c;
28.
          } else {
             h->pred4x4[9 ]= pred4x4_tm_vp8_c;
h->pred4x4[12 ]= pred4x4_127_dc_8_c;
29.
30.
31.
              h->pred4x4[13
                               ]= pred4x4_129_dc_8_c;
32.
             h->pred4x4[10 ]= pred4x4_vertical_8_c;
33.
              h->pred4x4[14
                              ]= pred4x4_horizontal_8_c;
34.
35.
          if (codec_id != AV_CODEC_ID_VP8)
36.
              h->pred4x4[11 ]= pred4x4_128_dc_8_c;
37.
      }else{
          h->pred4x4[0 ]= pred4x4 vertical 8 c;
38.
          h->pred4x4[1
39.
                                 l= pred4x4 horizontal 8 c:
         h->pred4x4[2
                                  ]= pred4x4 dc 8 c;
40.
41.
          h->pred4x4[3 ]= pred4x4 down left rv40 c:
42.
         h->pred4x4[4]= pred4x4_down_right_8_c;
43.
          h->pred4x4[5
                          ]= pred4x4 vertical right 8 c;
         h->pred4x4[6 ]= pred4x4_horizontal_down_8_c;
44.
45.
          h->pred4x4[7
                            ]= pred4x4 vertical left rv40 c;
46.
         h->pred4x4[8
                          ]= pred4x4_horizontal_up_rv40_c;
47.
          h->pred4x4[9
                              ]= pred4x4_left_dc_8_c;
48.
          h->pred4x4[10
                            ]= pred4x4_top_dc_8_c;
          h->pred4x4[11
                                ]= pred4x4 128 dc 8 c;
49.
50.
          h->pred4x4[12]= pred4x4_down_left_rv40_nodown_c;
51.
          h->pred4x4[13]= pred4x4_horizontal_up_rv40_nodown_c;
52.
          h->pred4x4[14]= pred4x4_vertical_left_rv40_nodown_c;
53.
54.
55.
      h->pred8x8l[0
                              ]= pred8x8l vertical 8 c;
56.
      h->pred8x8l[1
                              l= pred8x8l horizontal 8 c:
      h->pred8x8l[2
                                ]= pred8x8l dc 8 c;
```

```
h->pred8x8l[3 ]= pred8x8l_down_left_8_c;
        h->pred8x8l[4]= pred8x8l_down_right_8_c;
 59.
        h->pred8x8l[5
                        ]= pred8x8l_vertical_right_8_c;
 60.
                            ]= pred8x8l_horizontal_down_8_c;
 61.
        h->pred8x8l[6
 62.
        h->pred8x8l[7
                           ]= pred8x8l_vertical_left_8_c;
        h->pred8x8l[8
                               ]= pred8x8l horizontal up 8 c;
 63.
       h->pred8x8l[9
                              ]= pred8x8l left dc 8 c;
 64.
       h->pred8x8l[10
                               ]= pred8x8l top dc 8 c;
 65.
                               ]= pred8x8l_128_dc_8_c;
       h->pred8x8l[11
 66.
 67.
 68.
       if (chroma format idc <= 1) {</pre>
           h->pred8x8[2 ]= pred8x8_vertical_8_c;
h->pred8x8[1 ]= pred8x8_horizontal_8_c;
 69.
 70.
 71.
        } else {
 72.
           h->pred8x8[2 ]= pred8x16_vertical_8_c;
            h->pred8x8[1
 73.
                            ]= pred8x16_horizontal_8_c;
 74.
 75.
        if (codec_id != AV_CODEC_ID_VP7 && codec_id != AV_CODEC_ID_VP8) {
 76.
        if (chroma_format_idc <= 1) {</pre>
 77.
                h->pred8x8[3]= pred8x8_plane_8_c;
 78.
            } else {
 79.
                h->pred8x8[3]= pred8x16_plane_8_c;
 80.
 81.
       } else
           h->pred8x8[3]= pred8x8 tm vp8 c;
 82.
       if (codec_id != AV_CODEC_ID_RV40 && codec_id != AV_CODEC_ID VP7 &&
 83.
            codec_id != AV_CODEC_ID_VP8) {
 84.
 85.
            if (chroma_format_idc <= 1) {</pre>
 86.
                h->pred8x8[0 ]= pred8x8_dc_8_c;
 87.
                h->pred8x8[4]= pred8x8_left_dc_8_c;
 88.
                h->pred8x8[5 ]= pred8x8_top_dc_8_c;
 89.
                h->pred8x8[7 ]= pred8x8_mad_cow_dc_l0t_8;
 90.
                h->pred8x8[8 ]= pred8x8_mad_cow_dc_0lt_8;
                h->pred8x8[9 ]= pred8x8_mad_cow_dc_l00_8;
 91.
 92.
                h->pred8x8[10 ]= pred8x8 mad cow dc 0l0 8;
 93.
            } else {
 94.
                h \rightarrow pred8x8[0] = pred8x16_dc_8_c;
 95.
                h->pred8x8[4]= pred8x16 left dc 8 c:
                h->pred8x8[5] = pred8x16 top dc 8 c;
 96.
 97.
                h->pred8x8[7 ]= pred8x16_mad_cow_dc_l0t_8;
 98.
                h->pred8x8[8 ]= pred8x16_mad_cow_dc_0lt_8;
 99.
                h->pred8x8[9 ]= pred8x16_mad_cow_dc_l00_8;
100.
                h->pred8x8[10 ]= pred8x16_mad_cow_dc_0l0_8;
101.
102.
       }else{
103.
            h->pred8x8[0
                              ]= pred8x8_dc_rv40_c;
            h->pred8x8[4]= pred8x8_left_dc_rv40_c;
104.
105.
            h->pred8x8[5] = pred8x8_top_dc_rv40_c;
            if (codec id == AV CODEC ID VP7 || codec id == AV CODEC ID VP8)
106.
107.
                h->pred8x8[7]= pred8x8_127_dc_8_c;
108.
                h->pred8x8[8]= pred8x8_129_dc_8_c;
109.
            }
110.
111.
       if (chroma format idc <= 1) {</pre>
           h->pred8x8[6 ]= pred8x8_128_dc_8_c;
112.
113.
        } else {
114
           h->pred8x8[6 ]= pred8x16_128_dc_8_c;
115.
116.
117.
        h->pred16x16[0
                           ]= pred16x16_dc_8_c;
118.
        h->pred16x16[2 ]= pred16x16_vertical_8_c;
119.
        h->pred16x16[1
                          ]= pred16x16_horizontal_8_c;
        switch(codec_id){
120.
121.
        case AV CODEC ID SVQ3:
122.
           h->pred16x16[3 ]= pred16x16_plane_svq3_c;
123.
           break:
       case AV CODEC ID RV40:
124.
125.
           h->pred16x16[3 ]= pred16x16 plane rv40 c;
126.
           break:
127.
        case AV CODEC ID VP7:
128.
       case AV CODEC ID VP8:
129.
           h->pred16x16[3 ]= pred16x16_tm_vp8_c;
130.
           h->pred16x16[7]= pred16x16_127_dc_8_c;
           h->pred16x16[8]= pred16x16_129_dc_8_c;
131.
132.
           break;
133.
        default:
           h->pred16x16[3 ]= pred16x16_plane_8_c;
134.
135.
           break;
136.
137.
       h->pred16x16[4]= pred16x16 left dc 8 c;
138.
       h->pred16x16[5 ]= pred16x16 top dc 8 c:
139.
       h->pred16x16[6] = pred16x16 128 dc 8 c;
140.
141.
        /* special lossless h/v prediction for h264 */
       h->pred4x4_add [0 ]= pred4x4_vertical_add_8_c;
h->pred4x4_add [ 1 ]= pred4x4_horizontal_add_8_c;
142.
143.
       h->pred8x8l_add [0 ]= pred8x8l_vertical_add_8_c;
h->pred8x8l_add [ 1 ]= pred8x8l_horizontal_add_8_c;
144.
145.
146
       h->pred8x8l_filter_add [0 ]= pred8x8l_vertical_filter_add_8_c;
147.
        h->pred8x8l_filter_add [ 1 ]= pred8x8l_horizontal_filter_add_8_c;
       if (chroma_format_idc <= 1) {</pre>
```

```
149. h->pred8x8_add [2]= pred8x8_vertical_add_8_c;
150. h->pred8x8_add [1]= pred8x8_horizontal_add_8_c;
151. } else {
152. h->pred8x8_add [2]= pred8x16_vertical_add_8_c;
153. h->pred8x8_add [1]= pred8x16_horizontal_add_8_c;
154. }
155. h->pred16x16_add[2]= pred16x16_vertical_add_8_c;
156. h->pred16x16_add[1]= pred16x16_horizontal_add_8_c;
```

可以看出在H264_PRED(8)展开后的代码中,帧内预测模块的函数指针都被赋值以xxxxx_8_c()的函数。例如帧内4x4的模式0被赋值以pred4x4_vertical_8_c(),帧内4x4的模式1被赋值以pred4x4_horizontal_8_c(),如下所示。

在这里我们可以简单看一下pred4x4_vertical_8_c()函数。该函数完成了4x4帧内Vertical模式的预测。

pred4x4_vertical_8_c()

pred4x4_vertical_8_c()的定义位于libavcodec\h264pred_template.c,如下所示。

```
[cpp] 📳 📑
     //垂直预测
2.
     //由上边像素推出像素值
     static void pred4x4_vertical_8_c (uint8_t *_src, const uint8_t *topright,
3.
         ptrdiff_t _stride)
4.
5.
     pixel *src = (pixel*)_src;
6.
        int stride = _stride>>(sizeof(pixel)-1);
7.
8.
9.
     * Vertical预测方式
10.
          * | X1 X2 X3 X4
11.
12.
13.
             |X1 X2 X3 X4
        * |X1 X2 X3 X4
14.
15.
             |X1 X2 X3 X4
     * |X1 X2 X3 X4
16.
17.
18.
19.
     //pixel4代表4个像素值。1个像素值占用8bit,4个像素值占用32bit。
20.
21.
         const pixel4 a= AV_RN4PA(src-stride);
     /* 宏定义展开后:
22.
          * const uint32 t a=(((const av alias32*)(src-stride))->u32);
23.
     * 注:av alias32是一个union类型的变量,存储4byte的int或者float。
24.
          * -stride代表了上一行对应位置的像素
25.
     * 即a取的是上1行像素的值。
26.
27.
28.
     AV_WN4PA(src+0*stride, a);
29.
         AV WN4PA(src+1*stride, a):
     AV_WN4PA(src+2*stride, a);
30.
31.
        AV_WN4PA(src+3*stride, a);
32.
33.
         /* 宏定义展开后:
     * (((av alias32*)(src+0*stride))->u32 = (a));
34.
          * (((av_alias32*)(src+1*stride))->u32 = (a));
35.
         * (((av alias32*)(src+2*stride))->u32 = (a));
36.
          * (((av alias32*)(src+3*stride))->u32 = (a));
37.
         * 即a把a的值赋给下面4行。
38.
39.
40.
41.
     }
```

有关pred4x4_vertical_8_c()的代码在后续文章中再做详细分析,在这里就不再做过多解释了。

ff_h264_pred_init_x86()

当系统支持ARM汇编优化的时候(ARCH_ARM取值为1),就会调用ff_h264_pred_init_arm()初始化ARM平台下帧内预测汇编优化的函数;当系统支持X86汇编优化的时候(ARCH_X86取值为1),就会调用ff_h264_pred_init_x86()初始化X86平台下帧内预测汇编优化的函数。在这里我们简单看一下ff_h264_pred_init_x86()的定义。ff_h264_pred_init_x86()的定义位于libavcodec\x86\h264_intrapred_init.c,如下所示。

```
int cpu flags = av get cpu flags();
6.
          if (bit depth == 8) {
              if (EXTERNAL MMX(cpu flags)) {
8.
9.
                  h->pred16x16[VERT PRED8x8
                                                     ] = ff pred16x16 vertical 8 mmx;
10.
                  h->pred16x16[HOR PRED8x8
                                                     ] = ff_pred16x16_horizontal_8_mmx;
11
                   if (chroma_format_idc <= 1) {</pre>
12.
                      h->pred8x8 [VERT_PRED8x8
                                                     ] = ff_pred8x8_vertical_8_mmx;
13.
                      h->pred8x8 [HOR PRED8x8
                                                     ] = ff pred8x8 horizontal 8 mmx;
14.
15.
                   if (codec_id == AV_CODEC_ID_VP7 || codec_id == AV_CODEC_ID_VP8) {
                      h->pred16x16[PLANE_PRED8x8 ] = ff_pred16x16_tm_vp8_8_mmx;
16.
17.
                       h->pred8x8 [PLANE PRED8x8
                                                     ] = ff pred8x8 tm vp8 8 mmx;
18.
                      h->pred4x4 [TM_VP8_PRED
                                                    ] = ff pred4x4 tm vp8 8 mmx;
19.
                  } else {
20.
                    if (chroma format idc <= 1)</pre>
                          h->pred8x8 [PLANE_PRED8x8] = ff_pred8x8_plane_8_mmx;
21.
                       if (codec id == AV CODEC ID SVO3) {
22.
23.
                           if (cpu flags & AV CPU FLAG CMOV)
24.
                              h->pred16x16[PLANE_PRED8x8] = ff_pred16x16_plane_svq3_8_mm
25.
                       } else if (codec_id == AV_CODEC_ID_RV40) {
26
                          h->pred16x16[PLANE_PRED8x8] = ff_pred16x16_plane_rv40_8_mmx;
27.
28
                          h->pred16x16[PLANE_PRED8x8] = ff_pred16x16_plane_h264_8_mmx;
29
30.
31.
32.
              if (EXTERNAL MMXEXT(cpu flags)) {
33.
34.
                  h->pred16x16[HOR PRED8x8
                                                       l = ff pred16x16 horizontal 8 mmxext:
35.
                   h->pred16x16[DC PRED8x8
                                                       ] = ff pred16x16 dc 8 mmxext;
                   if (chroma format idc <= 1)</pre>
36.
37.
                      h->pred8x8[HOR PRED8x8
                                                        ] = ff pred8x8 horizontal 8 mmxext;
38.
                   h->pred8x8l [TOP DC PRED
                                                        ] = ff_pred8x8l_top_dc_8_mmxext;
39.
                  h->pred8x8l [DC PRED
                                                        ] = ff pred8x8l dc 8 mmxext;
40.
                  h->pred8x8l [HOR_PRED
                                                        ] = ff_pred8x8l_horizontal_8_mmxext;
                   h->pred8x8l [VERT_PRED
41.
                                                        ] = ff_pred8x8l_vertical_8_mmxext;
42.
                  h->pred8x8l [DIAG_DOWN_RIGHT_PRED
                                                       ] = ff_pred8x8l_down_right_8_mmxext;
43.
                   h->pred8x8l [VERT_RIGHT_PRED
                                                       ] = ff_pred8x8l_vertical_right_8_mmxext;
44.
                  h->pred8x8l [HOR UP PRED
                                                       ] = ff_pred8x8l_horizontal_up_8_mmxext;
45.
                   h->pred8x8l [DIAG_DOWN_LEFT_PRED
                                                       ] = ff_pred8x8l_down_left_8_mmxext;
46.
                  h->pred8x8l [HOR DOWN PRED
                                                       ] = ff pred8x8l horizontal down 8 mmxext
47.
                  h->pred4x4 [DIAG_DOWN_RIGHT_PRED
                                                       ] = ff_pred4x4_down_right_8_mmxext;
                                                       ] = ff_pred4x4_vertical_right_8_mmxext;
48.
                  h->pred4x4 [VERT RIGHT PRED
                   h->pred4x4 [HOR DOWN PRED
                                                       ] = ff pred4x4_horizontal_down_8_mmxext;
49.
                  h->pred4x4 [DC_PRED
                                                       ] = ff_pred4x4_dc_8_mmxext;
50.
                   if (codec_id == AV_CODEC_ID_VP7 || codec_id == AV_CODEC_ID_VP8 ||
51.
                      codec_id == AV_CODEC ID H264) {
52.
53.
                      h->pred4x4 [DIAG DOWN LEFT PRED] = ff pred4x4 down left 8 mmxext;
54.
55.
                   if (codec_id == AV_CODEC_ID_SVQ3 || codec_id == AV_CODEC_ID_H264) {
                      h->pred4x4 [VERT_LEFT_PRED ] = ff_pred4x4_vertical_left_8_mmxext;
56
57.
58
                   if (codec_id != AV_CODEC_ID_RV40) {
59.
                      h->pred4x4 [HOR_UP_PRED
                                                       ] = ff_pred4x4_horizontal_up_8_mmxext;
60.
61.
                   if (codec_id == AV_CODEC_ID_SVQ3 || codec_id == AV_CODEC_ID_H264) {
62.
                      if (chroma format idc <= 1) {</pre>
                          h->pred8x8[TOP DC PRED8x8
                                                       1 = ff pred8x8 top dc 8 mmxext:
63.
                          h->pred8x8[DC_PRED8x8 ] = ff_pred8x8_dc_8_mmxext;
64.
65.
66.
67.
                   if (codec_id == AV_CODEC_ID_VP7 || codec_id == AV_CODEC_ID_VP8) {
68.
                      \label{eq:h-pred16x16} $$h$->pred16x16[PLANE\_PRED8x8] = ff\_pred16x16\_tm\_vp8\_8\_mmxext;
69
                       h->pred8x8 [DC PRED8x8
                                                       ] = ff_pred8x8_dc_rv40_8_mmxext;
70.
                      h->pred8x8 [PLANE_PRED8x8
                                                       ] = ff_pred8x8_tm_vp8_8_mmxext;
71
                       h->pred4x4 [TM_VP8_PRED
                                                        ] = ff_pred4x4_tm_vp8_8_mmxext;
72.
                      h->pred4x4 [VERT_PRED
                                                      ] = ff_pred4x4_vertical_vp8_8_mmxext;
73
74.
                      if (chroma_format_idc <= 1)</pre>
75.
                           h->pred8x8 [PLANE_PRED8x8] = ff_pred8x8_plane_8_mmxext;
                          (codec_id == AV_CODEC_ID_SVQ3) {
76.
77.
                           h->pred16x16[PLANE PRED8x8 ] = ff pred16x16 plane svq3 8 mmxext;
                        else if (codec id == AV CODEC ID RV40) {
78.
                          \label{eq:h-pred16x16_plane_rv40_8_mmxext} h-\mbox{-pred16x16_plane\_rv40_8\_mmxext};
79.
80.
                       } else {
81.
                          h->pred16x16[PLANE PRED8x8 ] = ff pred16x16 plane h264 8 mmxext;
82.
83.
84.
85.
               if (EXTERNAL_SSE(cpu_flags)) {
86
                  h->pred16x16[VERT_PRED8x8] = ff_pred16x16_vertical_8_sse;
87.
88.
89.
90.
               if (EXTERNAL_SSE2(cpu_flags)) {
                  h->pred16x16[DC PRED8x8
                                                     ] = ff pred16x16 dc 8 sse2;
91.
                  h->pred8x8l [DIAG_DOWN_LEFT_PRED ] = ff_pred8x8l_down_left_8_sse2;
92.
                   h->pred8x8l [DIAG DOWN RIGHT_PRED ] = ff_pred8x8l_down_right_8_sse2;
93.
                  h->pred8x8l [VERT RIGHT PRED
                                                   ] = ff pred8x8l vertical right 8 sse2;
94.
                  h->nred8x81 [VFRT | FFT PRFD
                                                     1 = ff nred8x81 vertical left 8 sse2:
```

```
96
                   h->pred8x8l [HOR_DOWN_PRED
                                                 ] = ff_pred8x8l_horizontal_down_8_sse2;
 97
                    if (codec_id == AV_CODEC_ID_VP7 || codec_id == AV_CODEC_ID_VP8) {
                       h->pred16x16[PLANE_PRED8x8 ] = ff_pred16x16_tm_vp8_8_sse2;
 98
 99.
                        h->pred8x8 [PLANE PRED8x8
                                                      ] = ff_pred8x8_tm_vp8_8_sse2;
100
                    else {
101.
                       if (chroma_format_idc <= 1)</pre>
                           h->pred8x8 [PLANE_PRED8x8] = ff_pred8x8_plane_8_sse2;
102
                        if (codec id == AV CODEC ID SVQ3) {
103.
                           h->pred16x16[PLANE_PRED8x8] = ff_pred16x16_plane_svq3_8_sse2;
104
                       } else if (codec id == AV CODEC ID RV40) {
105.
106
                          h->pred16x16[PLANE_PRED8x8] = ff_pred16x16_plane_rv40_8_sse2;
107
                       } else {
108
                           h->pred16x16[PLANE_PRED8x8] = ff_pred16x16_plane_h264_8_sse2;
109.
                       }
110
111
112
113.
                if (EXTERNAL_SSSE3(cpu_flags)) {
114.
                   h->pred16x16[HOR_PRED8x8
                                                      ] = ff_pred16x16_horizontal_8_ssse3;
                                                      ] = ff_pred16x16_dc_8_ssse3;
115
                    h->pred16x16[DC_PRED8x8
                   if (chroma format idc <= 1)</pre>
116.
117.
                       h->pred8x8 [HOR PRED8x8
                                                      1 = ff pred8x8 horizontal_8_ssse3;
                   h->pred8x8l [TOP_DC_PRED
                                                      ] = ff_pred8x8l_top_dc_8_ssse3;
118.
                   h->pred8x8l [DC PRED
119.
                                                      ] = ff_pred8x8l_dc_8_ssse3;
                   h->pred8x8l [HOR_PRED
                                                      ] = ff_pred8x8l_horizontal_8_ssse3;
120.
121.
                   h->pred8x8l [VERT PRED
                                                      ] = ff_pred8x8l_vertical_8_ssse3;
122.
                   h->pred8x8l [DIAG_DOWN_LEFT_PRED ] = ff_pred8x8l_down_left_8_ssse3;
123
                    h->pred8x8l [DIAG_DOWN_RIGHT_PRED ] = ff_pred8x8l_down_right_8_ssse3;
124.
                    h->pred8x8l [VERT_RIGHT_PRED
                                                   ] = ff_pred8x8l_vertical_right_8_ssse3;
125
                    h->pred8x8l [VERT_LEFT_PRED
                                                      ] = ff_pred8x8l_vertical_left_8_ssse3;
126
                    h->pred8x8l [HOR UP PRED
                                                      ] = ff_pred8x8l_horizontal_up_8_ssse3;
127
                    h->pred8x8l [HOR_DOWN_PRED
                                                      ] = ff_pred8x8l_horizontal_down_8_ssse3;
128
                    if (codec_id == AV_CODEC_ID_VP7 || codec_id == AV_CODEC_ID_VP8) {
129.
                        h->pred8x8 [PLANE_PRED8x8
                                                      ] = ff_pred8x8_tm_vp8_8_ssse3;
130.
                       h->pred4x4 [TM_VP8_PRED
                                                      ] = ff_pred4x4_tm_vp8_8_ssse3;
131.
                   } else {
132
                      if (chroma format idc <= 1)</pre>
                            h->pred8x8 [PLANE_PRED8x8] = ff_pred8x8_plane_8_ssse3;
133.
                        if (codec_id == AV_CODEC_ID_SVQ3) {
134
                            h->pred16x16[PLANE\_PRED8x8] = ff\_pred16x16\_plane\_svq3\_8\_ssse3;
135.
136
                          else if (codec id == AV CODEC ID RV40) {
137.
                            h\text{->pred16x16[PLANE\_PRED8x8] = ff\_pred16x16\_plane\_rv40\_8\_ssse3;}
138
                         else {
139.
                            h->pred16x16[PLANE_PRED8x8] = ff_pred16x16_plane_h264_8_ssse3;
140
141
142
           } else if (bit_depth == 10) {
143.
144.
               if (EXTERNAL_MMXEXT(cpu_flags)) {
145.
                                                   ] = ff pred4x4 dc 10 mmxext;
                   h->pred4x4[DC PRED
                   h->pred4x4[HOR_UP_PRED
146.
                                                  ] = ff pred4x4 horizontal up 10 mmxext;
147.
148.
                    if (chroma format idc <= 1)</pre>
149
                       h->pred8x8[DC PRED8x8
                                                   ] = ff pred8x8 dc 10 mmxext;
150
151
                   h->pred8x8l[DC_128_PRED
                                                   ] = ff_pred8x8l_128_dc_10_mmxext;
152.
153
                    h->pred16x16[DC_PRED8x8
                                                    ] = ff pred16x16 dc 10 mmxext;
154
                   h->pred16x16[TOP_DC_PRED8x8
                                                   ] = ff_pred16x16_top_dc_10_mmxext;
155
                    h->pred16x16[DC 128 PRED8x8
                                                    ] = ff_pred16x16_128_dc_10_mmxext;
                                                   ] = ff_pred16x16_left_dc_10_mmxext;
156
                    h->pred16x16[LEFT_DC_PRED8x8
157
                    h->pred16x16[VERT_PRED8x8
                                                    ] = ff_pred16x16_vertical_10_mmxext;
158
                   h->pred16x16[HOR_PRED8x8
                                                   ] = ff_pred16x16_horizontal_10_mmxext;
159.
160.
                if (EXTERNAL SSE2(cpu flags)) {
                   h->pred4x4[DIAG_DOWN_LEFT_PRED ] = ff_pred4x4_down_left_10_sse2;
161.
162
                   h->pred4x4[DIAG\_DOWN\_RIGHT\_PRED] = ff\_pred4x4\_down\_right\_10\_sse2;
                    h->pred4x4[VERT LEFT PRED
163.
                                                   ] = ff_pred4x4_vertical_left_10_sse2;
164
                   h->pred4x4[VERT RIGHT PRED
                                                   ] = ff_pred4x4_vertical_right_10_sse2;
165.
                   h\text{->}pred4x4[HOR\_DOWN\_PRED
                                                   ] = ff_pred4x4_horizontal_down_10_sse2;
166
167
                    if (chroma format idc <= 1) {</pre>
168
                        h->pred8x8[DC_PRED8x8
                                                   ] = ff_pred8x8_dc_10_sse2;
169.
                        h->pred8x8[TOP_DC_PRED8x8 ] = ff_pred8x8_top_dc_10_sse2;
170
                        h->pred8x8[PLANE_PRED8x8
                                                   ] = ff_pred8x8_plane_10_sse2;
171
                        h->pred8x8[VERT PRED8x8
                                                   ] = ff_pred8x8_vertical_10_sse2;
172
                       h->pred8x8[HOR_PRED8x8
                                                   ] = ff_pred8x8_horizontal_10_sse2;
173
174
175
                   h->pred8x8l[VERT PRED
                                                     ] = ff pred8x8l vertical 10 sse2;
                                                    ] = ff_pred8x8l_horizontal_10_sse2;
176
                   h->pred8x8l[HOR PRED
177.
                   h->pred8x8l[DC PRED
                                                    l = ff pred8x8l dc 10 sse2:
                                                    ] = ff_pred8x8l_128_dc_10_sse2;
178.
                   h->pred8x8l[DC 128 PRED
179
                    h->pred8x8l[TOP DC PRED
                                                    ] = ff_pred8x8l_top_dc_10_sse2;
                   h->pred8x8l[DIAG\_DOWN\_LEFT\_PRED~]~=~ff\_pred8x8l\_down\_left\_10\_sse2;
180.
181
                    h->pred8x8l[DIAG_DOWN_RIGHT_PRED] = ff_pred8x8l_down_right_10_sse2;
182
                   h->pred8x8l[VERT_RIGHT_PRED
                                                   ] = ff_pred8x8l_vertical_right_10_sse2;
183
                    h->pred8x8l[HOR_UP_PRED
                                                    ] = ff_pred8x8l_horizontal_up_10_sse2;
184
185
                    h->pred16x16[DC_PRED8x8
                                                   ] = ff pred16x16 dc 10 sse2;
                    h->pred16x16[TOP DC PRED8x8 ] = ff pred16x16 top dc 10 sse2;
```

```
187.
                    h->pred16x16[DC 128 PRED8x8
                                                   ] = ff pred16x16 128 dc 10 sse2;
                   h->pred16x16[LEFT_DC_PRED8x8 ] = ff_pred16x16_left_dc_10_sse2;
188.
                    h->pred16x16[VERT PRED8x8
                                                   l = ff pred16x16 vertical 10 sse2:
189.
190.
                   h->pred16x16[HOR PRED8x8
                                                   ] = ff_pred16x16_horizontal_10_sse2;
191.
192
                  (EXTERNAL_SSSE3(cpu_flags)) {
193.
                    h->pred4x4[DIAG_DOWN_RIGHT_PRED] = ff_pred4x4_down_right_10_ssse3;
194.
                    h->pred4x4[VERT RIGHT PRED
                                                   ] = ff pred4x4 vertical right 10 ssse3;
195.
                    h->pred4x4[HOR_DOWN_PRED
                                                   ] = ff_pred4x4_horizontal_down_10_ssse3;
196
197.
                    h->pred8x8l[HOR PRED
                                                    ] = ff_pred8x8l_horizontal_10_ssse3;
                   h->pred8x8l[DIAG_DOWN_LEFT_PRED ] = ff_pred8x8l_down_left_10_ssse3;
198.
199.
                    h->pred8x8l[DIAG_DOWN_RIGHT_PRED] = ff_pred8x8l_down_right_10_ssse3;
200.
                   h->pred8x8l[VERT_RIGHT_PRED
                                                   ] = ff_pred8x8l_vertical_right_10_ssse3;
201.
                    h->pred8x8l[HOR UP PRED
                                                    ] = ff_pred8x8l_horizontal_up_10_ssse3;
202.
                if (EXTERNAL_AVX(cpu_flags)) {
203.
                   h->pred4x4[DIAG_DOWN_LEFT_PRED ] = ff_pred4x4_down_left_10_avx;
204.
                    h->pred4x4[DIAG_DOWN_RIGHT_PRED] = ff_pred4x4_down_right_10_avx;
205.
206.
                    h->pred4x4[VERT LEFT PRED
                                                   ] = ff_pred4x4_vertical_left_10_avx;
207.
                    h->pred4x4[VERT RIGHT PRED
                                                   ] = ff_pred4x4_vertical_right_10_avx;
208.
                    h->pred4x4[HOR_DOWN_PRED
                                                   ] = ff_pred4x4_horizontal_down_10_avx;
209.
210.
                    h->pred8x8l[VERT_PRED
                                                    ] = ff_pred8x8l_vertical_10_avx;
                                                      = ff_pred8x8l_horizontal_10_avx;
                    h->pred8x8l[HOR PRED
211.
212.
                    h->pred8x8l[DC_PRED
                                                    ] = ff_pred8x8l_dc_10_avx;
213.
                    h->pred8x8l[TOP_DC_PRED
                                                      = ff_pred8x8l_top_dc_10_avx;
214.
                    h->pred8x8l[DIAG_DOWN_RIGHT_PRED] = ff_pred8x8l_down_right_10_avx;
                    h->pred8x8l[DIAG DOWN LEFT PRED ] = ff pred8x8l down left 10 avx;
215.
216.
                   h->pred8x8l[VERT RIGHT PRED
                                                   ] = ff_pred8x8l_vertical_right_10_avx;
                   h->pred8x8l[HOR UP PRED
                                                    ] = ff_pred8x8l_horizontal_up_10_avx;
217.
218.
219.
220
```

从源代码可以看出,ff_h264_pred_init_x86()首先调用av_get_cpu_flags()获取标记CPU特性的cpu_flags,然后根据cpu_flags初始化不同的函数,包括{xxx}_mmx(),{xxx}_mmxext(),{xxx}_sse(),{xxx}_sse2(),{xxx}_sse3(),{xxx}_avx()几种采用不同会变指令的函数。

h264_decode_end()

h264_decode_end()用于关闭FFmpeg的H.264解码器。该函数的定义位于libavcodec\h264.c,如下所示。

```
[cpp] 📳 📑
      //关闭解码器
1.
      static av_cold int h264_decode_end(AVCodecContext *avctx)
2.
3.
4.
          H264Context *h = avctx->priv data;
          //移除参考帧
5.
6.
          ff_h264_remove_all_refs(h);
7.
          //释放H264Context
8.
          ff h264 free context(h);
9.
10.
          ff_h264_unref_picture(h, &h->cur_pic);
11.
12.
          return 0;
13.
      }
```

从函数定义中可以看出,h264_decode_end()调用了ff_h264_remove_all_refs()移除了所有的参考帧,然后又调用了ff_h264_free_context()释放了 H264Context里面的所有内存。下面看一下这两个函数的定义。

ff_h264_remove_all_refs()

ff h264 remove all refs()的定义如下所示。

```
[cpp] 📳 📑
      //移除参考帧
 2.
      void ff_h264_remove_all_refs(H264Context *h)
 3.
 4.
         int i;
 5.
         //循环16次
     //长期参考帧
 6.
         for (i = 0; i < 16; i++) {
 7.
     remove_long(h, i, 0);
8.
 9.
10.
     assert(h->long_ref_count == 0);
11.
         //短期参考帧
     for (i = 0; i < h->short_ref_count; i++) {
12.
13.
             unreference_pic(h, h->short_ref[i], 0);
            h->short_ref[i] = NULL;
14.
15.
16.
     h->short_ref_count = 0;
17.
18.
         memset(h->default_ref_list, 0, sizeof(h->default_ref_list)
19.
         memset(h->ref_list, 0, sizeof(h->ref_list));
20. }
```

从ff_h264_remove_all_refs()的定义中可以看出,该函数调用了remove_long()释放了长期参考帧,调用unreference_pic()释放了短期参考帧。

ff_h264_free_context()

ff_h264_free_context()的定义如下所示。

```
[cpp] 📳 👔
      //释放H264Context
 2.
      av_cold void ff_h264_free_context(H264Context *h)
 4.
 5.
          //释放各种内存
     ff_h264_free_tables(h, 1); // FIXME cleanup init stuff perhaps
 6.
          //释放SPS缓存
     for (i = 0; i < MAX_SPS_COUNT; i++)</pre>
 8.
             av_freep(h->sps_buffers + i);
 9.
10.
     //释放PPS缓存
11.
         for (i = 0; i < MAX_PPS_COUNT; i++)</pre>
     av_freep(h->pps_buffers + i);
12.
13. }
```

从 $ff_h^264_f^2$ ree_context()的定义可以看出,该函数调用了 $ff_h^264_f^2$ ree_tables()释放H264Context中的各种内存。可以看一下该函数的定义。

ff_h264_free_tables()

ff_h264_free_tables()的定义如下所示。

```
[cpp]
      //释放各种内存
2.
      void ff_h264_free_tables(H264Context *h, int free_rbsp)
3.
 4.
5.
          H264Context *hx;
6.
          av freep(&h->intra4x4 pred mode);
7.
      av freep(&h->chroma pred mode table);
8.
          av freep(&h->cbp table);
9.
     av_freep(&h->mvd_table[0]);
10.
11.
          av freep(&h->mvd table[1]);
      av_freep(&h->direct_table);
12.
13.
          av_freep(&h->non_zero_count);
14.
         av_freep(&h->slice_table_base);
15.
          h->slice table = NULL;
16.
      av_freep(&h->list_counts);
17.
18.
     av freep(&h->mb2b xy);
19.
          av_freep(&h->mb2br_xy);
20.
21.
          av_buffer_pool_uninit(&h->qscale_table_pool);
22.
      av buffer pool uninit(&h->mb type pool);
23.
          av_buffer_pool_uninit(&h->motion_val_pool);
24.
      av_buffer_pool_uninit(&h->ref_index_pool);
25.
26.
      if (free_rbsp && h->DPB) {
              for (i = 0; i < H264 MAX PICTURE COUNT; i++)</pre>
27.
28.
                ff_h264_unref_picture(h, &h->DPB[i]);
29.
              memset(h->delayed_pic, 0, sizeof(h->delayed_pic));
30.
              av_freep(&h->DPB);
31.
          } else if (h->DPB) {
32.
             for (i = 0; i < H264 MAX PICTURE COUNT; i++)</pre>
33.
                  h->DPB[i].needs_realloc = 1;
34.
35.
36.
      h->cur pic ptr = NULL;
37.
      for (i = 0; i < H264 MAX THREADS; i++) {</pre>
38.
39.
              hx = h->thread context[i];
              if (!hx)
40.
41.
                  continue:
42.
              av_freep(&hx->top_borders[1]);
43.
              av_freep(&hx->top_borders[0]);
44.
              av_freep(&hx->bipred_scratchpad);
45.
              av_freep(&hx->edge_emu_buffer);
46.
              av_freep(&hx->dc_val_base);
47.
              av_freep(&hx->er.mb_index2xy);
48.
              av_freep(&hx->er.error_status_table);
49.
              av_freep(&hx->er.er_temp_buffer);
              av_freep(&hx->er.mbintra_table);
50.
51.
              av freep(&hx->er.mbskip table);
52.
53.
              if (free rbsp) {
54.
                  av freep(&hx->rbsp buffer[1]):
                  av freep(&hx->rbsp_buffer[0]);
55.
56.
                  hx - rbsp\_buffer\_size[0] = 0;
57.
                  hx - rbsp\_buffer\_size[1] = 0;
58.
59.
              if (i)
60.
                  av_freep(&h->thread_context[i]);
61.
62.
```

可以看出ff_h264_free_tables()调用了av_freep()等函数释放了H264Context中的各个内存。

h264 decode frame()

h264_decode_frame()用于解码一帧图像数据。该函数的定义位于libavcodec\h264.c,如下所示。

```
[cpp] 📳 📑
1.
      //H.264解码器-解码
     static int h264_decode_frame(AVCodecContext *avctx, void *data,
3.
                                 int *got_frame, AVPacket *avpkt)
4.
5.
         //赋值。buf对应的就是AVPacket的data
      const uint8_t *buf = avpkt->data;
6.
7.
         int buf size
                           = avpkt->size:
         //指向AVCodecContext的priv data
8.
         H264Context *h
9.
                           = avctx->priv data;
         AVFrame *pict
10.
                           = data;
11.
         int buf index
                            = 0;
12.
         H264Picture *out;
13.
         int i, out_idx;
14.
         int ret;
15.
       h->flags = avctx->flags;
```

```
/st reset data partitioning here, to ensure GetBitContexts from previous
 17.
            * packets do not get used. */
 18.
 19.
           h->data partitioning = 0:
 20.
 21.
            /* end of stream, output what is still in the buffers */
 22.
          // Flush Decoder的时候会调用,此时输入为空的AVPacket≕
 23.
           if (buf_size == 0) {
 24.
        out:
 25.
 26.
               h->cur_pic_ptr = NULL;
               h->first_field = 0;
 27.
 28.
 29.
                // FIXME factorize this with the output code below
 30.
               //输出out,源自于h->delayed_pic[]
 31.
               //初始化
               out = h->delayed pic[0];
 32.
               out idx = 0;
 33.
                for (i = 1:
 34.
 35.
                     h->delayed pic[i] &&
 36.
                    !h->delayed_pic[i]->f.key_frame &&
 37.
                     !h->delayed_pic[i]->mmco_reset;
 38.
                    i++)
 39.
                    if (h->delayed_pic[i]->poc < out->poc) {
 40.
                    //输出out,源自于h->delayed_pic[]
 41.
                        //逐个处理
 42.
                       out = h->delayed_pic[i];
 43.
                       out_idx = i;
 44.
 45.
 46.
                for (i = out idx: h->delayed pic[i]: i++)
                   h->delayed pic[i] = h->delayed pic[i + 1];
 47.
 48.
 49.
                if (out) {
 50.
                   out->reference &= ~DELAYED PIC REF;
 51.
                    //输出
 52.
                    //out输出到pict
 53.
                    //即H264Picture到AVFrame
 54.
                    ret = output_frame(h, pict, out);
 55.
                    if (ret < 0)
 56.
                       return ret;
 57.
                    *got frame = 1;
 58.
 59.
 60.
               return buf index;
 61.
 62.
 63.
       if (h->is_avc && av_packet_get_side_data(avpkt, AV_PKT_DATA_NEW_EXTRADATA, NULL)) {
 64.
 65.
               int side size;
 66.
               uint8_t *side = av_packet_get_side_data(avpkt, AV_PKT_DATA_NEW_EXTRADATA, &side_size);
 67.
               if (is_extra(side, side_size))
 68.
               ff_h264_decode_extradata(h, side, side_size);
 69.
 70.
           if(h-sis\_avc \& buf\_size >= 9 \& buf[0] == 1 \& buf[2] == 0 \& (buf[4]\&0xFC) == 0xFC \& (buf[5]\&0x1F) \& buf[8] == 0x67) \{ (buf[8] == 0x67) \} 
 71.
               if (is_extra(buf, buf_size))
 72.
                return ff_h264_decode_extradata(h, buf, buf_size);
 73.
 74.
 75.
            //关键:解码NALU最主要的函数
 76.
 77.
           buf index = decode nal units(h, buf, buf size, 0);
 78.
 79.
           if (buf index < 0)</pre>
 80.
               return AVERROR_INVALIDDATA;
 81.
 82.
       if (!h->cur_pic_ptr && h->nal_unit_type == NAL_END_SEQUENCE) {
 83.
               av_assert0(buf_index <= buf_size);</pre>
 84.
               qoto out;
 85.
 86.
           if (!(avctx->flags2 & CODEC_FLAG2_CHUNKS) && !h->cur_pic_ptr) {
 87.
               if (avctx->skip frame >= AVDISCARD NONREF ||
 88.
                   buf size \Rightarrow= 4 && !memcmp("Q264", buf, 4))
 89.
                   return buf size;
 90.
               av_log(avctx, AV_LOG_ERROR, "no frame!\n");
 91.
               return AVERROR_INVALIDDATA;
 92.
 93.
 94.
 95.
            if (!(avctx->flags2 & CODEC FLAG2 CHUNKS) ||
                (h->mb_y >= h->mb_height && h->mb_height)) {
 96.
 97.
                if (avctx->flags2 & CODEC_FLAG2_CHUNKS)
 98.
                   decode_postinit(h, 1);
 99.
100.
               ff_h264_field_end(h, 0);
101.
102.
                /* Wait for second field. */
103.
                //设置got frame为0
                *got_frame = 0;
104.
105.
                if (h->next output pic && (
                                          h->next_output_pic->recovered)) {
106.
107.
                    if (!h->next output pic->recovered)
```

```
108.
                        h->next_output_pic->f.flags |= AV_FRAME_FLAG_CORRUPT;
109.
                    //输出Frame
110.
                    //即H264Picture到AVFrame
111.
                    ret = output_frame(h, pict, h->next_output_pic);
112.
                    if (ret < 0)
113.
                        return ret;
                    //设置got frame为1
114.
                    *got frame = 1;
115.
                    if (CONFIG MPEGVIDEO) {
116.
                        ff_print_debug_info2(h->avctx, pict, h->er.mbskip_table,
117.
118.
                                            h->next_output_pic->mb_type,
119.
                                             h->next_output_pic->qscale_table,
120.
                                             h->next_output_pic->motion_val,
121.
                                             &h->low_delay,
122.
                                             h->mb_width, h->mb_height, h->mb_stride, 1);
123.
124.
125.
126.
127.
            assert(pict->buf[0] || !*got_frame);
128.
129.
            return get consumed bytes(buf index, buf size);
130.
```

从源代码可以看出,h264_decode_frame()根据输入的AVPacket的data是否为空作不同的处理:

- (1) 若果输入的AVPacket的data为空,则调用output_frame()输出delayed_pic[]数组中的H264Picture,即输出解码器中缓存的帧(对应的是通常称为"Flush Decoder"的功能)。
- (2)若果输入的AVPacket的data不为空,则首先调用decode_nal_units()解码AVPacket的data,然后再调用output_frame()输出解码后的视频帧(有一点需要注意:由于帧重排等因素,输出的AVFrame并非对应于输入的AVPacket)。

下面看一下解码压缩编码数据时候用到的函数decode_nal_units()。

decode_nal_units()

decode nal units()是用于解码NALU的函数。函数定义位于libavcodec\h264.c,如下所示。

```
[cpp] 📳 📑
1.
      //解码NALU最主要的函数
      //h264 decode frame()中:
2.
      //buf一般是AVPacket->data
3.
      //buf size一般是AVPacket->size
4.
5.
      static int decode_nal_units(H264Context *h, const uint8_t *buf, int buf_size,
6.
                                int parse extradata)
7.
      AVCodecContext *const avctx = h->avctx;
8.
          H264Context *hx; ///< thread context
9.
10.
         int buf index:
11.
          unsigned context count:
12.
      int next avc;
13.
          int nals\_needed = 0; ///< number of NALs that need decoding before the next frame thread starts
14.
        int nal_index;
15.
          int idr_cleared=0;
16.
      int ret = 0;
17.
18.
      h->nal_unit_type= 0;
19.
20.
      if(!h->slice context count)
21.
               h->slice_context_count= 1;
      h->max contexts = h->slice context count;
22.
          if (!(avctx->flags2 & CODEC FLAG2 CHUNKS)) {
23.
24.
              h->current slice = 0:
25.
              if (!h->first field)
26.
                  h->cur_pic_ptr = NULL;
27.
              ff_h264_reset_sei(h);
28.
29.
30.
      //AVC1和H264的区别:
31.
          //AVC1 描述:H.264 bitstream without start codes.是不带起始码0x00000001的。FLV/MKV/MOV种的H.264属于这种
32.
         //H264 描述:H.264 bitstream with start codes.是带有起始码0x00000001的。H.264裸流,MPEGTS种的H.264属于这种
33.
         //通过VLC播放器,可以查看到具体的格式。打开视频后,通过菜单【工具】/【编解码信息】可以查看到【编解码器】具体格式,举例如下,编解码器信息:
34.
35.
          //编码: H264 - MPEG-4 AVC (part 10) (avc1)
36.
          //编码: H264 - MPEG-4 AVC (part 10) (h264)
37.
38.
          if (h->nal length size == 4) {
              if (buf size > 8 && AV RB32(buf) == 1 && AV RB32(buf+5) > (unsigned)buf size) {
39.
                  //前面4位是起始码0x00000001
40.
41.
                  h \rightarrow is avc = 0;
              \label{eq:control_entropy}  \mbox{\tt lelse if}(\mbox{\tt buf\_size} > 3 \&\& \mbox{\tt AV\_RB32(buf)} > 1 \&\& \mbox{\tt AV\_RB32(buf)} <= (\mbox{\tt unsigned}) \mbox{\tt buf\_size}) 
42
                  //前面4位是长度数据
43.
44.
                  h \rightarrow is_avc = 1;
45.
46.
47.
          if (avctx->active_thread_type & FF_THREAD_FRAME)
48.
              nals_needed = get_last_needed_nal(h, buf, buf_size);
49.
```

```
51.
               buf index
                             = 0:
 52.
               context count = 0;
 53.
                next_avc = h->is_avc ? 0 : buf_size;
 54.
                nal_index
                             = 0:
 55.
                for (;;) {
 56.
                   int consumed;
 57.
                    int dst_length;
 58.
                    int bit_length;
 59.
                    const uint8_t *ptr;
                    int nalsize = 0;
 60.
 61.
                    int err;
 62.
 63.
                    if (buf index >= next avc) {
                       nalsize = get_avc_nalsize(h, buf, buf_size, &buf_index);
 64.
 65.
                        if (nalsize < 0)</pre>
 66.
                          break;
                        next_avc = buf_index + nalsize;
 67.
 68.
                    } else {
 69.
                        buf_index = find_start_code(buf, buf_size, buf_index, next_avc);
 70.
                        if (buf_index >= buf_size)
 71.
                           break;
 72.
                        if (buf_index >= next_avc)
 73.
                           continue;
 74.
 75.
 76.
                   hx = h->thread context[context count];
 77.
                    //解析得到NAL(获得nal unit type等信息)
                   ptr = ff_h264_decode_nal(hx, buf + buf_index, &dst_length,
 78.
 79.
                                            &consumed, next_avc - buf_index);
 80.
                    if (!ptr || dst_length < 0) {</pre>
 81.
                        ret = -1:
 82.
                        qoto end;
 83.
 84.
 85.
                    bit_length = get_bit_length(h, buf, ptr, dst_length,
 86.
                                buf_index + consumed, next_avc);
 87.
                    if (h->avctx->debug & FF_DEBUG_STARTCODE)
 88.
 89.
                       av_log(h->avctx, AV_LOG_DEBUG,
 90.
                               "NAL %d/%d at %d/%d length %d\n",
 91.
                               hx->nal unit type, hx->nal ref idc, buf index, buf size, dst length):
 92.
 93.
                    if (h->is avc && (nalsize != consumed) && nalsize)
                    av_log(h->avctx, AV_LOG_DEBUG,
 94.
 95.
                               "AVC: Consumed only %d bytes instead of %d\n",
 96
                              consumed. nalsize):
 97.
 98
                   buf index += consumed;
 99.
                    nal_index++;
100.
101.
                    if (avctx->skip_frame >= AVDISCARD_NONREF &&
102.
                        h->nal_ref_idc == 0 &&
103.
                        h->nal_unit_type != NAL_SEI)
104.
                       continue;
105.
106.
       again:
107.
                   if ( !(avctx->active_thread_type & FF_THREAD_FRAME)
108.
                        || nals_needed >= nal_index)
109.
                        h \rightarrow au pps id = -1;
                     * Ignore per frame NAL unit type during extradata
110.
111.
                     * parsing. Decoding slices is not possible in codec init
                    * with frame-mt */
112.
113.
                    if (parse_extradata) {
114.
                        switch (hx->nal_unit_type) {
115.
                        case NAL IDR SLICE:
116.
                        case NAL_SLICE:
                        case NAL_DPA:
117.
                        case NAL_DPB:
118.
119.
                        case NAL DPC:
120.
                        av_log(h->avctx, AV_LOG_WARNING,
                                   "Ignoring NAL %d in global header/extradata\n",
121.
122.
                                  hx->nal unit type);
                           // fall through to next case
123.
                        case NAL AUXILIARY SLICE:
124.
125.
                           hx->nal_unit_type = NAL_FF_IGNORE;
126.
127
                   }
128.
129.
                    err = 0:
130.
                    //根据不同的 NALU Type,调用不同的函数
131.
                    switch (hx->nal_unit_type) {
132.
                    //IDR帧
                    case NAL_IDR_SLICE:
133.
134.
                       if ((ptr[0] & 0xFC) == 0x98) {
135.
                           av_log(h->avctx, AV_LOG_ERROR, "Invalid inter IDR frame\n");
                           h->next_outputed_poc = INT_MIN;
136.
137.
                           ret = -1:
                           goto end:
138.
139.
140.
                        if (h->nal_unit_type != NAL_IDR_SLICE) {
141
                            av log(h->avcty AV LOG FRROR
```

```
142.
                                 "Invalid mix of idr and non-idr slices\n");
143.
                           ret = -1:
144.
                           goto end;
145.
146.
                       if(!idr cleared)
                           idr(h); // FIXME ensure we don't lose some frames if there is reordering
147.
148.
                       idr cleared = 1;
149.
                       h->has_recovery_point = 1;
                       //注意没有break
150.
151.
                   case NAL SLICE:
152.
                      init_get_bits(&hx->gb, ptr, bit_length);
                       hx->intra_gb_ptr =
hx->inter_gb_ptr = &hx->gb;
153.
154.
                       hx->data_partitioning = 0;
155.
156
                       //解码Slice Header
                       if ((err = ff_h264_decode_slice_header(hx, h)))
157.
158.
                         break:
159.
160.
                       if (h->sei_recovery_frame_cnt >= 0) {
                           if (h->frame_num != h->sei_recovery_frame_cnt || hx->slice_type_nos != AV_PICTURE_TYPE_I)
161.
                            h->valid_recovery_point = 1;
162.
163.
                           if ( h->recovery frame < 0</pre>
164.
165.
                               >sei recovery frame cnt) {
166
                             h->recovery_frame = (h->frame_num + h->sei_recovery_frame_cnt) &
167.
                                                  ((1 << h->sps.log2_max_frame_num) - 1);
168.
169.
                               if (!h->valid_recovery_point)
170.
                               h->recovery_frame = h->frame_num;
171.
172.
173.
174.
                       h->cur pic ptr->f.key frame |=
175.
                           (hx->nal unit type == NAL IDR SLICE);
176.
                       if (hx->nal unit type == NAL IDR SLICE ||
177.
                         h->recovery_frame == h->frame_num) {
h->recovery_frame = -1;
178.
179
180.
                           h->cur_pic_ptr->recovered = 1;
181.
182.
                       // If we have an IDR, all frames after it in decoded order are
183.
                       if (hx->nal_unit_type == NAL_IDR_SLICE)
184.
185.
                           h->frame_recovered |= FRAME_RECOVERED_IDR;
                       h->frame recovered |= 3*!!(avctx->flags2 & CODEC FLAG2 SHOW ALL);
186.
187.
                       h->frame_recovered |= 3*!!(avctx->flags & CODEC_FLAG_OUTPUT_CORRUPT);
188.
189.
                       h->cur pic ptr->recovered |= h->frame recovered;
190.
                       h->cur_pic_ptr->recovered |= !!(h->frame_recovered & FRAME RECOVERED IDR);
191.
192.
193.
194.
                       if (h->current slice == 1) {
195.
                           if (!(avctx->flags2 & CODEC FLAG2 CHUNKS))
196
                               decode_postinit(h, nal_index >= nals_needed);
197.
198
                           if (h->avctx->hwaccel &&
199.
                               (ret = h->avctx->hwaccel->start_frame(h->avctx, NULL, 0)) < 0)</pre>
200.
201.
                           if (CONFIG_H264_VDPAU_DECODER &&
202.
                              h->avctx->codec->capabilities & CODEC_CAP_HWACCEL_VDPAU)
203.
                               ff_vdpau_h264_picture_start(h);
204.
205.
                       if (hx->redundant pic count == 0) {
206.
207.
                           if (avctx->hwaccel) {
                              ret = avctx->hwaccel->decode_slice(avctx,
208.
209
                                                                 &buf[buf index - consumed],
210.
                                                                 consumed);
211.
                               if (ret < 0)
212.
                                  return ret;
213.
                           } else if (CONFIG_H264_VDPAU_DECODER &&
214.
                                    h->avctx->codec->capabilities & CODEC_CAP_HWACCEL_VDPAU)
215.
                               ff_vdpau_add_data_chunk(h->cur_pic_ptr->f.data[0],
216.
                                                   start_code,
217.
                                                       sizeof(start code));
218.
                               ff_vdpau_add_data_chunk(h->cur_pic_ptr->f.data[0],
                                                      &buf[buf index - consumed],
219.
220.
                                                      consumed):
221.
                           } else
222.
                              context count++;
223.
224.
                       break:
225
                   case NAL DPA:
226
                      if (h->avctx->flags & CODEC_FLAG2_CHUNKS) {
227.
                           av_log(h->avctx, AV_LOG_ERROR,
228.
                                  "Decoding in chunks is not supported for "
229.
                                  "partitioned slices.\n");
230.
                           return AVERROR(ENOSYS);
```

```
232.
233.
                        init_get_bits(&hx->gb, ptr, bit_length);
234.
                        hx - sintra_gb_ptr =
235
                        hx - sinter_gb_ptr = NULL;
236.
                        //解码Slice Header
237.
                        if ((err = ff_h264_decode_slice_header(hx, h))) {
238.
                           /* make sure data_partitioning is cleared if it was set
239.
                             * before, so we don't try decoding a slice without a valid
                            * slice header later */
240.
241.
                            h->data_partitioning = 0;
242.
                            break:
243.
                        }
244.
245.
                        hx->data partitioning = 1;
246.
                        break:
247.
                    case NAI DPR:
248.
                        init_get_bits(&hx->intra_gb, ptr, bit_length);
249
                        hx->intra_gb_ptr = &hx->intra_gb;
250
                        break:
251.
                    case NAL_DPC:
252
                        init_get_bits(&hx->inter_gb, ptr, bit_length);
                        hx->inter_gb_ptr = &hx->inter_gb;
253.
254.
255.
                        av_log(h->avctx, AV_LOG_ERROR, "Partitioned H.264 support is incomplete\n");
256.
257.
258.
                        if (hx->redundant pic count == 0 &&
259.
                            hx->intra qb ptr &&
260.
                            hx->data partitioning &&
                            h->cur pic ptr && h->context initialized &&
261.
                            (avctx->skip_frame < AVDISCARD_NONREF || hx->nal_ref_idc) &&
262.
263.
                            (avctx->skip frame < AVDISCARD BIDIR ||
264.
                             hx->slice_type_nos != AV_PICTURE_TYPE_B) &&
265.
                            (avctx->skip_frame < AVDISCARD_NONINTRA ||
266.
                             hx->slice_type_nos == AV_PICTURE_TYPE_I) &&
267.
                            avctx->skip_frame < AVDISCARD_ALL)</pre>
268.
                            context count++;
269.
270.
                    case NAL_SEI:
271.
                        init get bits(&h->gb, ptr, bit length);
272.
                        //解析SEI补充增强信息单元
273.
                        ret = ff h264 decode sei(h);
274.
                        if (ret < 0 && (h->avctx->err_recognition & AV_EF_EXPLODE))
275.
                            goto end:
276
                        hreak:
277.
                    case NAL SPS:
278
                        init_get_bits(&h->gb, ptr, bit_length);
                        //解析SPS序列参数集
279.
280
                        if (ff_h264_decode_seq_parameter_set(h) < 0 && (h->is_avc ? nalsize : 1)) {
281.
                            av_log(h->avctx, AV_LOG_DEBUG,
282.
                                   "SPS decoding failure, trying again with the complete NAL\n");
283.
                            if (h->is_avc)
284.
                                av_assert0(next_avc - buf_index + consumed == nalsize);
                            if ((next_avc - buf_index + consumed - 1) >= INT_MAX/8)
285.
286.
                                break;
287.
                            init_get_bits(&h->gb, &buf[buf_index + 1 - consumed],
                                         8*(next avc - buf index + consumed - 1));
288.
289.
                            ff h264_decode_seq_parameter_set(h);
290.
291.
292.
                        break;
293
294.
                    case NAL PPS:
295
                        init_get_bits(&h->gb, ptr, bit_length);
                        //解析PPS图像参数集
296.
297.
                        ret = ff_h264_decode_picture_parameter_set(h, bit_length);
298.
                        if (ret < 0 && (h->avctx->err_recognition & AV_EF_EXPLODE))
                            goto end;
299.
300.
                        break;
301.
                    case NAL_AUD:
302.
                    case NAL END SEQUENCE:
                    case NAL END STREAM:
303.
                    case NAL FILLER DATA:
304.
                    case NAL SPS EXT:
305.
306.
                    case NAL AUXILIARY SLICE:
307
                        break:
308.
                    case NAL_FF_IGNORE:
                        break;
309
310.
                    default:
311.
                        av\_log(avctx,\ AV\_LOG\_DEBUG,\ "Unknown\ NAL\ code:\ %d\ (%d\ bits)\n",
312.
                              hx->nal_unit_type, bit_length);
313.
314.
315.
                    if (context_count == h->max_contexts) {
                        ret = ff h264 execute decode slices(h, context count);
316.
                        if (ret < 0 && (h->avctx->err_recognition & AV_EF_EXPLODE))
317.
                            goto end:
318.
319.
                        context count = 0;
320.
321.
                    if (err < 0 || err == SLICE_SKIPED) {</pre>
322.
```

```
323.
                        if (err < 0)
324.
                            av_log(h->avctx, AV_LOG_ERROR, "decode_slice_header error\n");
                        h->ref count[0] = h->ref count[1] = h->list count = 0;
325.
                      else if (err == SLICE_SINGLETHREAD) {
326.
                        /* Slice could not be decoded in parallel mode, copy down
327.
                         * NAL unit stuff to context 0 and restart. Note that
328.
329.
                         * rbsp buffer is not transferred, but since we no longer
                         ^{*} run in parallel mode this should not be an issue. ^{*}/
330.
331.
                        h->nal_unit_type = hx->nal_unit_type;
332
                        h->nal_ref_idc = hx->nal_ref_idc;
333.
                                          = h;
334.
                        goto again;
335.
336.
337.
338.
            if (context_count) {
339.
                //真正的解码
340.
                ret = ff_h264_execute_decode_slices(h, context_count);
341.
                if (ret < 0 && (h->avctx->err_recognition & AV_EF_EXPLODE))
342.
                   goto end:
343.
344.
345.
            ret = 0;
       end:
346.
            /* clean up */
347.
            if (h->cur_pic_ptr && !h->droppable) {
348.
349.
                ff\_thread\_report\_progress(\&h->cur\_pic\_ptr->tf,\ INT\_MAX,
350.
                                        h->picture_structure == PICT_BOTTOM_FIELD)
351.
352.
353.
            return (ret < 0) ? ret : buf_index;</pre>
354.
```

从源代码可以看出,decode_nal_units()首先调用ff_h264_decode_nal()判断NALU的类型,然后根据NALU类型的不同调用了不同的处理函数。这些处理函数可以分为两类——解析函数和解码函数,如下所示。

(1) 解析函数 (获取信息):

ff_h264_decode_seq_parameter_set():解析SPS。
ff_h264_decode_picture_parameter_set():解析PPS。
ff_h264_decode_sei():解析SEI。
ff_h264_decode_slice_header():解析Slice Header。

(2) 解码函数(解码得到图像):

ff_h264_execute_decode_slices():解码Slice。

其中解析函数在文章《 FFmpeg的H.264解码器源代码简单分析:解析器(Parser)部分 》部分已经有过介绍,就不再重复叙述了。解码函数ff_h264_execute_decode _slices()完成了解码Slice的工作,下面看一下该函数的定义。

ff_h264_execute_decode_slices()

ff_h264_execute_decode_slices()用于解码获取图像信息,定义位于libavcodec\h264_slice.c,如下所示。

```
[cpp] 📳 📑
2.
      * Call decode_slice() for each context
3.
 4.
      * @param h h264 master context
5.
       * @param context_count number of contexts to execute
6.
      //真正的解码
7.
     int ff h264 execute decode slices(H264Context *h, unsigned context count)
8.
9.
10.
          AVCodecContext *const avctx = h->avctx:
11.
          H264Context *hx:
12.
         int i:
13.
14.
     av_assert0(h->mb_y < h->mb_height);
15.
16.
     if (h->avctx->hwaccel ||
17.
              h->avctx->codec->capabilities & CODEC_CAP_HWACCEL_VDPAU)
18.
              return 0;
19.
          //context count的数量
20.
      if (context_count == 1) {
21.
              //解码Slice
22.
              return decode slice(avctx, &h);
23.
          } else {
24.
          av assert0(context count > 0);
25.
              for (i = 1; i < context count; i++) {</pre>
                                    = h->thread context[i]
26.
              hx
                  if (CONFIG ERROR RESILIENCE) {
27.
28.
                  hx->er.error_count = 0;
29.
                 }
30.
                 hx->x264_build = h->x264_build;
31.
             }
32.
33.
              avctx->execute(avctx, decode_slice, h->thread_context,
34.
             NULL, context_count, sizeof(void *));
35.
36.
             /* pull back stuff from slices to master context */
37.
                                 = h->thread context[context count - 1];
              hx
                                 = hx -> mb x;
38.
             h->mb x
                                  = hx -> mb y;
39.
              h->mb y
             h->droppable = hx->droppable;
40.
              h->picture_structure = hx->picture_structure;
41.
42.
              if (CONFIG_ERROR_RESILIENCE) {
43.
                  for (i = 1; i < context_count; i++)</pre>
44.
                   h->er.error_count += h->thread_context[i]->er.error_count;
45.
46.
47.
48.
         return 0;
49.
```

可以看出ff_h264_execute_decode_slices()调用了decode_slice()函数。在decode_slice()函数中完成了熵解码,宏块解码,环路滤波,错误隐藏等解码的细节工作。由于decode_slice()的内容比较多,本文暂不详细分析该函数,仅简单看一下该函数的定义。

decode_slice()

decode slice()完成了熵解码,宏块解码,环路滤波,错误隐藏等解码的细节工作。该函数的定义位于定义位于libavcodec\h264 slice.c,如下所示。

```
[cpp] 📳 🔝
 1.
                 //解码slice
                 //三个主要步骤:
 3.
                 //1. 熵解码(CAVLC/CABAC)
                 //2.宏块解码
 4.
 5.
                 //3.环路滤波
 6.
                 //此外还包含了错误隐藏代码
                 static int decode slice(struct AVCodecContext *avctx, void *arg)
 7.
 8.
                {
                             H264Context *h = *(void **)arg:
 9.
10.
                 int lf_x_start = h->mb_x;
11.
12.
                 h->mb_skip_run = -1;
13.
                            av\_assert\theta(h->block\_offset[15] == (4 * ((scan8[15] - scan8[0]) \& 7) << h->pixel\_shift) + 4 * h->pixel\_shift)
14.
                 >linesize * ((scan8[15] - scan8[0]) >> 3));
15.
16.
                  h->is_complex = FRAME_MBAFF(h) || h->picture_structure != PICT_FRAME |
17.
                                                                            avctx->codec_id != AV_CODEC_ID_H264 ||
                                                                           (CONFIG_GRAY && (h->flags & CODEC_FLAG_GRAY));
18.
19.
20.
                             if (!(h->avctx->active thread type & FF THREAD SLICE) && h->picture structure == PICT FRAME && h->er.error status table) {
21.
                                        const int start_i = av_clip(h->resync_mb_x + h->resync_mb_y * h->mb_width, 0, h->mb_num - 1);
22.
                                        if (start i) {
23.
                                                    int prev status = h->er.error status table[h->er.mb index2xv[start i - 1]];
                                                    prev status &= ~ VP START;
24.
                                                    if (prev_status != (ER_MV_END | ER_DC_END | ER_AC_END))
25.
26.
                                                         h->er.error_occurred = 1;
```

```
28.
 29.
            //CABAC情况
 30.
           if (h->pps.cabac) {
 31.
                /* realign */
 32.
               align_get_bits(&h->gb);
 33.
               /* init cabac */
 34.
 35.
                //初始化CABAC解码器
                {\tt ff\_init\_cabac\_decoder(\&h->cabac,}
 36.
 37.
                                      h->gb.buffer + get\_bits\_count(\&h->gb) / 8,
 38.
                                      (get_bits_left(\&h->gb) + 7) / 8);
 39.
 40.
                ff_h264_init_cabac_states(h);
 41.
                //循环处理每个宏块
 42.
                for (;;) {
 43.
                    // START TIMER
 44.
                    //解码CABAC数据
 45.
                    int ret = ff h264 decode mb cabac(h);
 46.
                    int eos;
                    // STOP TIMER("decode mb cabac")
 47.
                    //解码宏块
 48.
                    if (ret >= 0)
 49.
                     ff_h264_hl_decode_mb(h);
 50.
 51.
 52.
                    // FIXME optimal? or let mb_decode decode 16x32 ?
 53.
                    //宏块级帧场自适应。很少接触
 54.
                    if (ret >= 0 && FRAME_MBAFF(h)) {
 55.
                        h->mb_y++;
 56.
 57.
                        ret = ff_h264_decode_mb_cabac(h);
                        //解码宏块
 58.
 59.
                        if (ret >= 0)
 60.
                           ff_h264_hl_decode_mb(h);
 61.
                        h->mb_y--;
 62.
                    eos = get cabac terminate(&h->cabac):
 63.
 64.
 65.
                    if ((h->workaround_bugs & FF_BUG_TRUNCATED) &&
 66.
                        h	ext{->cabac.bytestream} 	ext{ > } h	ext{->cabac.bytestream\_end + 2) } \{
 67.
                        //错误隐藏
 68.
                        er_add_slice(h, h->resync_mb_x, h->resync_mb_y, h->mb_x -
 69.
                                     h->mb_y, ER_MB_END);
 70.
                        if (h->mb_x >= lf_x_start)
 71.
                            loop_filter(h, lf_x_start, h->mb_x + 1);
 72.
                        return 0:
 73.
 74.
                    if (h->cabac.bytestream > h->cabac.bytestream_end + 2 )
                        av_log(h->avctx, AV_LOG_DEBUG, "bytestream overread %"PTRDIFF_SPECIFIER"\n", h->cabac.bytestream_end - h->cabac.byte
 75.
       stream);
 76.
                    if (ret < 0 || h->cabac.bytestream > h->cabac.bytestream end + 4) {
                        av log(h->avctx, AV LOG ERROR,
 77.
 78.
                               "error while decoding MB %d %d, bytestream %"PTRDIFF_SPECIFIER"\n",
 79.
                               h \rightarrow mb x, h \rightarrow mb y,
                               h->cabac.bytestream_end - h->cabac.bytestream);
 80.
 81.
                        er_add_slice(h, h->resync_mb_x, h->resync_mb_y, h->mb_x,
 82.
                                    h->mb_y, ER_MB_ERROR);
 83.
                        return AVERROR_INVALIDDATA;
 84.
 85.
                    //mb x自增
 86.
                    //如果自增后超过了一行的mb个数
 87.
                    if (++h->mb_x >= h->mb_width) {
 88.
                        //环路滤波
 89.
                        loop filter(h, lf x start, h->mb x);
                        h \rightarrow mb x = lf x start = 0;
 90.
 91.
                        decode finish row(h);
                        //mb_y自增(处理下一行)
 92.
 93.
                        ++h->mb_y;
                        //宏块级帧场自适应,暂不考虑
 94.
 95.
                        if (FIELD_OR_MBAFF_PICTURE(h)) {
 96.
                            ++h->mb_y;
 97.
                            if (FRAME_MBAFF(h) && h->mb_y < h->mb_height)
 98.
                               predict_field_decoding_flag(h);
 99.
                        }
100.
                    //如果mb_y超过了mb的行数
101.
102.
                    if (eos || h->mb_y >= h->mb_height) {
                        tprintf(h->avctx, "slice end %d %d\n",
103.
                              get_bits_count(&h->gb), h->gb.size_in_bits);
104.
                        er_add_slice(h, h->resync_mb_x, h->resync_mb_y, h->mb_x - 1,
105.
                                  h->mb_y, ER_MB_END);
106.
                        if (h->mb_x > lf_x_start)
107.
108.
                           loop_filter(h, lf_x_start, h->mb_x);
109.
                        return 0:
110.
111.
112.
           } else {
113.
               //CAVLC情况
114.
                //循环处理每个宏块
115.
                for (;;) {
                   //解码宏块的CAVLC
116.
                    int ret = ff h264 decode mb cavlc(h);
```

```
//解码宏块
118.
119
                     if (ret >= 0)
120.
                        ff_h264_hl_decode_mb(h);
121.
122.
                     // FIXME optimal? or let mb_decode decode 16x32
123
                     if (ret >= 0 && FRAME_MBAFF(h)) {
                        h->mb_y++;
124.
125.
                         ret = ff_h264_decode_mb_cavlc(h);
126.
127.
                        if (ret >= 0)
                           ff_h264_hl_decode_mb(h);
128.
129.
                        h->mb y--;
130.
131.
132.
                     if (ret < 0) {</pre>
133.
                        av_log(h->avctx, AV_LOG_ERROR,
134.
                                "error while decoding MB %d %d\n", h->mb_x, h->mb_y)
135.
                        er_add_slice(h, h->resync_mb_x, h->resync_mb_y, h->mb_x,
136
                                     h->mb_y, ER_MB_ERROR);
137.
138
139.
140.
                        (++h->mb_x >= h->mb_width) {
141.
142.
                         loop_filter(h, lf_x_start, h->mb_x);
143.
                         h->mb x = lf x start = 0;
                        decode_finish_row(h);
144.
145.
                         ++h->mb y;
146.
                        if (FIELD OR MBAFF PICTURE(h)) {
147.
                             ++h->mb v:
                             if (FRAME MBAFF(h) && h->mb y < h->mb height)
148
149.
                                 predict_field_decoding_flag(h);
150.
                         if (h->mb_y >= h->mb_height) {
151
152
                             tprintf(h->avctx, "slice end %d %d\n",
153.
                                     get_bits_count(&h->gb), h->gb.size_in_bits);
154.
155.
                                   get_bits_left(&h->gb) == 0
                                || get_bits_left(&h->gb) > 0 && !(h->avctx->err_recognition & AV_EF_AGGRESSIVE))
156.
157.
                                 //错误隐藏
158.
                                 er add slice(h, h->resync mb x, h->resync mb y,
159.
                                              h \rightarrow mb_x - 1, h \rightarrow mb_y, ER_MB_END);
160.
161.
                                 return 0:
162
                               else {
163.
                                 er_add_slice(h, h->resync_mb_x, h->resync_mb_y,
164
                                              h->mb_x, h->mb_y, ER_MB_END);
165.
166
                                 return AVERROR_INVALIDDATA;
167.
168.
169.
                    }
170.
171.
                     if (get_bits_left(&h->gb) <= 0 && h->mb_skip_run <= 0) {</pre>
                        tprintf(h->avctx, "slice end %d %d\n",
172.
173.
                                 get bits count(&h->gb), h->gb.size in bits);
174.
175.
                         if (get bits left(\&h->gb) == 0) {
176.
                             er_add_slice(h, h->resync_mb_x, h->resync_mb_y,
177.
                                          h \rightarrow mb_x - 1, h \rightarrow mb_y, ER_MB_END);
178.
                             if (h->mb_x > lf_x_start)
179
                                 loop_filter(h, lf_x_start, h->mb_x);
180.
181
                             return 0:
182.
                           else {
183.
                             er_add_slice(h, h->resync_mb_x, h->resync_mb_y, h->mb_x,
184.
                                     h->mb_y, ER_MB_ERROR);
185.
186.
                             return AVERROR INVALIDDATA;
187.
                        }
188.
189.
190.
191.
```

从源代码可以看出,decode_slice()按照宏块(16x16)的方式处理输入的视频流。每个宏块的压缩数据经过以下3个基本步骤的处理,得到解码后的数据:

- (1) 熵解码。如果熵编码为CABAC,则调用ff_h264_decode_mb_cabac();如果熵编码为CAVLC,则调用ff_h264_decode_mb_cavlc()
- (2) 宏块解码。这一步骤调用ff_h264_hl_decode_mb()
- (3) 环路滤波。这一步骤调用loop filter()

此外,还有可能调用错误隐藏函数er_add_slice()。

至此,decode_nal_units()函数的调用流程就基本分析完毕了。h264_decode_frame()在调用完decode_nal_units()之后,还需要把解码后得到的H264Picture转换为AVF rame输出出来,这时候会调用一个相对比较简单的函数output_frame()。

output_frame()用于将一个H264Picture结构体转换为一个AVFrame结构体。该函数的定义位于libavcodec\h264.c,如下所示。

```
[cpp] 📳 📑
1.
      //Flush Decoder的时候用到
2.
     //srcp输出到dst
3.
      //即H264Picture到AVFrame
4.
     static int output_frame(H264Context *h, AVFrame *dst, H264Picture *srcp)
5.
6.
        //src即H264Picture中的f
         AVFrame *src = &srcp->f;
8.
     const AVPixFmtDescriptor *desc = av_pix_fmt_desc_get(src->format);
9.
         int i;
     int ret = av frame ref(dst, src);
10.
11.
         if (ret < 0)
     return ret;
12.
13.
     av\_dict\_set(\&dst->metadata, \ "stereo\_mode", \ ff\_h264\_sei\_stereo\_mode(h), \ \theta);
14.
15.
     if (srcp->sei_recovery_frame_cnt == 0)
16.
17.
             dst->key_frame = 1;
     if (!srcp->crop)
18.
19.
              return 0;
20.
21.
          for (i = 0; i < desc->nb_components; i++) {
22.
             int hshift = (i > 0) ? desc->log2_chroma_w : 0;
23.
              int vshift = (i > 0) ? desc->log2_chroma_h : 0;
             int off = ((srcp->crop_left >> hshift) << h->pixel_shift) +
24.
                           (srcp->crop_top >> vshift) * dst->linesize[i];
25.
26.
             dst->data[i] += off;
27.
28.
         return 0:
29.
```

从源代码中可以看出,output_frame()实际上就是把H264Picture结构体中的"f"(AVFrame结构体)输出了出来。

至此,H.264解码器的主干部分的源代码就分析完毕了。

雷霉骅

leixiaohua1020@126.com

http://blog.csdn.net/leixiaohua1020

版权声明:本文为博主原创文章,未经博主允许不得转载。 https://blog.csdn.net/leixiaohua1020/article/details/45042755

文章标签: FFmpeg 解码器 NALU 初始化 源代码

个人分类: FFMPEG 所属专栏: FFmpeg

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com