

## 原 x264源代码简单分析：x264命令行工具（x264.exe）

2015年05月08日 18:30:23 阅读数：11985

=====

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](#)

=====

本文简单分析x264项目中的命令行工具（x264.exe）的源代码。该命令行工具可以调用libx264将YUV格式像素数据编码为H.264码流。

## 函数调用关系图

X264命令行工具的源代码在x264中的位置如下图所示。



X264命令行工具的源代码的调用关系如下图所示。



从图中可以看出，X264命令行工具调用了libx264的几个API完成了H.264编码工作。使用libx264的API进行编码可以参考《[最简单的视频编码器：基于libx264（编码YUV为H.264）](#)》，这个流程中最关键的API包括：

`x264_param_default()`：设置参数集结构体x264\_param\_t的缺省值。

`x264_encoder_open()`：打开编码器。

x264\_encoder\_headers()：输出SPS，PPS，SEI等信息。

x264\_encoder\_encode()：编码输出一帧图像。

x264\_encoder\_close()：关闭编码器。

在X264命令行工具中，main()首先调用parse()解析输入的命令行参数，然后调用encode()进行编码。parse()首先调用x264\_param\_default()为存储参数的结构体x264\_param\_t赋默认值；然后在一个大循环中调用getopt\_long()逐个解析输入的参数，并作相应的处理；最后调用select\_input()和select\_output()解析输入文件格式（例如yuv，y4m...）和输出文件格式（例如raw，flv，MP4...）。encode()首先调用x264\_encoder\_open()打开H.264编码器，然后调用x264\_encoder\_headers()输出H.264码流的头信息（例如SPS、PPS、SEI），接着进入一个循环并且调用encode\_frame()逐帧编码视频，最后调用x264\_encoder\_close()关闭解码器。其中encode\_frame()中又调用了x264\_encoder\_encode()完成了具体的编码工作。下文将会对上述流程展开分析。

## main()

main()是x264控制台程序的入口函数，定义如下所示。

```
[cpp]
1. //主函数
2. int main( int argc, char **argv )
3. {
4.     //参数集
5.     x264_param_t param;
6.     cli_opt_t opt = {0};
7.     int ret = 0;
8.
9.     FAIL_IF_ERROR( x264_threading_init(), "unable to initialize threading\n" )
10.
11. #ifdef WIN32
12.     FAIL_IF_ERROR( !get_argv_utf8( &argc, &argv ), "unable to convert command line to UTF-8\n" )
13.
14.     GetConsoleTitleW( org_console_title, CONSOLE_TITLE_SIZE );
15.     _setmode( _fileno( stdin ), _O_BINARY );
16.     _setmode( _fileno( stdout ), _O_BINARY );
17.     _setmode( _fileno( stderr ), _O_BINARY );
18. #endif
19.
20.     /* Parse command line */
21.     //解析命令行输入
22.     if( parse( argc, argv, ¶m, &opt ) < 0 )
23.         ret = -1;
24.
25. #ifdef WIN32
26.     /* Restore title; it can be changed by input modules */
27.     SetConsoleTitleW( org_console_title );
28. #endif
29.
30.     /* Control-C handler */
31.     signal( SIGINT, sigint_handler );
32.     //编码
33.     if( !ret )
34.         ret = encode( ¶m, &opt );
35.
36.     /* clean up handles */
37.     if( filter.free )
38.         filter.free( opt.hin );
39.     else if( opt.hin )
40.         cli_input.close_file( opt.hin );
41.     if( opt.hout )
42.         cli_output.close_file( opt.hout, 0, 0 );
43.     if( opt.tcfile_out )
44.         fclose( opt.tcfile_out );
45.     if( opt.qpfile )
46.         fclose( opt.qpfile );
47.
48. #ifdef WIN32
49.     SetConsoleTitleW( org_console_title );
50.     free( argv );
51. #endif
52.
53.     return ret;
54. }
```

可以看出main()的定义很简单，它主要调用了两个函数：parse()和encode()。main()首先调用parse()解析输入的命令行参数，然后调用encode()进行编码。下面分别分析这两个函数。

## parse()

parse()用于解析命令行输入的参数（存储于argv[]中）。它的定义如下所示。

```
[cpp]
1. //解析命令行输入
2. static int parse( int argc, char **argv, x264_param_t *param, cli_opt_t *opt )
3. {
4.     char *input_filename = NULL;
5.     const char *demuxer = demuxer_names[0];
6.     char *output_filename = NULL;
7.     const char *muxer = muxer_names[0];
8.     char *tcfile_name = NULL;
9.     x264_param_t defaults;
10.    char *profile = NULL;
11.    char *vid_filters = NULL;
12.    int b_thread_input = 0;
13.    int b_turbo = 1;
14.    int b_user_ref = 0;
15.    int b_user_fps = 0;
16.    int b_user_interlaced = 0;
17.    cli_input_opt_t input_opt;
18.    cli_output_opt_t output_opt;
19.    char *preset = NULL;
20.    char *tune = NULL;
21.    //初始化参数默认值
22.    x264_param_default( &defaults );
23.    cli_log_level = defaults.i_log_level;
24.
25.    memset( &input_opt, 0, sizeof(cli_input_opt_t) );
26.    memset( &output_opt, 0, sizeof(cli_output_opt_t) );
27.    input_opt.bit_depth = 8;
28.    input_opt.input_range = input_opt.output_range = param->vui.b_fullrange = RANGE_AUTO;
29.    int output_csp = defaults.i_csp;
30.    opt->b_progress = 1;
31.
32.    /* Presets are applied before all other options. */
33.    for( optind = 0;; )
34.    {
35.        int c = getopt_long( argc, argv, short_options, long_options, NULL );
36.        if( c == -1 )
37.            break;
38.        if( c == OPT_PRESET )
39.            preset = optarg;
40.        if( c == OPT_TUNE )
41.            tune = optarg;
42.        else if( c == '?' )
43.            return -1;
44.    }
45.
46.    if( preset && !strcasecmp( preset, "placebo" ) )
47.        b_turbo = 0;
48.    //设置preset, tune
49.    if( x264_param_default_preset( param, preset, tune ) < 0 )
50.        return -1;
51.
52.    /* Parse command line options */
53.    //解析命令行选项
54.    for( optind = 0;; )
55.    {
56.        int b_error = 0;
57.        int long_options_index = -1;
58.
59.        int c = getopt_long( argc, argv, short_options, long_options, &long_options_index );
60.
61.        if( c == -1 )
62.        {
63.            break;
64.        }
65.        //不同的选项做不同的处理
66.        switch( c )
67.        {
68.            case 'h':
69.                help( &defaults, 0 );//"-h"帮助菜单
70.                exit(0);
71.            case OPT_LONGHELP:
72.                help( &defaults, 1 );
73.                exit(0);
74.            case OPT_FULLHELP:
75.                help( &defaults, 2 );
76.                exit(0);
77.            case 'V':
78.                print_version_info();//打印版本信息
79.                exit(0);
80.            case OPT_FRAMES:
81.                param->i_frame_total = X264_MAX( atoi( optarg ), 0 );
82.                break;
83.            case OPT_SEEK:
84.                opt->i_seek = X264_MAX( atoi( optarg ), 0 );
85.                break;
86.            case 'o':
87.                output_filename = optarg;//输出文件路径
88.                break;
```

```

88.         break;
89.     case OPT_MUXER:
90.         FAIL_IF_ERROR( parse_enum_name( optarg, muxer_names, &muxer ), "Unknown muxer `%s'\n", optarg )
91.         break;
92.     case OPT_DEMUXER:
93.         FAIL_IF_ERROR( parse_enum_name( optarg, demuxer_names, &demuxer ), "Unknown demuxer `%s'\n", optarg )
94.         break;
95.     case OPT_INDEX:
96.         input_opt.index_file = optarg;
97.         break;
98.     case OPT_QPFILE:
99.         opt->qpfile = x264_fopen( optarg, "rb" );
100.        FAIL_IF_ERROR( !opt->qpfile, "can't open qpfile `%s'\n", optarg )
101.        if( !x264_is_regular_file( opt->qpfile ) )
102.        {
103.            x264_cli_log( "x264", X264_LOG_ERROR, "qpfile incompatible with non-regular file `%s'\n", optarg );
104.            fclose( opt->qpfile );
105.            return -1;
106.        }
107.        break;
108.     case OPT_THREAD_INPUT:
109.         b_thread_input = 1;
110.         break;
111.     case OPT_QUIET:
112.         cli_log_level = param->i_log_level = X264_LOG_NONE; //设置log级别
113.         break;
114.     case 'v':
115.         cli_log_level = param->i_log_level = X264_LOG_DEBUG; //设置log级别
116.         break;
117.     case OPT_LOG_LEVEL:
118.         if( !parse_enum_value( optarg, log_level_names, &cli_log_level ) )
119.             cli_log_level += X264_LOG_NONE;
120.         else
121.             cli_log_level = atoi( optarg );
122.         param->i_log_level = cli_log_level; //设置log级别
123.         break;
124.     case OPT_NOPROGRESS:
125.         opt->b_progress = 0;
126.         break;
127.     case OPT_TUNE:
128.     case OPT_PRESET:
129.         break;
130.     case OPT_PROFILE:
131.         profile = optarg;
132.         break;
133.     case OPT_SLOWFIRSTPASS:
134.         b_turbo = 0;
135.         break;
136.     case 'r':
137.         b_user_ref = 1;
138.         goto generic_option;
139.     case OPT_FPS:
140.         b_user_fps = 1;
141.         param->b_vfr_input = 0;
142.         goto generic_option;
143.     case OPT_INTERLACED:
144.         b_user_interlaced = 1;
145.         goto generic_option;
146.     case OPT_TCFILE_IN:
147.         tcfile_name = optarg;
148.         break;
149.     case OPT_TCFILE_OUT:
150.         opt->tcfile_out = x264_fopen( optarg, "wb" );
151.         FAIL_IF_ERROR( !opt->tcfile_out, "can't open `%s'\n", optarg )
152.         break;
153.     case OPT_TIMEBASE:
154.         input_opt.timebase = optarg;
155.         break;
156.     case OPT_PULLDOWN:
157.         FAIL_IF_ERROR( parse_enum_value( optarg, pulldown_names, &opt->i_pulldown ), "Unknown pulldown `%s'\n", optarg )
158.         break;
159.     case OPT_VIDEO_FILTER:
160.         vid_filters = optarg;
161.         break;
162.     case OPT_INPUT_FMT:
163.         input_opt.format = optarg; //输入文件格式
164.         break;
165.     case OPT_INPUT_RES:
166.         input_opt.resolution = optarg; //输入分辨率
167.         break;
168.     case OPT_INPUT_CSP:
169.         input_opt.colorspace = optarg; //输入色域
170.         break;
171.     case OPT_INPUT_DEPTH:
172.         input_opt.bit_depth = atoi( optarg ); //输入颜色位深
173.         break;
174.     case OPT_DTS_COMPRESSION:
175.         output_opt.use_dts_compress = 1;
176.         break;
177.     case OPT_OUTPUT_CSP:
178.         FAIL_IF_ERROR( parse_enum_value( optarg, output_csp_names, &output_csp ), "Unknown output csp `%s'\n", optarg )
179.         // correct the parsed value to the libx264 csp value

```

```

180. // correct the parsed value to the closest csp value
181. #if X264_CHROMA_FORMAT
182.     static const uint8_t output_csp_fix[] = { X264_CHROMA_FORMAT, X264_CSP_RGB };
183. #else
184.     static const uint8_t output_csp_fix[] = { X264_CSP_I420, X264_CSP_I422, X264_CSP_I444, X264_CSP_RGB };
185. #endif
186.     param->i_csp = output_csp = output_csp_fix[output_csp];
187.     break;
188. case OPT_INPUT_RANGE:
189.     FAIL_IF_ERROR( parse_enum_value( optarg, range_names, &input_opt.input_range ), "Unknown input range '%s'\n", optarg
190. )
191.     input_opt.input_range += RANGE_AUTO;
192.     break;
193. case OPT_RANGE:
194.     FAIL_IF_ERROR( parse_enum_value( optarg, range_names, &m->vui.b_fullrange ), "Unknown range '%s'\n", optarg );
195.     input_opt.output_range = param->vui.b_fullrange += RANGE_AUTO;
196.     break;
197. default:
198. generic_option:
199.     {
200.         if( long_options_index < 0 )
201.         {
202.             for( int i = 0; long_options[i].name; i++ )
203.                 if( long_options[i].val == c )
204.                 {
205.                     long_options_index = i;
206.                     break;
207.                 }
208.             if( long_options_index < 0 )
209.             {
210.                 /* getopt_long already printed an error message */
211.                 return -1;
212.             }
213.         }
214.         //解析以字符串方式输入的参数
215.         //即选项名称和选项值都是字符串
216.         b_error |= x264_param_parse( param, long_options[long_options_index].name, optarg );
217.     }
218. }
219. if( b_error )
220. {
221.     const char *name = long_options_index > 0 ? long_options[long_options_index].name : argv[optind-2];
222.     x264_cli_log( "x264", X264_LOG_ERROR, "invalid argument: %s = %s\n", name, optarg );
223.     return -1;
224. }
225.
226. /* If first pass mode is used, apply faster settings. */
227. if( b_turbo )
228.     x264_param_apply_fastfirstpass( param );
229.
230. /* Apply profile restrictions. */
231. //设置profile
232. if( x264_param_apply_profile( param, profile ) < 0 )
233.     return -1;
234.
235. /* Get the file name */
236. FAIL_IF_ERROR( optind > argc - 1 || !output_filename, "No %s file. Run x264 --help for a list of options.\n",
237.     optind > argc - 1 ? "input" : "output" )
238. //根据文件名的后缀确定输出的文件格式 (raw H264, flv, mp4...)
239. if( select_output( muxer, output_filename, param ) )
240.     return -1;
241. FAIL_IF_ERROR( cli_output.open_file( output_filename, &opt->hout, &output_opt ), "could not open output file '%s'\n", output_fil
242. ename )
243. //输入文件路径
244. input_filename = argv[optind++];
245. video_info_t info = {0};
246. char demuxername[5];
247.
248. /* set info flags to be overwritten by demuxer as necessary. */
249. //设置info结构体
250. info.csp = param->i_csp;
251. info.fps_num = param->i_fps_num;
252. info.fps_den = param->i_fps_den;
253. info.fullrange = input_opt.input_range == RANGE_PC;
254. info.interlaced = param->b_interlaced;
255. if( param->vui.i_sar_width > 0 && param->vui.i_sar_height > 0 )
256. {
257.     info.sar_width = param->vui.i_sar_width;
258.     info.sar_height = param->vui.i_sar_height;
259. }
260. info.tff = param->b_tff;
261. info.vfr = param->b_vfr_input;
262.
263. input_opt.seek = opt->i_seek;
264. input_opt.progress = opt->b_progress;
265. input_opt.output_csp = output_csp;
266. //设置输入文件的格式 (yuv, y4m...)
267. if( select_input( demuxer, demuxername, input_filename, &opt->hin, &info, &input_opt ) )
268.     return -1;

```

```

269.     FAIL_IF_ERROR( !opt->hin && cli_input.open_file( input_filename, &opt->hin, &info, &input_opt ),
270.                  "could not open input file '%s'\n", input_filename )
271.
272.     x264_reduce_fraction( &info.sar_width, &info.sar_height );
273.     x264_reduce_fraction( &info.fps_num, &info.fps_den );
274.     x264_cli_log( demuxername, X264_LOG_INFO, "%dx%d%c %u:%u @ %u/%u fps (%cfr)\n", info.width,
275.                  info.height, info.interlaced ? 'i' : 'p', info.sar_width, info.sar_height,
276.                  info.fps_num, info.fps_den, info.vfr ? 'v' : 'c' );
277.
278.     if( tcfile_name )
279.     {
280.         FAIL_IF_ERROR( b_user_fps, "--fps + --tcfile-in is incompatible.\n" )
281.         FAIL_IF_ERROR( timecode_input.open_file( tcfile_name, &opt->hin, &info, &input_opt ), "timecode input failed\n" )
282.         cli_input = timecode_input;
283.     }
284.     else FAIL_IF_ERROR( !info.vfr && input_opt.timebase, "--timebase is incompatible with cfr input\n" )
285.
286.     /* init threaded input while the information about the input video is unaltered by filtering */
287. #if HAVE_THREAD
288.     if( info.thread_safe && (b_thread_input || param->i_threads > 1
289.        || (param->i_threads == X264_THREADS_AUTO && x264_cpu_num_processors() > 1)) )
290.     {
291.         if( thread_input.open_file( NULL, &opt->hin, &info, NULL ) )
292.         {
293.             fprintf( stderr, "x264 [error]: threaded input failed\n" );
294.             return -1;
295.         }
296.         cli_input = thread_input;
297.     }
298. #endif
299.
300.     /* override detected values by those specified by the user */
301.     if( param->vui.i_sar_width > 0 && param->vui.i_sar_height > 0 )
302.     {
303.         info.sar_width = param->vui.i_sar_width;
304.         info.sar_height = param->vui.i_sar_height;
305.     }
306.     if( b_user_fps )
307.     {
308.         info.fps_num = param->i_fps_num;
309.         info.fps_den = param->i_fps_den;
310.     }
311.     if( !info.vfr )
312.     {
313.         info.timebase_num = info.fps_den;
314.         info.timebase_den = info.fps_num;
315.     }
316.     if( !tcfile_name && input_opt.timebase )
317.     {
318.         uint64_t i_user_timebase_num;
319.         uint64_t i_user_timebase_den;
320.         int ret = sscanf( input_opt.timebase, "%SCNu64"/"%SCNu64", &i_user_timebase_num, &i_user_timebase_den );
321.         FAIL_IF_ERROR( !ret, "invalid argument: timebase = %s\n", input_opt.timebase )
322.         else if( ret == 1 )
323.         {
324.             i_user_timebase_num = info.timebase_num;
325.             i_user_timebase_den = strtoul( input_opt.timebase, NULL, 10 );
326.         }
327.         FAIL_IF_ERROR( i_user_timebase_num > UINT32_MAX || i_user_timebase_den > UINT32_MAX,
328.                      "timebase you specified exceeds H.264 maximum\n" )
329.         opt->timebase_convert_multiplier = ((double)i_user_timebase_den / info.timebase_den)
330.             * ((double)info.timebase_num / i_user_timebase_num);
331.         info.timebase_num = i_user_timebase_num;
332.         info.timebase_den = i_user_timebase_den;
333.         info.vfr = 1;
334.     }
335.     if( b_user_interlaced )
336.     {
337.         info.interlaced = param->b_interlaced;
338.         info.tff = param->b_tff;
339.     }
340.     if( input_opt.input_range != RANGE_AUTO )
341.         info.fullrange = input_opt.input_range;
342.
343.     //初始化滤镜filter
344.     //filter可以认为是一种“扩展”了的输入源
345.     if( init_vid_filters( vid_filters, &opt->hin, &info, param, output_csp ) )
346.         return -1;
347.
348.     /* set param flags from the post-filtered video */
349.     param->b_vfr_input = info.vfr;
350.     param->i_fps_num = info.fps_num;
351.     param->i_fps_den = info.fps_den;
352.     param->i_timebase_num = info.timebase_num;
353.     param->i_timebase_den = info.timebase_den;
354.     param->vui.i_sar_width = info.sar_width;
355.     param->vui.i_sar_height = info.sar_height;
356.
357.     info.num_frames = X264_MAX( info.num_frames - opt->i_seek, 0 );
358.     if( (!info.num_frames || param->i_frame_total < info.num_frames)
359.         && param->i_frame_total > 0 )

```

```

360.         info.num_frames = param->i_frame_total;
361.         param->i_frame_total = info.num_frames;
362.
363.         if( !b_user_interlaced && info.interlaced )
364.         {
365.             #if HAVE_INTERLACED
366.                 x264_cli_log( "x264", X264_LOG_WARNING, "input appears to be interlaced, enabling %cff interlaced mode.\n"
367.                     "                If you want otherwise, use --no-interlaced or --%cff\n",
368.                     info.tff ? 't' : 'b', info.tff ? 'b' : 't' );
369.                 param->b_interlaced = 1;
370.                 param->b_tff = !!info.tff;
371.             #else
372.                 x264_cli_log( "x264", X264_LOG_WARNING, "input appears to be interlaced, but not compiled with interlaced support\n" );
373.             #endif
374.         }
375.         /* if the user never specified the output range and the input is now rgb, default it to pc */
376.         int csp = param->i_csp & X264_CSP_MASK;
377.         if( csp >= X264_CSP_BGR && csp <= X264_CSP_RGB )
378.         {
379.             if( input_opt.output_range == RANGE_AUTO )
380.                 param->vui.b_fullrange = RANGE_PC;
381.             /* otherwise fail if they specified tv */
382.             FAIL_IF_ERROR( !param->vui.b_fullrange, "RGB must be PC range" )
383.         }
384.
385.         /* Automatically reduce reference frame count to match the user's target level
386.          * if the user didn't explicitly set a reference frame count. */
387.         if( !b_user_ref )
388.         {
389.             int mbs = (((param->i_width)+15)>>4) * (((param->i_height)+15)>>4);
390.             for( int i = 0; x264_levels[i].level_idc != 0; i++ )
391.                 if( param->i_level_idc == x264_levels[i].level_idc )
392.                 {
393.                     while( mbs * param->i_frame_reference > x264_levels[i].dpb && param->i_frame_reference > 1 )
394.                         param->i_frame_reference--;
395.                     break;
396.                 }
397.         }
398.
399.
400.         return 0;
401.     }

```

下面简单梳理parse()的流程：

- (1) 调用x264\_param\_default()为存储参数的结构体x264\_param\_t赋默认值
- (2) 调用x264\_param\_default\_preset()为x264\_param\_t赋值
- (3) 在一个大循环中调用getopt\_long()逐个解析输入的参数，并作相应的处理。举几个例子：
  - a)“-h”：调用help()打开帮助菜单。
  - b)“-V”调用print\_version\_info()打印版本信息。
  - c)对于长选项，调用x264\_param\_parse()进行处理。
- (4) 调用select\_input()解析输出文件格式（例如raw, flv, MP4...）
- (5) 调用select\_output()解析输入文件格式（例如yuv, y4m...）

下文按照顺序记录parse()中涉及到的函数：

```

x264_param_default()
x264_param_default_preset()
help()
print_version_info()
x264_param_parse()
select_input()
select_output()

```

## x264\_param\_default()

x264\_param\_default()是一个x264的API。该函数用于设置x264中x264\_param\_t结构体的默认值。函数的声明如下所示。

```

1.  /* x264_param_default:
2.   *      fill x264_param_t with default values and do CPU detection */
3.  void x264_param_default( x264_param_t * );

```

x264\_param\_default()的定义如下所示。

```

1.  /**
2.   * x264_param_default:
3.   *      ****
4.   *      //初始化参数默认值
5.   void x264_param_default( x264_param_t *param )
6.  {

```

```

7.  /* */
8.  memset( param, 0, sizeof( x264_param_t ) );
9.
10. /* CPU autodetect */
11. param->cpu = x264_cpu_detect();
12. param->i_threads = X264_THREADS_AUTO;
13. param->i_lookahead_threads = X264_THREADS_AUTO;
14. param->b_deterministic = 1;
15. param->i_sync_lookahead = X264_SYNC_LOOKAHEAD_AUTO;
16.
17. /* Video properties */
18. param->i_csp = X264_CHROMA_FORMAT ? X264_CHROMA_FORMAT : X264_CSP_I420;
19. param->i_width = 0;
20. param->i_height = 0;
21. param->vui.i_sar_width = 0;
22. param->vui.i_sar_height = 0;
23. param->vui.i_overscan = 0; /* undef */
24. param->vui.i_vidformat = 5; /* undef */
25. param->vui.b_fullrange = -1; /* default depends on input */
26. param->vui.i_colorprim = 2; /* undef */
27. param->vui.i_transfer = 2; /* undef */
28. param->vui.i_colmatrix = -1; /* default depends on input */
29. param->vui.i_chroma_loc = 0; /* left center */
30. param->i_fps_num = 25;
31. param->i_fps_den = 1;
32. param->i_level_idc = -1;
33. param->i_slice_max_size = 0;
34. param->i_slice_max_mbs = 0;
35. param->i_slice_count = 0;
36.
37. /* Encoder parameters */
38. //编码参数--最常见
39. param->i_frame_reference = 3;
40. param->i_keyint_max = 250;
41. param->i_keyint_min = X264_KEYINT_MIN_AUTO;
42. param->i_bframe = 3;
43. param->i_scenecut_threshold = 40;
44. param->i_bframe_adaptive = X264_B_ADAPT_FAST;
45. param->i_bframe_bias = 0;
46. param->i_bframe_pyramid = X264_B_PYRAMID_NORMAL;
47. param->b_interlaced = 0;
48. param->b_constrained_intra = 0;
49.
50. param->b_deblocking_filter = 1;
51. param->i_deblocking_filter_alphac0 = 0;
52. param->i_deblocking_filter_beta = 0;
53.
54. param->b_cabac = 1;
55. param->i_cabac_init_idc = 0;
56. //码率控制模块 Rate Control
57. param->rc.i_rc_method = X264_RC_CRF;
58. param->rc.i_bitrate = 0;
59. param->rc.f_rate_tolerance = 1.0;
60. param->rc.i_vbv_max_bitrate = 0;
61. param->rc.i_vbv_buffer_size = 0;
62. param->rc.f_vbv_buffer_init = 0.9;
63. param->rc.i_qp_constant = 23 + QP_BD_OFFSET;
64. param->rc.f_rf_constant = 23;
65. param->rc.i_qp_min = 0;
66. param->rc.i_qp_max = QP_MAX;
67. param->rc.i_qp_step = 4;
68. param->rc.f_ip_factor = 1.4;
69. param->rc.f_pb_factor = 1.3;
70. param->rc.i_aq_mode = X264_AQ_VARIANCE;
71. param->rc.f_aq_strength = 1.0;
72. param->rc.i_lookahead = 40;
73.
74. param->rc.b_stat_write = 0;
75. param->rc.psz_stat_out = "x264_2pass.log";
76. param->rc.b_stat_read = 0;
77. param->rc.psz_stat_in = "x264_2pass.log";
78. param->rc.f_qcompress = 0.6;
79. param->rc.f_qblur = 0.5;
80. param->rc.f_complexity_blur = 20;
81. param->rc.i_zones = 0;
82. param->rc.b_mb_tree = 1;
83.
84. /* Log */
85. //日志模块
86. param->pf_log = x264_log_default;
87. param->p_log_private = NULL;
88. param->i_log_level = X264_LOG_INFO;
89.
90. /* */
91. //分析模块 Analysis
92. param->analyse.intra = X264_ANALYSE_I4x4 | X264_ANALYSE_I8x8;
93. param->analyse.inter = X264_ANALYSE_I4x4 | X264_ANALYSE_I8x8
94. | X264_ANALYSE_PSUB16x16 | X264_ANALYSE_BSUB16x16;
95. param->analyse.i_direct_mv_pred = X264_DIRECT_PRED_SPATIAL;
96. param->analyse.i_me_method = X264_ME_HEX;
97. param->analyse.f_psy_rd = 1.0;

```



```

98.     param->analyse.b_psy = 1;
99.     param->analyse.f_psy_trellis = 0;
100.    param->analyse.i_me_range = 16;
101.    param->analyse.i_subpel_refine = 7;
102.    param->analyse.b_mixed_references = 1;
103.    param->analyse.b_chroma_me = 1;
104.    param->analyse.i_mv_range_thread = -1;
105.    param->analyse.i_mv_range = -1; // set from level_idc
106.    param->analyse.i_chroma_qp_offset = 0;
107.    param->analyse.b_fast_pskip = 1;
108.    param->analyse.b_weighted_bipred = 1;
109.    param->analyse.i_weighted_pred = X264_WEIGHTP_SMART;
110.    param->analyse.b_dct_decimate = 1;
111.    param->analyse.b_transform_8x8 = 1;
112.    param->analyse.i_trellis = 1;
113.    param->analyse.i_luma_deadzone[0] = 21;
114.    param->analyse.i_luma_deadzone[1] = 11;
115.    param->analyse.b_psnr = 0;
116.    param->analyse.b_ssim = 0;
117.
118.    param->i_cqm_preset = X264_CQM_FLAT;
119.    memset( param->cqm_4iy, 16, sizeof( param->cqm_4iy ) );
120.    memset( param->cqm_4py, 16, sizeof( param->cqm_4py ) );
121.    memset( param->cqm_4ic, 16, sizeof( param->cqm_4ic ) );
122.    memset( param->cqm_4pc, 16, sizeof( param->cqm_4pc ) );
123.    memset( param->cqm_8iy, 16, sizeof( param->cqm_8iy ) );
124.    memset( param->cqm_8py, 16, sizeof( param->cqm_8py ) );
125.    memset( param->cqm_8ic, 16, sizeof( param->cqm_8ic ) );
126.    memset( param->cqm_8pc, 16, sizeof( param->cqm_8pc ) );
127.
128.    param->b_repeat_headers = 1;
129.    param->b_annexb = 1;
130.    param->b_aud = 0;
131.    param->b_vfr_input = 1;
132.    param->i_nal_hrd = X264_NAL_HRD_NONE;
133.    param->b_tff = 1;
134.    param->b_pic_struct = 0;
135.    param->b_fake_interlaced = 0;
136.    param->i_frame_packing = -1;
137.    param->b_opencl = 0;
138.    param->i_opencl_device = 0;
139.    param->opencl_device_id = NULL;
140.    param->psz_clbin_file = NULL;
141. }

```

从源代码可以看出，x264\_param\_default()对输入的存储参数的结构体x264\_param\_t的成员变量进行了赋值工作。

## x264\_param\_default\_preset()

x264\_param\_default\_preset()是一个libx264的API，用于设置x264的preset和tune。该函数的声明如下所示。

```

1.  /*      Multiple tunings can be used if separated by a delimiter in ",./-+",
2.  *      however multiple psy tunings cannot be used.
3.  *      film, animation, grain, stillimage, psnr, and ssim are psy tunings.
4.  *
5.  *      returns 0 on success, negative on failure (e.g. invalid preset/tune name). */
6.  int x264_param_default_preset( x264_param_t *, const char *preset, const char *tune );

```

x264\_param\_default\_preset()的定义如下所示。

```

1.  //设置preset, tune
2.  int x264_param_default_preset( x264_param_t *param, const char *preset, const char *tune )
3.  {
4.      x264_param_default( param );
5.
6.      //设置preset
7.      if( preset && x264_param_apply_preset( param, preset ) < 0 )
8.          return -1;
9.
10.     //设置tune
11.     if( tune && x264_param_apply_tune( param, tune ) < 0 )
12.         return -1;
13.     return 0;
14. }

```

从源代码可以看出，x264\_param\_default\_preset()调用x264\_param\_apply\_preset()设置preset，调用x264\_param\_apply\_tune()设置tune。记录一下这两个函数。

### x264\_param\_apply\_preset()

x264\_param\_apply\_preset()用于设置preset。该函数的定义如下所示。

```

1.  //设置preset

```

```

1. // x264 preset
2. static int x264_param_apply_preset( x264_param_t *param, const char *preset )
3. {
4.     char *end;
5.     int i = strtol( preset, &end, 10 );
6.     if( *end == 0 && i >= 0 && i < sizeof(x264_preset_names)/sizeof(*x264_preset_names)-1 )
7.         preset = x264_preset_names[i];
8.
9.     //几种不同的preset设置不同的参数
10.    if( !strcasecmp( preset, "ultrafast" ) )
11.    {
12.        param->i_frame_reference = 1;
13.        param->i_scenecut_threshold = 0;
14.        param->b_deblocking_filter = 0; //不使用去块滤波
15.        param->b_cabac = 0; //不使用CABAC
16.        param->i_bframe = 0; //不使用B帧
17.        param->analyse.intra = 0;
18.        param->analyse.inter = 0;
19.        param->analyse.b_transform_8x8 = 0; //不使用8x8DCT
20.        param->analyse.i_me_method = X264_ME_DIA; //运动搜索方法使用"Diamond"
21.        param->analyse.i_subpel_refine = 0;
22.        param->rc.i_aq_mode = 0;
23.        param->analyse.b_mixed_references = 0;
24.        param->analyse.i_trellis = 0;
25.        param->i_bframe_adaptive = X264_B_ADAPT_NONE;
26.        param->rc.b_mb_tree = 0;
27.        param->analyse.i_weighted_pred = X264_WEIGHTP_NONE; //不使用加权
28.        param->analyse.b_weighted_bipred = 0;
29.        param->rc.i_lookahead = 0;
30.    }
31.    else if( !strcasecmp( preset, "superfast" ) )
32.    {
33.        param->analyse.inter = X264_ANALYSE_I8x8|X264_ANALYSE_I4x4;
34.        param->analyse.i_me_method = X264_ME_DIA; //钻石模板
35.        param->analyse.i_subpel_refine = 1; //亚像素运动估计质量为1
36.        param->i_frame_reference = 1;
37.        param->analyse.b_mixed_references = 0;
38.        param->analyse.i_trellis = 0;
39.        param->rc.b_mb_tree = 0;
40.        param->analyse.i_weighted_pred = X264_WEIGHTP_SIMPLE;
41.        param->rc.i_lookahead = 0;
42.    }
43.    else if( !strcasecmp( preset, "veryfast" ) )
44.    {
45.        param->analyse.i_me_method = X264_ME_HEX; //六边形模板
46.        param->analyse.i_subpel_refine = 2;
47.        param->i_frame_reference = 1;
48.        param->analyse.b_mixed_references = 0;
49.        param->analyse.i_trellis = 0;
50.        param->analyse.i_weighted_pred = X264_WEIGHTP_SIMPLE;
51.        param->rc.i_lookahead = 10;
52.    }
53.    else if( !strcasecmp( preset, "faster" ) )
54.    {
55.        param->analyse.b_mixed_references = 0;
56.        param->i_frame_reference = 2;
57.        param->analyse.i_subpel_refine = 4;
58.        param->analyse.i_weighted_pred = X264_WEIGHTP_SIMPLE;
59.        param->rc.i_lookahead = 20;
60.    }
61.    else if( !strcasecmp( preset, "fast" ) )
62.    {
63.        param->i_frame_reference = 2;
64.        param->analyse.i_subpel_refine = 6;
65.        param->analyse.i_weighted_pred = X264_WEIGHTP_SIMPLE;
66.        param->rc.i_lookahead = 30;
67.    }
68.    else if( !strcasecmp( preset, "medium" ) )
69.    {
70.        /* Default is medium */
71.    }
72.    else if( !strcasecmp( preset, "slow" ) )
73.    {
74.        param->analyse.i_me_method = X264_ME_UMH; //UMH相对复杂
75.        param->analyse.i_subpel_refine = 8; //亚像素运动估计质量为8
76.        param->i_frame_reference = 5;
77.        param->i_bframe_adaptive = X264_B_ADAPT_TRELLIS;
78.        param->analyse.i_direct_mv_pred = X264_DIRECT_PRED_AUTO;
79.        param->rc.i_lookahead = 50;
80.    }
81.    else if( !strcasecmp( preset, "slower" ) )
82.    {
83.        param->analyse.i_me_method = X264_ME_UMH;
84.        param->analyse.i_subpel_refine = 9;
85.        param->i_frame_reference = 8;
86.        param->i_bframe_adaptive = X264_B_ADAPT_TRELLIS;
87.        param->analyse.i_direct_mv_pred = X264_DIRECT_PRED_AUTO;
88.        param->analyse.inter |= X264_ANALYSE_PSUB8x8;
89.        param->analyse.i_trellis = 2;
90.        param->rc.i_lookahead = 60;
91.    }
92.    else if( !strcasecmp( preset, "veryslow" ) )

```

```

93.     {
94.         param->analyse.i_me_method = X264_ME_UMH;
95.         param->analyse.i_subpel_refine = 10;
96.         param->analyse.i_me_range = 24;
97.         param->i_frame_reference = 16;
98.         param->i_bframe_adaptive = X264_B_ADAPT_TRELLIS;
99.         param->analyse.i_direct_mv_pred = X264_DIRECT_PRED_AUTO;
100.        param->analyse.inter |= X264_ANALYSE_PSUB8x8;
101.        param->analyse.i_trellis = 2;
102.        param->i_bframe = 8;
103.        param->rc.i_lookahead = 60;
104.    }
105.    else if( !strcasecmp( preset, "placebo" ) )
106.    {
107.        param->analyse.i_me_method = X264_ME_TESA;//TESA很慢
108.        param->analyse.i_subpel_refine = 11;
109.        param->analyse.i_me_range = 24;
110.        param->i_frame_reference = 16;
111.        param->i_bframe_adaptive = X264_B_ADAPT_TRELLIS;
112.        param->analyse.i_direct_mv_pred = X264_DIRECT_PRED_AUTO;
113.        param->analyse.inter |= X264_ANALYSE_PSUB8x8;
114.        param->analyse.b_fast_pskip = 0;
115.        param->analyse.i_trellis = 2;
116.        param->i_bframe = 16;
117.        param->rc.i_lookahead = 60;
118.    }
119.    else
120.    {
121.        x264_log( NULL, X264_LOG_ERROR, "invalid preset '%s'\n", preset );
122.        return -1;
123.    }
124.    return 0;
125. }

```

可以看出x264\_param\_apply\_preset()通过strcasecmp()比较字符串的方法得到输入的preset类型；然后根据preset类型，设定 x264\_param\_t中相应的参数。

#### x264\_param\_apply\_tune()

x264\_param\_apply\_tune()用于设置tune。该函数的定义如下所示。

```

[cpp]
1. //设置tune
2. static int x264_param_apply_tune( x264_param_t *param, const char *tune )
3. {
4.     char *tmp = x264_malloc( strlen( tune ) + 1 );
5.     if( !tmp )
6.         return -1;
7.     tmp = strcpy( tmp, tune );
8.     //分解一个字符串为一个字符串数组。第2个参数为分隔符
9.     char *s = strtok( tmp, ",./-+" );
10.    int psy_tuning_used = 0;
11.
12.    //设置
13.    //这里是循环的，可以设置多次
14.    while( s )
15.    {
16.        if( !strcasecmp( s, "film", 4 ) )
17.        {
18.            if( psy_tuning_used++ ) goto psy_failure;
19.            param->i_deblocking_filter_alphac0 = -1;
20.            param->i_deblocking_filter_beta = -1;
21.            param->analyse.f_psy_trellis = 0.15;
22.        }
23.        else if( !strcasecmp( s, "animation", 9 ) )
24.        {
25.            if( psy_tuning_used++ ) goto psy_failure;
26.            param->i_frame_reference = param->i_frame_reference > 1 ? param->i_frame_reference*2 : 1;
27.            param->i_deblocking_filter_alphac0 = 1;
28.            param->i_deblocking_filter_beta = 1;
29.            param->analyse.f_psy_rd = 0.4;
30.            param->rc.f_aq_strength = 0.6;
31.            param->i_bframe += 2;
32.        }
33.        else if( !strcasecmp( s, "grain", 5 ) )
34.        {
35.            if( psy_tuning_used++ ) goto psy_failure;
36.            param->i_deblocking_filter_alphac0 = -2;
37.            param->i_deblocking_filter_beta = -2;
38.            param->analyse.f_psy_trellis = 0.25;
39.            param->analyse.b_dct_decimate = 0;
40.            param->rc.f_pb_factor = 1.1;
41.            param->rc.f_ip_factor = 1.1;
42.            param->rc.f_aq_strength = 0.5;
43.            param->analyse.i_luma_deadzone[0] = 6;
44.            param->analyse.i_luma_deadzone[1] = 6;
45.            param->rc.f_qcompress = 0.8;
46.        }
47.        else if( !strcasecmp( s, "stillimage", 10 ) )
48.        {

```

```

49.         if( psy_tuning_used++ ) goto psy_failure;
50.         param->i_deblocking_filter_alphac0 = -3;
51.         param->i_deblocking_filter_beta = -3;
52.         param->analyse.f_psy_rd = 2.0;
53.         param->analyse.f_psy_trellis = 0.7;
54.         param->rc.f_aq_strength = 1.2;
55.     }
56.     else if( !strncasecmp( s, "psnr", 4 ) )
57.     {
58.         if( psy_tuning_used++ ) goto psy_failure;
59.         param->rc.i_aq_mode = X264_AQ_NONE;
60.         param->analyse.b_psy = 0;
61.     }
62.     else if( !strncasecmp( s, "ssim", 4 ) )
63.     {
64.         if( psy_tuning_used++ ) goto psy_failure;
65.         param->rc.i_aq_mode = X264_AQ_AUTOVARIANCE;
66.         param->analyse.b_psy = 0;
67.     }
68.     else if( !strncasecmp( s, "fastdecode", 10 ) )
69.     {
70.         param->b_deblocking_filter = 0;
71.         param->b_cabac = 0;
72.         param->analyse.b_weighted_bipred = 0;
73.         param->analyse.i_weighted_pred = X264_WEIGHTP_NONE;
74.     }
75.     else if( !strncasecmp( s, "zerolatency", 11 ) )
76.     {
77.         //zerolatency速度快
78.         param->rc.i_lookahead = 0;
79.         param->i_sync_lookahead = 0;
80.         param->i_bframe = 0;//不使用B帧
81.         param->b_sliced_threads = 1;
82.         param->b_vfr_input = 0;
83.         param->rc.b_mb_tree = 0;
84.     }
85.     else if( !strncasecmp( s, "touhou", 6 ) )
86.     {
87.         if( psy_tuning_used++ ) goto psy_failure;
88.         param->i_frame_reference = param->i_frame_reference > 1 ? param->i_frame_reference*2 : 1;
89.         param->i_deblocking_filter_alphac0 = -1;
90.         param->i_deblocking_filter_beta = -1;
91.         param->analyse.f_psy_trellis = 0.2;
92.         param->rc.f_aq_strength = 1.3;
93.         if( param->analyse.inter & X264_ANALYSE_PSUB16x16 )
94.             param->analyse.inter |= X264_ANALYSE_PSUB8x8;
95.     }
96.     else
97.     {
98.         x264_log( NULL, X264_LOG_ERROR, "invalid tune '%s'\n", s );
99.         x264_free( tmp );
100.        return -1;
101.    }
102.    if( 0 )
103.    {
104.        psy_failure:
105.            x264_log( NULL, X264_LOG_WARNING, "only 1 psy tuning can be used: ignoring tune %s\n", s );
106.        }
107.        s = strtok( NULL, ",./-+" );
108.    }
109.    x264_free( tmp );
110.    return 0;
111. }

```

可以看出x264\_param\_apply\_tune()首先通过strtok()得到存储tune[]数组；然后通过strncasecmp()比较字符串的方法判断当前的tune类型；最后根据tune类型，设定 x264\_param\_t中相应的参数。

## help()

help()用于打印帮助菜单。在x264命令行程序中添加“-h”参数后会调用该函数。该函数的定义如下所示。

```

[cpp]
1. //帮助菜单
2. //longhelp标识是否展开更长的帮助菜单
3. static void help( x264_param_t *defaults, int longhelp )
4. {
5.     char buf[50];
6.     //H0(),H1(),H2()都是printf()
7.     //H1(),H2()只有“长帮助菜单”的情况下才会调用printf()
8.     #define H0 printf
9.     #define H1 if(longhelp>=1) printf
10.    #define H2 if(longhelp==2) printf
11.    H0( "x264 core:%d%s\n",
12.        "Syntax: x264 [options] -o outfile infile\n",
13.        "\n",
14.        "Infile can be raw (in which case resolution is required),\n",
15.        " or YUV4MPEG (*.y4m),\n",
16.        " or AVI (if compiled with support for it)."

```

```

16.         " or Avisynth it compiled with support (%s).\n"
17.         " or libav* formats if compiled with lavf support (%s) or ffms support (%s).\n"
18.         "Output file type is selected by filename:\n"
19.         ".264 -> Raw bytestream\n"
20.         ".mkv -> Matroska\n"
21.         ".flv -> Flash Video\n"
22.         ".mp4 -> MP4 if compiled with GPAC or L-SMASH support (%s).\n"
23.         "Output bit depth: %d (configured at compile time)\n"
24.         "\n"
25.         "Options:\n"
26.         "\n"
27.         " -h, --help             List basic options\n"
28.         "    --longhelp         List more options\n"
29.         "    --fullhelp          List all options\n"
30.         "\n",
31.         X264_BUILD, X264_VERSION,
32. #if HAVE_AVS
33.         "yes",
34. #else
35.         "no",
36. #endif
37. #if HAVE_LAVF
38.         "yes",
39. #else
40.         "no",
41. #endif
42. #if HAVE_FFMS
43.         "yes",
44. #else
45.         "no",
46. #endif
47. #if HAVE_GPAC
48.         "gpac",
49. #elif HAVE_LSMASH
50.         "lsmash",
51. #else
52.         "no",
53. #endif
54.         x264_bit_depth
55.     );
56.     H0( "Example usage:\n" );
57.     H0( "\n" );
58.     H0( "    Constant quality mode:\n" );
59.     H0( "        x264 --crf 24 -o <output> <input>\n" );
60.     H0( "\n" );
61.     H0( "        Two-pass with a bitrate of 1000kbps:\n" );
62.     H0( "            x264 --pass 1 --bitrate 1000 -o <output> <input>\n" );
63.     H0( "            x264 --pass 2 --bitrate 1000 -o <output> <input>\n" );
64.     H0( "\n" );
65.     H0( "        Lossless:\n" );
66.     H0( "            x264 --qp 0 -o <output> <input>\n" );
67.     H0( "\n" );
68.     H0( "        Maximum PSNR at the cost of speed and visual quality:\n" );
69.     H0( "            x264 --preset placebo --tune psnr -o <output> <input>\n" );
70.     H0( "\n" );
71.     H0( "        Constant bitrate at 1000kbps with a 2 second-buffer:\n" );
72.     H0( "            x264 --vbv-buftype 2000 --bitrate 1000 -o <output> <input>\n" );
73.     H0( "\n" );
74.     H0( "Presets:\n" );
75.     H0( "\n" );
76.     H0( "    --profile <string>    Force the limits of an H.264 profile\n"
77.         "                          Overrides all settings.\n" );
78.     H2(
79. #if X264_CHROMA_FORMAT <= X264_CSP_I420
80. #if BIT_DEPTH==8
81.         "                - baseline:\n"
82.         "                --no-8x8dct --bframes 0 --no-cabac\n"
83.         "                --cqm flat --weightp 0\n"
84.         "                No interlaced.\n"
85.         "                No lossless.\n"
86.         "                - main:\n"
87.         "                --no-8x8dct --cqm flat\n"
88.         "                No lossless.\n"
89.         "                - high:\n"
90.         "                No lossless.\n"
91. #endif
92.         "                - high10:\n"
93.         "                No lossless.\n"
94.         "                Support for bit depth 8-10.\n"
95. #endif
96. #if X264_CHROMA_FORMAT <= X264_CSP_I422
97.         "                - high422:\n"
98.         "                No lossless.\n"
99.         "                Support for bit depth 8-10.\n"
100.         "                Support for 4:2:0/4:2:2 chroma subsampling.\n"
101. #endif
102.         "                - high444:\n"
103.         "                Support for bit depth 8-10.\n"
104.         "                Support for 4:2:0/4:2:2/4:4:4 chroma subsampling.\n" );
105.     else H0(
106.         "                - "
107. #if X264_CHROMA_FORMAT <= X264_CSP_I420

```

```

108. #if BIT_DEPTH==8
109.     "baseline,main,high,"
110. #endif
111.     "high10,"
112. #endif
113. #if X264_CHROMA_FORMAT <= X264_CSP_I422
114.     "high422,"
115. #endif
116.     "high444\n"
117. );
118. H0( "    --preset <string>      Use a preset to select encoding settings [medium]\n"
119.     "                                Overridden by user settings.\n" );
120. H2( "    - ultrafast:\n"
121.     "        --no-8x8dct --aq-mode 0 --b-adapt 0\n"
122.     "        --bframes 0 --no-cabac --no-deblock\n"
123.     "        --no-mbtree --me dia --no-mixed-refs\n"
124.     "        --partitions none --rc-lookahead 0 --ref 1\n"
125.     "        --scenecut 0 --subme 0 --trellis 0\n"
126.     "        --no-weightb --weightp 0\n"
127.     "    - superfast:\n"
128.     "        --no-mbtree --me dia --no-mixed-refs\n"
129.     "        --partitions i8x8,i4x4 --rc-lookahead 0\n"
130.     "        --ref 1 --subme 1 --trellis 0 --weightp 1\n"
131.     "    - veryfast:\n"
132.     "        --no-mixed-refs --rc-lookahead 10\n"
133.     "        --ref 1 --subme 2 --trellis 0 --weightp 1\n"
134.     "    - faster:\n"
135.     "        --no-mixed-refs --rc-lookahead 20\n"
136.     "        --ref 2 --subme 4 --weightp 1\n"
137.     "    - fast:\n"
138.     "        --rc-lookahead 30 --ref 2 --subme 6\n"
139.     "        --weightp 1\n"
140.     "    - medium:\n"
141.     "        Default settings apply.\n"
142.     "    - slow:\n"
143.     "        --b-adapt 2 --direct auto --me umh\n"
144.     "        --rc-lookahead 50 --ref 5 --subme 8\n"
145.     "    - slower:\n"
146.     "        --b-adapt 2 --direct auto --me umh\n"
147.     "        --partitions all --rc-lookahead 60\n"
148.     "        --ref 8 --subme 9 --trellis 2\n"
149.     "    - veryslow:\n"
150.     "        --b-adapt 2 --bframes 8 --direct auto\n"
151.     "        --me umh --merange 24 --partitions all\n"
152.     "        --ref 16 --subme 10 --trellis 2\n"
153.     "        --rc-lookahead 60\n"
154.     "    - placebo:\n"
155.     "        --bframes 16 --b-adapt 2 --direct auto\n"
156.     "        --slow-firstpass --no-fast-pskip\n"
157.     "        --me tesa --merange 24 --partitions all\n"
158.     "        --rc-lookahead 60 --ref 16 --subme 11\n"
159.     "        --trellis 2\n" );
160. else H0( "    - ultrafast,superfast,veryfast,faster,fast\n"
161.     "    - medium,slow,slower,veryslow,placebo\n" );
162. H0( "    --tune <string>      Tune the settings for a particular type of source\n"
163.     "                                or situation\n"
164.     "                                Overridden by user settings.\n"
165.     "                                Multiple tunings are separated by commas.\n"
166.     "                                Only one psy tuning can be used at a time.\n" );
167. H2( "    - film (psy tuning):\n"
168.     "        --deblock -1:-1 --psy-rd <unset>:0.15\n"
169.     "    - animation (psy tuning):\n"
170.     "        --bframes {+2} --deblock 1:1\n"
171.     "        --psy-rd 0.4:<unset> --aq-strength 0.6\n"
172.     "        --ref {Double if >1 else 1}\n"
173.     "    - grain (psy tuning):\n"
174.     "        --aq-strength 0.5 --no-dct-decimate\n"
175.     "        --deadzone-inter 6 --deadzone-intra 6\n"
176.     "        --deblock -2:-2 --ipratio 1.1\n"
177.     "        --pbratio 1.1 --psy-rd <unset>:0.25\n"
178.     "        --qcomp 0.8\n"
179.     "    - stillimage (psy tuning):\n"
180.     "        --aq-strength 1.2 --deblock -3:-3\n"
181.     "        --psy-rd 2.0:0.7\n"
182.     "    - psnr (psy tuning):\n"
183.     "        --aq-mode 0 --no-psy\n"
184.     "    - ssim (psy tuning):\n"
185.     "        --aq-mode 2 --no-psy\n"
186.     "    - fastdecode:\n"
187.     "        --no-cabac --no-deblock --no-weightb\n"
188.     "        --weightp 0\n"
189.     "    - zerolatency:\n"
190.     "        --bframes 0 --force-cfr --no-mbtree\n"
191.     "        --sync-lookahead 0 --sliced-threads\n"
192.     "        --rc-lookahead 0\n" );
193. else H0( "    - psy tunings: film,animation,grain,\n"
194.     "    - stillimage,psnr,ssim\n"
195.     "    - other tunings: fastdecode,zerolatency\n" );
196. H2( "    --slow-firstpass      Don't force these faster settings with --pass 1\n"
197.     "        --no-8x8dct --me dia --partitions none\n"
198.     "        --ref 1 --subme {2 if >2 else unchanged}\n"

```

```

199.         --trellis 0 --fast-pskip\n" );
200.     else H1( "      --slow-firstpass      Don't force faster settings with --pass 1\n" );
201.     H0( "\n" );
202.     H0( "Frame-type options:\n" );
203.     H0( "\n" );
204.     H0( "  -I, --keyint <integer or \"infinite\"> Maximum GOP size [%d]\n", defaults->i_keyint_max );
205.     H2( "  -i, --min-keyint <integer> Minimum GOP size [auto]\n" );
206.     H2( "      --no-scenecut      Disable adaptive I-frame decision\n" );
207.     H2( "      --scenecut <integer> How aggressively to insert extra I-frames [%d]\n", defaults->i_scenecut_threshold );
208.     H2( "      --intra-refresh    Use Periodic Intra Refresh instead of IDR frames\n" );
209.     H1( "  -b, --bframes <integer> Number of B-frames between I and P [%d]\n", defaults->i_bframe );
210.     H1( "      --b-adapt <integer> Adaptive B-frame decision method [%d]\n"
211.         "      Higher values may lower threading efficiency.\n"
212.         "      - 0: Disabled\n"
213.         "      - 1: Fast\n"
214.         "      - 2: Optimal (slow with high --bframes)\n", defaults->i_bframe_adaptive );
215.     H2( "      --b-bias <integer> Influences how often B-frames are used [%d]\n", defaults->i_bframe_bias );
216.     H1( "      --b-pyramid <string> Keep some B-frames as references [%s]\n"
217.         "      - none: Disabled\n"
218.         "      - strict: Strictly hierarchical pyramid\n"
219.         "      - normal: Non-strict (not Blu-ray compatible)\n",
220.         strtable_lookup( x264_b_pyramid_names, defaults->i_bframe_pyramid ) );
221.     H1( "      --open-gop      Use recovery points to close GOPs\n"
222.         "      Only available with b-frames\n" );
223.     H1( "      --no-cabac      Disable CABAC\n" );
224.     H1( "      -r, --ref <integer> Number of reference frames [%d]\n", defaults->i_frame_reference );
225.     H1( "      --no-deblock      Disable loop filter\n" );
226.     H1( "      -f, --deblock <alpha:beta> Loop filter parameters [%d:%d]\n",
227.         defaults->i_deblocking_filter_alphac0, defaults->i_deblocking_filter_beta );
228.     H2( "      --slices <integer> Number of slices per frame; forces rectangular\n"
229.         "      slices and is overridden by other slicing options\n" );
230.     else H1( "      --slices <integer> Number of slices per frame\n" );
231.     H2( "      --slices-max <integer> Absolute maximum slices per frame; overrides\n"
232.         "      slice-max-size/slice-max-mbs when necessary\n" );
233.     H2( "      --slice-max-size <integer> Limit the size of each slice in bytes\n" );
234.     H2( "      --slice-max-mbs <integer> Limit the size of each slice in macroblocks (max)\n" );
235.     H2( "      --slice-min-mbs <integer> Limit the size of each slice in macroblocks (min)\n" );
236.     H0( "      --tff      Enable interlaced mode (top field first)\n" );
237.     H0( "      --bff      Enable interlaced mode (bottom field first)\n" );
238.     H2( "      --constrained-intra Enable constrained intra prediction.\n" );
239.     H0( "      --pulldown <string> Use soft pulldown to change frame rate\n"
240.         "      - none, 22, 32, 64, double, triple, euro (requires cfr input)\n" );
241.     H2( "      --fake-interlaced Flag stream as interlaced but encode progressive.\n"
242.         "      Makes it possible to encode 25p and 30p Blu-Ray\n"
243.         "      streams. Ignored in interlaced mode.\n" );
244.     H2( "      --frame-packing <integer> For stereoscopic videos define frame arrangement\n"
245.         "      - 0: checkerboard - pixels are alternatively from L and R\n"
246.         "      - 1: column alternation - L and R are interlaced by column\n"
247.         "      - 2: row alternation - L and R are interlaced by row\n"
248.         "      - 3: side by side - L is on the left, R on the right\n"
249.         "      - 4: top bottom - L is on top, R on bottom\n"
250.         "      - 5: frame alternation - one view per frame\n" );
251.     H0( "\n" );
252.     H0( "Ratecontrol:\n" );
253.     H0( "\n" );
254.     H1( "  -q, --qp <integer> Force constant QP (0-%d, 0=lossless)\n", QP_MAX );
255.     H0( "  -B, --bitrate <integer> Set bitrate (kbit/s)\n" );
256.     H0( "      --crf <float> Quality-based VBR (%d-51) [%d]\n", 51 - QP_MAX_SPEC, defaults->rc.f_rf_constant );
257.     H1( "      --rc-lookahead <integer> Number of frames for frametype lookahead [%d]\n", defaults->rc.i_lookahead );
258.     H0( "      --vbv-maxrate <integer> Max local bitrate (kbit/s) [%d]\n", defaults->rc.i_vbv_max_bitrate );
259.     H0( "      --vbv-buFSIZE <integer> Set size of the VBv buffer (kbit) [%d]\n", defaults->rc.i_vbv_buffer_size );
260.     H2( "      --vbv-init <float> Initial VBv buffer occupancy [%d]\n", defaults->rc.f_vbv_buffer_init );
261.     H2( "      --crf-max <float> With CRF+VBV, limit RF to this value\n"
262.         "      May cause VBv underflows!\n" );
263.     H2( "      --qpmin <integer> Set min QP [%d]\n", defaults->rc.i_qp_min );
264.     H2( "      --qpmax <integer> Set max QP [%d]\n", defaults->rc.i_qp_max );
265.     H2( "      --qpstep <integer> Set max QP step [%d]\n", defaults->rc.i_qp_step );
266.     H2( "      --ratetol <float> Tolerance of ABR ratecontrol and VBv [%d]\n", defaults->rc.f_rate_tolerance );
267.     H2( "      --ipratio <float> QP factor between I and P [%d]\n", defaults->rc.f_ip_factor );
268.     H2( "      --pbratio <float> QP factor between P and B [%d]\n", defaults->rc.f_pb_factor );
269.     H2( "      --chroma-qp-offset <integer> QP difference between chroma and luma [%d]\n", defaults->
270.         analyse.i_chroma_qp_offset );
271.     H2( "      --aq-mode <integer> AQ method [%d]\n"
272.         "      - 0: Disabled\n"
273.         "      - 1: Variance AQ (complexity mask)\n"
274.         "      - 2: Auto-variance AQ (experimental)\n", defaults->rc.i_aq_mode );
275.     H1( "      --aq-strength <float> Reduces blocking and blurring in flat and\n"
276.         "      textured areas. [%d]\n", defaults->rc.f_aq_strength );
277.     H0( "\n" );
278.     H0( "  -p, --pass <integer> Enable multipass ratecontrol\n"
279.         "      - 1: First pass, creates stats file\n"
280.         "      - 2: Last pass, does not overwrite stats file\n" );
281.     H1( "      --stats <string> Filename for 2 pass stats [%s]\n", defaults->rc.psz_stat_out );
282.     H2( "      --no-mbtree      Disable mb-tree ratecontrol.\n" );
283.     H2( "      --qcomp <float> QP curve compression [%d]\n", defaults->rc.f_qcompress );
284.     H2( "      --cplxblur <float> Reduce fluctuations in QP (before curve compression) [%d]\n", defaults->
285.         rc.f_complexity_blur );
286.     H2( "      --qblur <float> Reduce fluctuations in QP (after curve compression) [%d]\n", defaults->rc.f_qblur );
287.     H2( "      --zones <zone0>/<zone1>/... Tweak the bitrate of regions of the video\n"
288.         "      Each zone is of the form\n"

```

```

288.         <start frame>,<end frame>,<option>\n"
289.         where <option> is either\n"
290.         q=<integer> (force QP)\n"
291.         or b=<float> (bitrate multiplier)\n" );
292. H2( "    --qpfile <string> Force frametypes and QPs for some or all frames\n"
293.     "    Format of each line: framenumbers frametype QP\n"
294.     "    QP is optional (none lets x264 choose). Frametypes: I,i,K,P,B,b.\n"
295.     "    K=<I or i> depending on open-gop setting\n"
296.     "    QPs are restricted by qpmin/qpmax.\n" );
297. H1( "\n" );
298. H1( "Analysis:\n" );
299. H1( "\n" );
300. H1( "    -A, --partitions <string> Partitions to consider [\"p8x8,b8x8,i8x8,i4x4\"]\n"
301.     "    - p8x8, p4x4, b8x8, i8x8, i4x4\n"
302.     "    - none, all\n"
303.     "    (p4x4 requires p8x8. i8x8 requires --8x8dct.)\n" );
304. H1( "    --direct <string> Direct MV prediction mode [\"%s\"]\n"
305.     "    - none, spatial, temporal, auto\n",
306.     strttable_lookup( x264_direct_pred_names, defaults->analyse.i_direct_mv_pred ) );
307. H2( "    --no-weightb Disable weighted prediction for B-frames\n" );
308. H1( "    --weightp <integer> Weighted prediction for P-frames [%d]\n"
309.     "    - 0: Disabled\n"
310.     "    - 1: Weighted refs\n"
311.     "    - 2: Weighted refs + Duplicates\n", defaults->analyse.i_weighted_pred );
312. H1( "    --me <string> Integer pixel motion estimation method [\"%s\"]\n",
313.     strttable_lookup( x264_motion_est_names, defaults->analyse.i_me_method ) );
314. H2( "    - dia: diamond search, radius 1 (fast)\n"
315.     "    - hex: hexagonal search, radius 2\n"
316.     "    - umh: uneven multi-hexagon search\n"
317.     "    - esa: exhaustive search\n"
318.     "    - tesa: hadamard exhaustive search (slow)\n" );
319. else H1( "    - dia, hex, umh\n" );
320. H2( "    --merange <integer> Maximum motion vector search range [%d]\n", defaults->analyse.i_me_range );
321. H2( "    --mvrage <integer> Maximum motion vector length [-1 (auto)]\n" );
322. H2( "    --mvrage-thread <int> Minimum buffer between threads [-1 (auto)]\n" );
323. H1( "    -m, --subme <integer> Subpixel motion estimation and mode decision [%d]\n", defaults->analyse.i_subpel_refine );
324. H2( "    - 0: fullpel only (not recommended)\n"
325.     "    - 1: SAD mode decision, one qpel iteration\n"
326.     "    - 2: SATD mode decision\n"
327.     "    - 3-5: Progressively more qpel\n"
328.     "    - 6: RD mode decision for I/P-frames\n"
329.     "    - 7: RD mode decision for all frames\n"
330.     "    - 8: RD refinement for I/P-frames\n"
331.     "    - 9: RD refinement for all frames\n"
332.     "    - 10: QP-RD - requires trellis=2, aq-mode>0\n"
333.     "    - 11: Full RD: disable all early terminations\n" );
334. else H1( "    decision quality: 1=fast, 11=best\n" );
335. H1( "    --psy-rd <float:float> Strength of psychovisual optimization [\"%.1f:%.1f\"]\n"
336.     "    #1: RD (requires subme>=6)\n"
337.     "    #2: Trellis (requires trellis, experimental)\n",
338.     defaults->analyse.f_psy_rd, defaults->analyse.f_psy_trellis );
339. H2( "    --no-psy Disable all visual optimizations that worsen\n"
340.     "    both PSNR and SSIM.\n" );
341. H2( "    --no-mixed-refs Don't decide references on a per partition basis\n" );
342. H2( "    --no-chroma-me Ignore chroma in motion estimation\n" );
343. H1( "    --no-8x8dct Disable adaptive spatial transform size\n" );
344. H1( "    -t, --trellis <integer> Trellis RD quantization. [%d]\n"
345.     "    - 0: disabled\n"
346.     "    - 1: enabled only on the final encode of a MB\n"
347.     "    - 2: enabled on all mode decisions\n", defaults->analyse.i_trellis );
348. H2( "    --no-fast-pskip Disables early SKIP detection on P-frames\n" );
349. H2( "    --no-dct-decimate Disables coefficient thresholding on P-frames\n" );
350. H1( "    --nr <integer> Noise reduction [%d]\n", defaults->analyse.i_noise_reduction );
351. H2( "\n" );
352. H2( "    --deadzone-inter <int> Set the size of the inter luma quantization deadzone [%d]\n", defaults->analyse.i_luma_deadzone[0] );
353. H2( "    --deadzone-intra <int> Set the size of the intra luma quantization deadzone [%d]\n", defaults->analyse.i_luma_deadzone[1] );
354. H2( "    Deadzones should be in the range 0 - 32.\n" );
355. H2( "    --cqm <string> Preset quant matrices [\"flat\"]\n"
356.     "    - jvt, flat\n" );
357. H1( "    --cqmfile <string> Read custom quant matrices from a JM-compatible file\n" );
358. H2( "    Overrides any other --cqm* options.\n" );
359. H2( "    --cqm4 <list> Set all 4x4 quant matrices\n"
360.     "    Takes a comma-separated list of 16 integers.\n" );
361. H2( "    --cqm8 <list> Set all 8x8 quant matrices\n"
362.     "    Takes a comma-separated list of 64 integers.\n" );
363. H2( "    --cqm4i, --cqm4p, --cqm8i, --cqm8p <list>\n"
364.     "    Set both luma and chroma quant matrices\n" );
365. H2( "    --cqm4iy, --cqm4ic, --cqm4py, --cqm4pc <list>\n"
366.     "    Set individual quant matrices\n" );
367. H2( "\n" );
368. H2( "Video Usability Info (Annex E):\n" );
369. H2( "The VUI settings are not used by the encoder but are merely suggestions to\n" );
370. H2( "the playback equipment. See doc/vui.txt for details. Use at your own risk.\n" );
371. H2( "\n" );
372. H2( "    --overscan <string> Specify crop overscan setting [\"%s\"]\n"
373.     "    - undef, show, crop\n",
374.     strttable_lookup( x264_overscan_names, defaults->vui.i_overscan ) );
375. H2( "    --videoformat <string> Specify video format [\"%s\"]\n"
376.     "    - component, pal, ntsc, secam, mac, undef\n",

```



```

377.          strttable_lookup( x264_vidformat_names, defaults->vui.i_vidformat );
378.      H2( "    --range <string> Specify color range [%s%]\n"
379.          "    - %s\n", range_names[0], stringify_names( buf, range_names ) );
380.      H2( "    --colorprim <string> Specify color primaries [%s%]\n"
381.          "    - undef, bt709, bt470m, bt470bg, smpte170m,\n"
382.          "    smpte240m, film, bt2020\n",
383.          strttable_lookup( x264_colorprim_names, defaults->vui.i_colorprim );
384.      H2( "    --transfer <string> Specify transfer characteristics [%s%]\n"
385.          "    - undef, bt709, bt470m, bt470bg, smpte170m,\n"
386.          "    smpte240m, linear, log100, log316,\n"
387.          "    iec61966-2-4, bt1361e, iec61966-2-1,\n"
388.          "    bt2020-10, bt2020-12\n",
389.          strttable_lookup( x264_transfer_names, defaults->vui.i_transfer );
390.      H2( "    --colormatrix <string> Specify color matrix setting [%s%]\n"
391.          "    - undef, bt709, fcc, bt470bg, smpte170m,\n"
392.          "    smpte240m, GBR, YCbCo, bt2020nc, bt2020c\n",
393.          strttable_lookup( x264_colmatrix_names, defaults->vui.i_colmatrix );
394.      H2( "    --chromaloc <integer> Specify chroma sample location (0 to 5) [%d]\n",
395.          defaults->vui.i_chroma_loc );
396.
397.      H2( "    --nal-hrd <string> Signal HRD information (requires vbv-buftype)\n"
398.          "    - none, vbr, cbr (cbr not allowed in .mp4)\n" );
399.      H2( "    --filler Force hard-CBR and generate filler (implied by\n"
400.          "    --nal-hrd cbr)\n" );
401.      H2( "    --pic-struct Force pic_struct in Picture Timing SEI\n" );
402.      H2( "    --crop-rect <string> Add 'left,top,right,bottom' to the bitstream-level\n"
403.          "    cropping rectangle\n" );
404.
405.      H0( "\n" );
406.      H0( "Input/Output:\n" );
407.      H0( "\n" );
408.      H0( "    -o, --output <string> Specify output file\n" );
409.      H1( "    --muxer <string> Specify output container format [%s%]\n"
410.          "    - %s\n", muxer_names[0], stringify_names( buf, muxer_names ) );
411.      H1( "    --demuxer <string> Specify input container format [%s%]\n"
412.          "    - %s\n", demuxer_names[0], stringify_names( buf, demuxer_names ) );
413.      H1( "    --input-fmt <string> Specify input file format (requires lavf support)\n" );
414.      H1( "    --input-csp <string> Specify input colorspace format for raw input\n" );
415.      print_csp_names( longhelp );
416.      H1( "    --output-csp <string> Specify output colorspace [%s%]\n"
417.          "    - %s\n", output_csp_names[0], stringify_names( buf, output_csp_names ) );
418.      H1( "    --input-depth <integer> Specify input bit depth for raw input\n" );
419.      H1( "    --input-range <string> Specify input color range [%s%]\n"
420.          "    - %s\n", range_names[0], stringify_names( buf, range_names ) );
421.      H1( "    --input-res <int> Specify input resolution (width x height)\n" );
422.      H1( "    --index <string> Filename for input index file\n" );
423.      H0( "    --sar width:height Specify Sample Aspect Ratio\n" );
424.      H0( "    --fps <float|rational> Specify framerate\n" );
425.      H0( "    --seek <integer> First frame to encode\n" );
426.      H0( "    --frames <integer> Maximum number of frames to encode\n" );
427.      H0( "    --level <string> Specify level (as defined by Annex A)\n" );
428.      H1( "    --blu-ray-compat Enable compatibility hacks for Blu-ray support\n" );
429.      H1( "    --avcintra-class <integer> Use compatibility hacks for AVC-Intra class\n"
430.          "    - 50, 100, 200\n" );
431.      H1( "    --stitchable Don't optimize headers based on video content\n"
432.          "    Ensures ability to recombine a segmented encode\n" );
433.      H1( "\n" );
434.      H1( "    -v, --verbose Print stats for each frame\n" );
435.      H1( "    --no-progress Don't show the progress indicator while encoding\n" );
436.      H0( "    --quiet Quiet Mode\n" );
437.      H1( "    --log-level <string> Specify the maximum level of logging [%s%]\n"
438.          "    - %s\n", strttable_lookup( log_level_names, cli_log_level - X264_LOG_NONE ),
439.          stringify_names( buf, log_level_names ) );
440.      H1( "    --psnr Enable PSNR computation\n" );
441.      H1( "    --ssim Enable SSIM computation\n" );
442.      H1( "    --threads <integer> Force a specific number of threads\n" );
443.      H2( "    --lookahead-threads <integer> Force a specific number of lookahead threads\n" );
444.      H2( "    --sliced-threads Low-latency but lower-efficiency threading\n" );
445.      H2( "    --thread-input Run Avisynth in its own thread\n" );
446.      H2( "    --sync-lookahead <integer> Number of buffer frames for threaded lookahead\n" );
447.      H2( "    --non-deterministic Slightly improve quality of SMP, at the cost of repeatability\n" );
448.      H2( "    --cpu-independent Ensure exact reproducibility across different cpus,\n"
449.          "    as opposed to letting them select different algorithms\n" );
450.      H2( "    --asm <integer> Override CPU detection\n" );
451.      H2( "    --no-asm Disable all CPU optimizations\n" );
452.      H2( "    --opencl Enable use of OpenCL\n" );
453.      H2( "    --opencl-clbin <string> Specify path of compiled OpenCL kernel cache\n" );
454.      H2( "    --opencl-device <integer> Specify OpenCL device ordinal\n" );
455.      H2( "    --dump-yuv <string> Save reconstructed frames\n" );
456.      H2( "    --sps-id <integer> Set SPS and PPS id numbers [%d]\n", defaults->i_sps_id );
457.      H2( "    --aud Use access unit delimiters\n" );
458.      H2( "    --force-cfr Force constant framerate timestamp generation\n" );
459.      H2( "    --tcfile-in <string> Force timestamp generation with timecode file\n" );
460.      H2( "    --tcfile-out <string> Output timecode v2 file from input timestamps\n" );
461.      H2( "    --timebase <int/int> Specify timebase numerator and denominator\n"
462.          "    <integer> Specify timebase numerator for input timecode file\n"
463.          "    or specify timebase denominator for other input\n" );
464.      H2( "    --dts-compress Eliminate initial delay with container DTS hack\n" );
465.      H0( "\n" );
466.      H0( "Filtering:\n" );
467.      H0( "\n" );

```

```

468.     H0( "      --vf, --video-filter <filter0>/<filter1>/... Apply video filtering to the input file\n" );
469.     H0( "\n" );
470.     H0( "      Filter options may be specified in <filter>:<option>=<value> format.\n" );
471.     H0( "\n" );
472.     H0( "      Available filters:\n" );
473.     x264_register_vid_filters();
474.     x264_vid_filter_help( longhelp );
475.     H0( "\n" );
476. }

```

help()中主要有3个宏定义：H0(), H1()和H2()。这三个宏定义实质上都是printf()。它们之间的区别在于：H0()无论如何都会调用print()；H1()在longhelp大于等于1的时候才会调用print()；而H2()在longhelp等于2时候才会调用print()。

## print\_version\_info()

print\_version\_info()用于打印x264的版本信息。在x264命令行程序中添加“-V”参数后会调用该函数。该函数的定义如下所示。

```

[cpp]
1. //打印版本信息
2. static void print_version_info( void )
3. {
4.     #ifdef X264_POINTVER
5.         printf( "x264 "X264_POINTVER"\n" );
6.     #else
7.         printf( "x264 0.%d.X\n", X264_BUILD );
8.     #endif
9.     #if HAVE_SWSCALE
10.        printf( "(libswscale %d.%d.%d)\n", LIBSWSCALE_VERSION_MAJOR, LIBSWSCALE_VERSION_MINOR, LIBSWSCALE_VERSION_MICRO );
11.    #endif
12.    #if HAVE_LAVF
13.        printf( "(libavformat %d.%d.%d)\n", LIBAVFORMAT_VERSION_MAJOR, LIBAVFORMAT_VERSION_MINOR, LIBAVFORMAT_VERSION_MICRO );
14.    #endif
15.    #if HAVE_FFMS
16.        printf( "
(ffmpegsource %d.%d.%d.%d)\n", FFMS_VERSION >> 24, (FFMS_VERSION & 0xff0000) >> 16, (FFMS_VERSION & 0xff00) >> 8, FFMS_VERSION & 0xff
;
17.    #endif
18.        printf( "built on " __DATE__ ", " );
19.    #ifdef __INTEL_COMPILER
20.        printf( "intel: %.2f (%d)\n", __INTEL_COMPILER / 100.f, __INTEL_COMPILER_BUILD_DATE );
21.    #elif defined( __GNUC__ )
22.        printf( "gcc: " __VERSION__ "\n" );
23.    #elif defined( _MSC_FULL_VER )
24.        printf( "msvc: %.2f (%u)\n", _MSC_VER / 100.f, _MSC_FULL_VER );
25.    #else
26.        printf( "using an unknown compiler\n" );
27.    #endif
28.        printf( "configuration: --bit-depth=%d --chroma-format=%s\n", x264_bit_depth, X264_CHROMA_FORMAT ? (output_csp_names[0]+1) : "all" );
29.        printf( "x264 license: " );
30.        #if HAVE_GPL
31.            printf( "GPL version 2 or later\n" );
32.        #else
33.            printf( "Non-GPL commercial\n" );
34.        #endif
35.        #if HAVE_SWSCALE
36.            const char *license = swscale_license();
37.            printf( "libswscale%s license: %s\n", HAVE_LAVF ? "/libavformat" : "", HAVE_FFMS ? "/ffmpegsource" : "", license );
38.            if( !strcmp( license, "nonfree and unredistributable" ) ||
39.                (!HAVE_GPL && (!strcmp( license, "GPL version 2 or later" )
40.                    || !strcmp( license, "GPL version 3 or later" ) )))
41.                printf( "WARNING: This binary is unredistributable!\n" );
42.        #endif
43.    }

```

该函数定义比较浅显易懂，不再详细记录。

## x264\_param\_parse()

x264\_param\_parse()是一个x264的API。该函数以字符串键值对的方式设置x264\_param\_t结构体的一个成员变量。该函数的声明如下所示。

```

[cpp]
1. /* x264_param_parse:
2.  * set one parameter by name.
3.  * returns 0 on success, or returns one of the following errors.
4.  * note: BAD_VALUE occurs only if it can't even parse the value,
5.  * numerical range is not checked until x264_encoder_open() or
6.  * x264_encoder_reconfig().
7.  * value=NULL means "true" for boolean options, but is a BAD_VALUE for non-booleans. */
8. int x264_param_parse( x264_param_t *, const char *name, const char *value );

```

x264\_param\_parse()的定义如下所示。

x264\_param\_parse()中判断参数的宏OPT()和OPT2()实质上就是strcmp()。由此可见该函数的流程首先是调用strcmp()判断当前输入参数的名称name，然后再调用atoi()，atof()，或者atobool()等将当前输入参数值value转换成相应类型的值并赋值给对应的参数。

## x264\_param\_apply\_profile()

x264\_param\_apply\_profile()是一个x264的API。该函数用于设置x264的profile，它的声明如下所示。

```
/* (can be NULL, in which case the function will do nothing)
 *
 * Does NOT guarantee that the given profile will be used: if the restrictions
 * of "High" are applied to settings that are already Baseline-compatible, the
 * stream will remain baseline. In short, it does not increase settings, only
 * decrease them.
 *
 * returns 0 on success, negative on failure (e.g. invalid profile name). */
int x264_param_apply_profile( x264_param_t *, const char *profile );
```

x264\_param\_apply\_profile()的定义如下所示。

```

//设置profile
int x264_param_apply_profile( x264_param_t *param, const char *profile )
{
    if( !profile )
        return 0;
    //字符串到整型
    int p = profile_string_to_int( profile );
    //检查profile设置是否正确
    if( p < 0 )
    {
        x264_log( NULL, X264_LOG_ERROR, "invalid profile: %s\n", profile );
        return -1;
    }
    if( p < PROFILE_HIGH444_PREDICTIVE && ((param->rc.i_rc_method == X264_RC_CQP && param->rc.i_qp_constant <= 0) ||
        (param->rc.i_rc_method == X264_RC_CRF && (int)(param->rc.f_rf_constant + QP_BD_OFFSET) <= 0)) )
    {
        x264_log( NULL, X264_LOG_ERROR, "%s profile doesn't support lossless\n", profile );
        return -1;
    }
    if( p < PROFILE_HIGH444_PREDICTIVE && (param->i_csp & X264_CSP_MASK) >= X264_CSP_I444 )
    {
        x264_log( NULL, X264_LOG_ERROR, "%s profile doesn't support 4:4:4\n", profile );
        return -1;
    }
    if( p < PROFILE_HIGH422 && (param->i_csp & X264_CSP_MASK) >= X264_CSP_I422 )
    {
        x264_log( NULL, X264_LOG_ERROR, "%s profile doesn't support 4:2:2\n", profile );
        return -1;
    }
    if( p < PROFILE_HIGH10 && BIT_DEPTH > 8 )
    {
        x264_log( NULL, X264_LOG_ERROR, "%s profile doesn't support a bit depth of %d\n", profile, BIT_DEPTH );
        return -1;
    }
    //根据不同的Profile做设置
    //Baseline基本型
    if( p == PROFILE_BASELINE )
    {
        //不支持DCT8x8
        param->analyse.b_transform_8x8 = 0;
        //不使用CABAC
        param->b_cabac = 0;
        param->i_cqm_preset = X264_CQM_FLAT;
        param->psz_cqm_file = NULL;
        //没有B帧
        param->i_bframe = 0;
        //没有加权
        param->analyse.i_weighted_pred = X264_WEIGHTP_NONE;
        //不支持隔行扫描
        if( param->b_interlaced )
        {
            x264_log( NULL, X264_LOG_ERROR, "baseline profile doesn't support interlacing\n" );
            return -1;
        }
        if( param->b_fake_interlaced )
        {
            x264_log( NULL, X264_LOG_ERROR, "baseline profile doesn't support fake interlacing\n" );
            return -1;
        }
    }
    //Main主型
    else if( p == PROFILE_MAIN )
    {
        //不支持DCT8x8
        param->analyse.b_transform_8x8 = 0;
        param->i_cqm_preset = X264_CQM_FLAT;
        param->psz_cqm_file = NULL;
    }
    return 0;
}

```

从定义可以看出，x264\_param\_apply\_profile()首先调用了函数profile\_string\_to\_int()将输入的profile字符串转换为int类型的profile；然后会检查该profile的设置是否合理；最后会根据profile对x264\_param\_t中的参数进行相应的设置。

该函数中调用的profile\_string\_to\_int()的定义如下。

```
static int profile_string_to_int( const char *str )
{
    if( !strcasecmp( str, "baseline" ) )
        return PROFILE_BASELINE;
    if( !strcasecmp( str, "main" ) )
        return PROFILE_MAIN;
    if( !strcasecmp( str, "high" ) )
        return PROFILE_HIGH;
    if( !strcasecmp( str, "high10" ) )
        return PROFILE_HIGH10;
    if( !strcasecmp( str, "high422" ) )
        return PROFILE_HIGH422;
    if( !strcasecmp( str, "high444" ) )
        return PROFILE_HIGH444_PREDICTIVE;
    return -1;
}
```

从定义可以看出profile\_string\_to\_int()根据输入的字符串str返回不同的整型变量。

## select\_output()

select\_output()用于设定输出的文件格式。该函数的定义如下所示。

//根据文件名的后缀确定输出的文件格式（raw H264，flv，mp4...）

```
static int select_output( const char *muxer, char *filename, x264_param_t *param )
{
    //从文件路径字符串中解析出扩展名，存入ext
    //解析的方式就是反向搜索字符"."
    const char *ext = get_filename_extension( filename );

    //strcasecmp(char *s1, char *s2)用于忽略大小写比较字符串.
    //参数s1和s2字符串相等则返回0。s1大于s2则返回大于0 的值，s1 小于s2 则返回小于0的值。

    if( !strcmp( filename, "-" ) || strcmp( muxer, "auto" ) )
        ext = muxer;
    //后缀为"mp4"
    if( !strcasecmp( ext, "mp4" ) )
    {
#ifdef HAVE_GPAC || HAVE_LSMASH
        cli_output = mp4_output;
        param->b_annexb = 0;
        param->b_repeat_headers = 0;
        if( param->i_nal_hrd == X264_NAL_HRD_CBR )
        {
            x264_cli_log( "x264", X264_LOG_WARNING, "cbr nal-hrd is not compatible with mp4\n" );
            param->i_nal_hrd = X264_NAL_HRD_VBR;
        }
    }
#else
        x264_cli_log( "x264", X264_LOG_ERROR, "not compiled with MP4 output support\n" );
        return -1;
#endif
    }
    else if( !strcasecmp( ext, "mkv" ) )
    {
        //设定cli_output_t
        cli_output = mkv_output;
        //不加起始码0x00000001
        param->b_annexb = 0;
        //不再每个Keyframe前面加SPS和PPS
        param->b_repeat_headers = 0;
    }
    else if( !strcasecmp( ext, "flv" ) )
    {
        cli_output = flv_output;
        param->b_annexb = 0;
        param->b_repeat_headers = 0;
    }
    else
        cli_output = raw_output; //不符合上述后缀，则输出裸流
    return 0;
}
```

从函数定义可以看出，select\_output()首先调用get\_filename\_extension()从输入文件路径的字符串中提取出了扩展名，然后根据不同的扩展名设定不同的输出格式。其

中get\_filename\_extension()是一个提取扩展名的函数，定义如下所示。

//根据"."确定文件后缀

```
static inline char *get_filename_extension( char *filename )
{
    char *ext = filename + strlen( filename );
    while( *ext != '.' && ext > filename )
        ext--;
    ext += *ext == '.';
    return ext;
}
```

可以看出get\_filename\_extension()从字符串的末尾开始向前搜索点符号“.”，并且将“.”后面的内容作为提取出来的扩展名。

## select\_input()

select\_input()用于设定输入的文件格式。该函数的定义如下所示。

//设置输入文件的格式 (yuv, y4m...)

```
static int select_input( const char *demuxer, char *used_demuxer, char *filename,
                        hnd_t *p_handle, video_info_t *info, cli_input_opt_t *opt )
{
    int b_auto = !strcasecmp( demuxer, "auto" );
    //从文件路径字符串中解析出扩展名，存入ext
    //解析的方式就是反向搜索字符"."
    const char *ext = b_auto ? get_filename_extension( filename ) : "";
    int b_regular = strcmp( filename, "-" );
    if( !b_regular && b_auto )
        ext = "raw";
    b_regular = b_regular && x264_is_regular_file_path( filename );
    if( b_regular )
    {
        FILE *f = x264_fopen( filename, "r" );
        if( f )
        {
            b_regular = x264_is_regular_file( f );
            fclose( f );
        }
    }
    const char *module = b_auto ? ext : demuxer;

    //strcasecmp(char *s1, char *s2)用于忽略大小写比较字符串.
    //参数s1和s2字符串相等则返回0。s1大于s2则返回大于0 的值，s1 小于s2 则返回小于0的值。

    if( !strcasecmp( module, "avs" ) || !strcasecmp( ext, "d2v" ) || !strcasecmp( ext, "dga" ) )
    {
#ifdef HAVE_AVS
        cli_input = avs_input;
        module = "avs";
#else
        x264_cli_log( "x264", X264_LOG_ERROR, "not compiled with AVS input support\n" );
        return -1;
#endif
    }
    else if( !strcasecmp( module, "y4m" ) )
        cli_input = y4m_input;
    else if( !strcasecmp( module, "raw" ) || !strcasecmp( ext, "yuv" ) )
        cli_input = raw_input;
    else
    {
#ifdef HAVE_FFMS
        if( b_regular && ( b_auto || !strcasecmp( demuxer, "ffms" ) ) &&
            !ffms_input.open_file( filename, p_handle, info, opt ) )
        {
            module = "ffms";
            b_auto = 0;
            cli_input = ffms_input;
        }
#endif
#ifdef HAVE_LAVF
        if( ( b_auto || !strcasecmp( demuxer, "lavf" ) ) &&
            !lavf_input.open_file( filename, p_handle, info, opt ) )
        {
            module = "lavf";
            b_auto = 0;
            cli_input = lavf_input;
        }
#endif
    }
}
```

```

    }
#endif
#if HAVE_AVS
    if( b_regular && (b_auto || !strcasecmp( demuxer, "avs" )) &&
        !avs_input.open_file( filename, p_handle, info, opt ) )
    {
        module = "avs";
        b_auto = 0;
        cli_input = avs_input;
    }
#endif
    if( b_auto && !raw_input.open_file( filename, p_handle, info, opt ) )
    {
        module = "raw";
        b_auto = 0;
        cli_input = raw_input;
    }

    FAIL_IF_ERROR( !(*p_handle), "could not open input file '%s' via any method!\n", filename )
}
strcpy( used_demuxer, module );

return 0;
}

```

从源代码中可以看出，select\_input()首先调用get\_filename\_extension()获取输入文件名的扩展名；然后根据扩展名设置不同的输入格式。

至此x264命令行程序main()函数调用的parse()函数就分析完毕了。下面分析main()函数调用的另一个函数encode()。

## encode()

encode()编码YUV为H.264码流，该函数的定义如下所示。

```

//编码（在内部有一个循环用于一帧一帧编码）
static int encode( x264_param_t *param, cli_opt_t *opt )
{
    x264_t *h = NULL;
    x264_picture_t pic;
    cli_pic_t cli_pic;
    const cli_pulldown_t *pulldown = NULL; // shut up gcc

    int i_frame = 0;
    int i_frame_output = 0;
    int64_t i_end, i_previous = 0, i_start = 0;
    int64_t i_file = 0;
    int i_frame_size;
    int64_t last_dts = 0;
    int64_t prev_dts = 0;
    int64_t first_dts = 0;
    # define MAX_PTS_WARNING 3 /* arbitrary */
    int pts_warning_cnt = 0;
    int64_t largest_pts = -1;
    int64_t second_largest_pts = -1;
    int64_t ticks_per_frame;
    double duration;
    double pulldown_pts = 0;
    int retval = 0;

    opt->b_progress &= param->i_log_level < X264_LOG_DEBUG;

    /* set up pulldown */
    if( opt->i_pulldown && !param->b_vfr_input )
    {
        param->b_pulldown = 1;
        param->b_pic_struct = 1;
        pulldown = &pulldown_values[opt->i_pulldown];
        param->i_timebase_num = param->i_fps_den;
        FAIL_IF_ERROR2( fmod( param->i_fps_num * pulldown->fps_factor, 1 ),
            "unsupported framerate for chosen pulldown\n" )
        param->i_timebase_den = param->i_fps_num * pulldown->fps_factor;
    }
    //打开编码器
    h = x264_encoder_open( param );

```

```

FAIL_IF_ERROR2( !h, "x264_encoder_open failed\n" );
//获得参数
x264_encoder_parameters( h, param );
//一些不是裸流的封装格式（FLV，MP4等）需要一些参数，例如宽高等等
//cli_output_t是代表输出媒体文件的结构体
FAIL_IF_ERROR2( cli_output.set_param( opt->hout, param ), "can't set outfile param\n" );
//计时
i_start = x264_mdate();

/* ticks/frame = ticks/second / frames/second */
ticks_per_frame = (int64_t)param->i_timebase_den * param->i_fps_den / param->i_timebase_num / param->i_fps_num;
FAIL_IF_ERROR2( ticks_per_frame < 1 && !param->b_vfr_input, "ticks_per_frame invalid: %"PRIu64"\n", ticks_per_frame );
ticks_per_frame = X264_MAX( ticks_per_frame, 1 );

//如果不是在每个keyframe前面都增加SPS/PPS/SEI的话，就在整个码流前面加SPS/PPS/SEI
//Header指的就是SPS/PPS/SEI
if( !param->b_repeat_headers )
{
    // Write SPS/PPS/SEI
    x264_nal_t *headers;
    int i_nal;
    //获得文件头（SPS、PPS、SEI）
    FAIL_IF_ERROR2( x264_encoder_headers( h, &headers, &i_nal ) < 0, "x264_encoder_headers failed\n" )
    //把文件头写入输出文件
    FAIL_IF_ERROR2( (i_file = cli_output.write_headers( opt->hout, headers )) < 0, "error writing headers to output file\n" );
}

if( opt->tcfile_out )
    fprintf( opt->tcfile_out, "%# timecode format v2\n" );

/* Encode frames */
//循环进行编码
for( ; !b_ctrl_c && (i_frame < param->i_frame_total || !param->i_frame_total); i_frame++ )
{
    //从输入源中获取1帧YUV数据，存于cli_pic
    //cli_vid_filter_t可以认为是x264一种“扩展”后的输入源，可以在像素域对图像进行拉伸裁剪等工作。
    //原本代表输入源的结构体是cli_input_t
    if( filter.get_frame( opt->hin, &cli_pic, i_frame + opt->i_seek ) )
        break;
    //初始化x264_picture_t结构体pic
    x264_picture_init( &pic );
    //cli_pic到pic
    convert_cli_to_lib_pic( &pic, &cli_pic );

    if( !param->b_vfr_input )
        pic.i_pts = i_frame;

    if( opt->i_pulldown && !param->b_vfr_input )
    {
        pic.i_pic_struct = pulldown->pattern[ i_frame % pulldown->mod ];
        pic.i_pts = (int64_t)( pulldown_pts + 0.5 );
        pulldown_pts += pulldown_frame_duration[pic.i_pic_struct];
    }
    else if( opt->timebase_convert_multiplier )
        pic.i_pts = (int64_t)( pic.i_pts * opt->timebase_convert_multiplier + 0.5 );

    if( pic.i_pts <= largest_pts )
    {
        if( cli_log_level >= X264_LOG_DEBUG || pts_warning_cnt < MAX_PTS_WARNING )
            x264_cli_log( "x264", X264_LOG_WARNING, "non-strictly-monotonic pts at frame %d (%"PRIu64" <= %"PRIu64")\n",
                i_frame, pic.i_pts, largest_pts );
        else if( pts_warning_cnt == MAX_PTS_WARNING )
            x264_cli_log( "x264", X264_LOG_WARNING, "too many nonmonotonic pts warnings, suppressing further ones\n" );
        pts_warning_cnt++;
        pic.i_pts = largest_pts + ticks_per_frame;
    }
}

second_largest_pts = largest_pts;
largest_pts = pic.i_pts;
if( opt->tcfile_out )
    fprintf( opt->tcfile_out, "%.6f\n", pic.i_pts * ((double)param->i_timebase_num / param->i_timebase_den) * 1e3 );

if( opt->qpfile )
    parse_qpfile( opt, &pic, i_frame + opt->i_seek );

```



```

prev_dts = last_dts;
//编码pic中存储的1帧YUV数据
i_frame_size = encode_frame( h, opt->hout, &pic, &last_dts );
if( i_frame_size < 0 )
{
    b_ctrl_c = 1; /* lie to exit the loop */
    retval = -1;
}
else if( i_frame_size )
{
    i_file += i_frame_size;
    i_frame_output++;
    if( i_frame_output == 1 )
        first_dts = prev_dts = last_dts;
}
//释放处理完的YUV数据
if( filter.release_frame( opt->hin, &cli_pic, i_frame + opt->i_seek ) )
    break;

/* update status line (up to 1000 times per input file) */
if( opt->b_progress && i_frame_output )
    i_previous = print_status( i_start, i_previous, i_frame_output, param->i_frame_total, i_file, param, 2 * last_dts - prev_dts - first_dts );
}
/* Flush delayed frames */
//输出编码器中剩余的帧
//x264_encoder_delayed_frames()返回剩余的帧的个数
while( !b_ctrl_c && x264_encoder_delayed_frames( h ) )
{
    prev_dts = last_dts;
    //编码
    //注意第3个参数为NULL
    i_frame_size = encode_frame( h, opt->hout, NULL, &last_dts );
    if( i_frame_size < 0 )
    {
        b_ctrl_c = 1; /* lie to exit the loop */
        retval = -1;
    }
    else if( i_frame_size )
    {
        i_file += i_frame_size;
        i_frame_output++;
        if( i_frame_output == 1 )
            first_dts = prev_dts = last_dts;
    }
    //输出一些统计信息
    if( opt->b_progress && i_frame_output )
        i_previous = print_status( i_start, i_previous, i_frame_output, param->i_frame_total, i_file, param, 2 * last_dts - prev_dts - first_dts );
}
fail:
if( pts_warning_cnt >= MAX_PTS_WARNING && cli_log_level < X264_LOG_DEBUG )
    x264_cli_log( "x264", X264_LOG_WARNING, "%d suppressed nonmonotonic pts warnings\n", pts_warning_cnt-MAX_PTS_WARNING );

/* duration algorithm fails when only 1 frame is output */
if( i_frame_output == 1 )
    duration = (double)param->i_fps_den / param->i_fps_num;
else if( b_ctrl_c )
    duration = (double)(2 * last_dts - prev_dts - first_dts) * param->i_timebase_num / param->i_timebase_den;
else
    duration = (double)(2 * largest_pts - second_largest_pts) * param->i_timebase_num / param->i_timebase_den;
//计时
i_end = x264_mdate();
/* Erase progress indicator before printing encoding stats. */
if( opt->b_progress )
    fprintf( stderr, "
                                \r" );
//关闭编码器
if( h )
    x264_encoder_close( h );
fprintf( stderr, "\n" );

if( b_ctrl_c )
    fprintf( stderr, "aborted at input frame %d, output frame %d\n", opt->i_seek + i_frame, i_frame_output );
//关闭输出文件
cli_output.close_file( opt->hout, largest_pts, second_largest_pts );
opt->hout = NULL;

```

```

if( i_frame_output > 0 )
{
    double fps = (double)i_frame_output * (double)1000000 /
        (double)( i_end - i_start );

    fprintf( stderr, "encoded %d frames, %.2f fps, %.2f kb/s\n", i_frame_output, fps,
        (double) i_file * 8 / ( 1000 * duration ) );
}

return retval;
}

```

从源代码可以梳理出来encode()的流程：

- (1) 调用x264\_encoder\_open()打开H.264编码器。
- (2) 调用x264\_encoder\_parameters()获得当前的参数集x264\_param\_t，用于后续步骤中的一些配置。
- (3) 调用输出格式（H.264裸流、FLV、mp4等）对应cli\_output\_t结构体的set\_param()方法，为输出格式的封装器设定参数。其中参数源自于上一步骤得到的x264\_param\_t。
- (4) 如果不是在每个keyframe前面都增加SPS/PPS/SEI的话，就调用x264\_encoder\_headers()在整个码流前面加SPS/PPS/SEI。
- (5) 进入一个循环中进行一帧一帧的将YUV编码为H.264：
  - a)调用输入格式（YUV、Y4M等）对应的cli\_vid\_filter\_t结构体get\_frame()方法，获取一帧YUV数据。
  - b)调用encode\_frame()编码该帧YUV数据为H.264数据，并且输出出来。该函数内部调用x264\_encoder\_encode()完成编码工作，调用输出格式对应cli\_output\_t结构体的write\_frame()完成了输出工作。
  - c)调用输入格式（YUV、Y4M等）对应的cli\_vid\_filter\_t结构体release\_frame()方法，释放刚才获取的YUV数据。
  - d)调用print\_status()输出一些统计信息。
- (6) 编码即将结束的时候，进入另一个循环，输出编码器中缓存的视频帧：
  - a)不再传递新的YUV数据，直接调用encode\_frame()，将编码器中缓存的剩余几帧数据编码输出出来。
  - b)调用print\_status()输出一些统计信息。
- (7) 调用x264\_encoder\_close()关闭H.264编码器。

encode()的流程中涉及到libx264的几个关键的API，在这里暂时不做详细分析（后续文章中再进行补充）：

x264\_encoder\_open()：打开H.264编码器。  
 x264\_encoder\_headers()：输出SPS/PPS/SEI。  
 x264\_encoder\_encode()：编码一帧数据。  
 x264\_encoder\_close()：关闭H.264编码器。

此外上述流程中涉及到两个比较简单的函数：encode\_frame()和print\_status()。其中encode\_frame()用于编码一帧数据，而print\_status()用于输出一帧数据编码后的统计信息。下文记录一下这两个函数的定义。

## encode\_frame()

encode\_frame()的定义如下。

```

//编码1帧
static int encode_frame( x264_t *h, hnd_t hout, x264_picture_t *pic, int64_t *last_dts )
{
    x264_picture_t pic_out;
    x264_nal_t *nal;
    int i_nal;
    int i_frame_size = 0;
    //编码API
    //编码x264_picture_t为x264_nal_t
    i_frame_size = x264_encoder_encode( h, &nal, &i_nal, pic, &pic_out );

    FAIL_IF_ERROR( i_frame_size < 0, "x264_encoder_encode failed\n" );

    if( i_frame_size )
    {
        //通过cli_output_t中的方法输出
        //输出raw H.264流的话，等同于直接fwrite()
        //其他封装格式，则还需进行一定的封装
        i_frame_size = cli_output.write_frame( hout, nal[0].p_payload, i_frame_size, &pic_out );
        *last_dts = pic_out.i_dts;
    }

    return i_frame_size;
}

```

从源代码可以看出，encode\_frame()内部调用x264\_encoder\_encode()完成编码工作，调用输出格式对应cli\_output\_t结构体的write\_frame()完成了输出工作。其中有关c

li\_output\_t结构体的知识将在后文中记录。

## print\_status()

print\_status()的定义如下。

//打印一些和时间有关的统计信息

```
static int64_t print_status( int64_t i_start, int64_t i_previous, int i_frame, int i_frame_total, int64_t i_file, x264_param_t *param, int64_t last_ts )
{
    char buf[200];
    int64_t i_time = x264_mdate();
    if( i_previous && i_time - i_previous < UPDATE_INTERVAL )
        return i_previous;
    int64_t i_elapsed = i_time - i_start;
    double fps = i_elapsed > 0 ? i_frame * 1000000. / i_elapsed : 0;
    double bitrate;
    if( last_ts )
        bitrate = (double) i_file * 8 / ( (double) last_ts * 1000 * param->i_timebase_num / param->i_timebase_den );
    else
        bitrate = (double) i_file * 8 / ( (double) 1000 * param->i_fps_den / param->i_fps_num );
    if( i_frame_total )
    {
        //形成输出的字符串
        int eta = i_elapsed * (i_frame_total - i_frame) / ((int64_t)i_frame * 1000000);
        sprintf( buf, "x264 [%0.1f%%] %d/%d frames, %0.2f fps, %0.2f kb/s, eta %d:%02d:%02d",
            100. * i_frame / i_frame_total, i_frame, i_frame_total, fps, bitrate,
            eta/3600, (eta/60)%60, eta%60 );
    }
    else
        sprintf( buf, "x264 %d frames: %0.2f fps, %0.2f kb/s", i_frame, fps, bitrate );
    //输出到stderr
    fprintf( stderr, "%s \r", buf+5 );
    //设置到标题栏？
    x264_cli_set_console_title( buf );
    fflush( stderr ); // needed in windows
    return i_time;
}
```

print\_status()的代码不再详细记录，它的输出效果如下图中红框中的文字。

## X264控制台程序中和输入输出相关的结构体

在x264控制台程序中有3个和输入输出相关的结构体：

cli\_output\_t：输出格式对应的结构体。输出格式一般为H.264裸流、FLV、MP4等。

cli\_input\_t：输入格式对应的结构体。输入格式一般为纯YUV像素数据，Y4M格式数据等。

cli\_vid\_filter\_t：输入格式滤镜结构体。滤镜可以对输入数据做一些简单的处理，例如拉伸、裁剪等等（当然滤镜也可以不作任何处理，直接读取输入数据）。

在x264的编码过程中，调用cli\_vid\_filter\_t结构体的get\_frame()读取YUV数据，调用cli\_output\_t的write\_frame()写入数据。下面简单分析一下它们之间的关系。

## cli\_output\_t

x264项目中和cli\_output\_t结构体相关的源代码都位于根目录的output文件夹下。cli\_output\_t的定义位于output/output.h，如下所示。

```
typedef struct
{
    int (*open_file)( char *psz_filename, hnd_t *p_handle, cli_output_opt_t *opt );
    int (*set_param)( hnd_t handle, x264_param_t *p_param );
    int (*write_headers)( hnd_t handle, x264_nal_t *p_nal );
    int (*write_frame)( hnd_t handle, uint8_t *p_nal, int i_size, x264_picture_t *p_picture );
    int (*close_file)( hnd_t handle, int64_t largest_pts, int64_t second_largest_pts );
} cli_output_t;

extern const cli_output_t raw_output;
extern const cli_output_t mkv_output;
extern const cli_output_t mp4_output;
extern const cli_output_t flv_output;
```

从源代码中可以看出，cli\_output\_t中一共包含了open\_file()，set\_param()，write\_headers()，write\_frame()，close\_file()五个接口。在x264中有raw\_output，mkv\_outp

ut, mp4\_output, flv\_output这几个cli\_output\_t结构体，分别对应H.264裸流，MKV，MP4，FLV格式。下面举例看两个结构体：raw\_output和flv\_output。

### raw\_output (H.264裸流的cli\_output\_t结构体)

raw\_output的定义位于output\raw.c，该文件内容如下所示。

```
#include "output.h"

static int open_file( char *psz_filename, hnd_t *p_handle, cli_output_opt_t *opt )
{
    if( !strcmp( psz_filename, "-" ) )
        *p_handle = stdout;
    else if( !( *p_handle = x264_fopen( psz_filename, "w+b" ) ) )
        return -1;

    return 0;
}

static int set_param( hnd_t handle, x264_param_t *p_param )
{
    return 0;
}

static int write_headers( hnd_t handle, x264_nal_t *p_nal )
{
    int size = p_nal[0].i_payload + p_nal[1].i_payload + p_nal[2].i_payload;

    if( fwrite( p_nal[0].p_payload, size, 1, (FILE*)handle ) )
        return size;
    return -1;
}

static int write_frame( hnd_t handle, uint8_t *p_nalu, int i_size, x264_picture_t *p_picture )
{
    if( fwrite( p_nalu, i_size, 1, (FILE*)handle ) )
        return i_size;
    return -1;
}

static int close_file( hnd_t handle, int64_t largest_pts, int64_t second_largest_pts )
{
    if( !handle || handle == stdout )
        return 0;

    return fclose( (FILE*)handle );
}

const cli_output_t raw_output = { open_file, set_param, write_headers, write_frame, close_file };
```

可以看出raw\_output中的函数定义都比较简单，只是封装了fwrite()，fclose()等函数。

### flv\_output (FLV格式的cli\_output\_t结构体)

flv\_output的定义位于output\flv.c，如下所示。

```
const cli_output_t flv_output = { open_file, set_param, write_headers, write_frame, close_file };
```

该文件内容比较多，只举例看一下其中的两个函数：open\_file()和write\_frame()。

#### open\_file()

flv\_output 中的open\_file()的定义如下所示。

```

static int write_header( flv_buffer *c )
{
    flv_put_tag( c, "FLV" ); // Signature
    flv_put_byte( c, 1 ); // Version
    flv_put_byte( c, 1 ); // Video Only
    flv_put_be32( c, 9 ); // DataOffset
    flv_put_be32( c, 0 ); // PreviousTagSize0

    return flv_flush_data( c );
}

static int open_file( char *psz_filename, hnd_t *p_handle, cli_output_opt_t *opt )
{
    *p_handle = NULL;
    flv_hnd_t *p_flv = calloc( 1, sizeof(flv_hnd_t) );
    if( !p_flv )
        return -1;

    p_flv->b_dts_compress = opt->use_dts_compress;

    p_flv->c = flv_create_writer( psz_filename );
    if( !p_flv->c )
        return -1;

    CHECK( write_header( p_flv->c ) );
    *p_handle = p_flv;

    return 0;
}

```

可以看出flv\_output 中的open\_file()中完成了FLV封装格式文件头的创建。

#### write\_frame()

flv\_output 中的write\_frame()的定义如下所示。

```

static int write_frame( hnd_t handle, uint8_t *p_nalu, int i_size, x264_picture_t *p_picture )
{
    flv_hnd_t *p_flv = handle;
    flv_buffer *c = p_flv->c;

#define convert_timebase_ms( timestamp, timebase ) (int64_t)((timestamp) * (timebase) * 1000 + 0.5)

    if( !p_flv->i_framenum )
    {
        p_flv->i_delay_time = p_picture->i_dts * -1;
        if( !p_flv->b_dts_compress && p_flv->i_delay_time )
            x264_cli_log( "flv", X264_LOG_INFO, "initial delay %"PRIu64" ms\n",
                convert_timebase_ms( p_picture->i_pts + p_flv->i_delay_time, p_flv->d_timebase ) );
    }

    int64_t dts;
    int64_t cts;
    int64_t offset;

    if( p_flv->b_dts_compress )
    {
        if( p_flv->i_framenum == 1 )
            p_flv->i_init_delta = convert_timebase_ms( p_picture->i_dts + p_flv->i_delay_time, p_flv->d_timebase );
        dts = p_flv->i_framenum > p_flv->i_delay_frames
            ? convert_timebase_ms( p_picture->i_dts, p_flv->d_timebase )
            : p_flv->i_framenum * p_flv->i_init_delta / (p_flv->i_delay_frames + 1);
        cts = convert_timebase_ms( p_picture->i_pts, p_flv->d_timebase );
    }
    else
    {
        dts = convert_timebase_ms( p_picture->i_dts + p_flv->i_delay_time, p_flv->d_timebase );
        cts = convert_timebase_ms( p_picture->i_pts + p_flv->i_delay_time, p_flv->d_timebase );
    }
    offset = cts - dts;

    if( p_flv->i_framenum )
    {
        if( p_flv->i_prev_dts == dts )
            x264_cli_log( "flv", X264_LOG_WARNING, "duplicate DTS %"PRIu64" generated by rounding\n",
                "          decoding framerate cannot exceed 1000fps\n", dts );
        if( p_flv->i_prev_cts == cts )

```

```

    if( p_flv->i_prev_cts == cts )
        x264_cli_log( "flv", X264_LOG_WARNING, "duplicate CTS %"PRIu64" generated by rounding\n",
            "composition framerate cannot exceed 1000fps\n", cts );
}
p_flv->i_prev_dts = dts;
p_flv->i_prev_cts = cts;

// A new frame - write packet header
flv_put_byte( c, FLV_TAG_TYPE_VIDEO );
flv_put_be24( c, 0 ); // calculated later
flv_put_be24( c, dts );
flv_put_byte( c, dts >> 24 );
flv_put_be24( c, 0 );

p_flv->start = c->d_cur;
flv_put_byte( c, p_picture->b_keyframe ? FLV_FRAME_KEY : FLV_FRAME_INTER );
flv_put_byte( c, 1 ); // AVC NALU
flv_put_be24( c, offset );

if( p_flv->sei )
{
    flv_append_data( c, p_flv->sei, p_flv->sei_len );
    free( p_flv->sei );
    p_flv->sei = NULL;
}
flv_append_data( c, p_nalu, i_size );

unsigned length = c->d_cur - p_flv->start;
flv_rewrite_amf_be24( c, length, p_flv->start - 10 );
flv_put_be32( c, 11 + length ); // Last tag size
CHECK( flv_flush_data( c ) );

p_flv->i_framenum++;

return i_size;
}

```

flv\_output 中的可以看出write\_frame()中完成了FLV封装格式中一个Tag单元的创建。

## cli\_input\_t

x264项目中和cli\_input\_t结构体相关的源代码都位于根目录的input文件夹下。cli\_input\_t的定义位于input/input.h，如下所示。

```

typedef struct
{
    int (*open_file)( char *psz_filename, hnd_t *p_handle, video_info_t *info, cli_input_opt_t *opt );
    int (*picture_alloc)( cli_pic_t *pic, int csp, int width, int height );
    int (*read_frame)( cli_pic_t *pic, hnd_t handle, int i_frame );
    int (*release_frame)( cli_pic_t *pic, hnd_t handle );
    void (*picture_clean)( cli_pic_t *pic );
    int (*close_file)( hnd_t handle );
} cli_input_t;

extern const cli_input_t raw_input;
extern const cli_input_t y4m_input;
extern const cli_input_t avs_input;
extern const cli_input_t lavf_input;
extern const cli_input_t ffms_input;

```

从源代码中可以看出，cli\_input\_t中一共包含了open\_file(), picture\_alloc(), read\_frame(), release\_frame(), picture\_clean(), close\_file()六个接口。在x264中有raw\_input, y4m\_input, avs\_input, lavf\_input, ffms\_input这几个cli\_output\_t结构体，分别对应H.264裸流，Y4M，AVS，LAVF，FFMS格式（后几种没有接触过）。下面举例看两个结构体：raw\_input和y4m\_input。

### raw\_input（纯YUV像素数据的cli\_input\_t结构体）

raw\_input的定义位于input/raw.c，该文件内容如下所示。

```

#include "input.h"
#define FAIL_IF_ERROR( cond, ... ) FAIL_IF_ERR( cond, "raw", __VA_ARGS__ )

typedef struct
{
    FILE *fh;
    int next_frame;
    uint64_t plane_size[4];
    uint64_t frame_size;
} raw_input_t;

```

```

uint64_t frame_size;
int bit_depth;
} raw_hnd_t;

//打开raw YUV格式文件
static int open_file( char *psz_filename, hnd_t *p_handle, video_info_t *info, cli_input_opt_t *opt )
{
    raw_hnd_t *h = calloc( 1, sizeof(raw_hnd_t) );
    if( !h )
        return -1;

    if( !opt->resolution )
    {
        //如果没有设置分辨率
        //尝试从文件名中解析分辨率
        /* try to parse the file name */
        for( char *p = psz_filename; *p; p++ )
            if( *p >= '0' && *p <= '9' && sscanf( p, "%dx%d", &info->width, &info->height ) == 2 )
                break;
    }
    else
        sscanf( opt->resolution, "%dx%d", &info->width, &info->height );
    //没有分辨率信息的话，会弹出错误信息
    FAIL_IF_ERROR( !info->width || !info->height, "raw input requires a resolution.\n" )
    //设置颜色空间
    if( opt->colorspace )
    {
        for( info->csp = X264_CSP_CLI_MAX-1; info->csp > X264_CSP_NONE; info->csp-- )
        {
            if( x264_cli_csps[info->csp].name && !strcasecmp( x264_cli_csps[info->csp].name, opt->colorspace ) )
                break;
        }
        FAIL_IF_ERROR( info->csp == X264_CSP_NONE, "unsupported colorspace `%s'\n", opt->colorspace );
    }
    else /* default */
        info->csp = X264_CSP_I420;//默认为YUV420P
    //颜色位深
    h->bit_depth = opt->bit_depth;
    FAIL_IF_ERROR( h->bit_depth < 8 || h->bit_depth > 16, "unsupported bit depth `%d'\n", h->bit_depth );
    if( h->bit_depth > 8 )
        info->csp |= X264_CSP_HIGH_DEPTH;

    if( !strcmp( psz_filename, "-" ) )
        h->fh = stdin; //从管道输入
    else
        h->fh = x264_fopen( psz_filename, "rb" ); //打开文件
    if( h->fh == NULL )
        return -1;

    info->thread_safe = 1;
    info->num_frames = 0;
    info->vfr = 0;

    const x264_cli_csp_t *csp = x264_cli_get_csp( info->csp );
    for( int i = 0; i < csp->planes; i++ )
    {
        h->plane_size[i] = x264_cli_pic_plane_size( info->csp, info->width, info->height, i );
        h->frame_size += h->plane_size[i];
        /* x264_cli_pic_plane_size returns the size in bytes, we need the value in pixels from here on */
        h->plane_size[i] /= x264_cli_csp_depth_factor( info->csp );
    }

    if( x264_is_regular_file( h->fh ) )
    {
        fseek( h->fh, 0, SEEK_END );
        uint64_t size = ftell( h->fh );
        fseek( h->fh, 0, SEEK_SET );
        info->num_frames = size / h->frame_size;
    }

    *p_handle = h;
    return 0;
}

//读取一帧数据-内部

```

```

static int read_frame_internal( cli_pic_t *pic, raw_hnd_t *h, int bit_depth_uc )
{
    int error = 0;
    int pixel_depth = x264_cli_csp_depth_factor( pic->img.csp );
    //一个分量一个分量读
    for( int i = 0; i < pic->img.planes && !error; i++ )
    {
        //fread()读取
        error |= fread( pic->img.plane[i], pixel_depth, h->plane_size[i], h->fh ) != h->plane_size[i];
        if( bit_depth_uc )
        {
            /* upconvert non 16bit high depth planes to 16bit using the same
             * algorithm as used in the depth filter. */
            uint16_t *plane = (uint16_t*)pic->img.plane[i];
            uint64_t pixel_count = h->plane_size[i];
            int lshift = 16 - h->bit_depth;
            for( uint64_t j = 0; j < pixel_count; j++ )
                plane[j] = plane[j] << lshift;
        }
    }
    return error;
}

//读取一帧数据
static int read_frame( cli_pic_t *pic, hnd_t handle, int i_frame )
{
    raw_hnd_t *h = handle;

    if( i_frame > h->next_frame )
    {
        if( x264_is_regular_file( h->fh ) )
            fseek( h->fh, i_frame * h->frame_size, SEEK_SET ); //fseek()。偏移量=帧序号*帧大小。
        else
            while( i_frame > h->next_frame )
            {
                //读取一帧数据-内部
                if( read_frame_internal( pic, h, 0 ) )
                    return -1;
                h->next_frame++;
            }
    }

    if( read_frame_internal( pic, h, h->bit_depth & 7 ) )
        return -1;

    h->next_frame = i_frame+1;
    return 0;
}

//关闭文件
static int close_file( hnd_t handle )
{
    raw_hnd_t *h = handle;
    if( !h || !h->fh )
        return 0;
    //fclose()关闭文件
    fclose( h->fh );
    free( h );
    return 0;
}

//raw格式对应的数组
const cli_input_t raw_input = { open_file, x264_cli_pic_alloc, read_frame, NULL, x264_cli_pic_clean, close_file };

```

从源代码中可以看出,raw\_input 中的open\_file()函数在打开YUV像素数据的时候,会首先判断是否设置了宽和高(YUV是纯像素数据,没有宽和高信息),如果没有设置,则会尝试从文件路径中解析宽和高信息。如果成功完成上述步骤,open\_file()就会调用x264\_fopen()打开输入文件。其他的函数在源代码中都写了注释,就不再重复记录了。

#### y4m\_input (Y4M格式的cli\_input\_t结构体)

y4m\_input的定义位于inputy4m.c,如下所示。

```
const cli_input_t y4m_input = { open_file, x264_cli_pic_alloc, read_frame, NULL, x264_cli_pic_clean, close_file };
```

该文件内容较多,不再进行详细分析。在这里看一个打开文件的函数open\_file()。该函数的定义如下所示。



```

typedef struct
{
    FILE *fh;
    int next_frame;
    int seq_header_len;
    int frame_header_len;
    uint64_t frame_size;
    uint64_t plane_size[3];
    int bit_depth;
} y4m_hnd_t;

#define Y4M_MAGIC "YUV4MPEG2"
#define MAX_YUV4_HEADER 80
#define Y4M_FRAME_MAGIC "FRAME"
#define MAX_FRAME_HEADER 80

static int parse_csp_and_depth( char *csp_name, int *bit_depth )
{
    int csp    = X264_CSP_MAX;

    /* Set colorspace from known variants */
    if( !strcmp( "420", csp_name, 3 ) )
        csp = X264_CSP_I420;
    else if( !strcmp( "422", csp_name, 3 ) )
        csp = X264_CSP_I422;
    else if( !strcmp( "444", csp_name, 3 ) && strcmp( "444alpha", csp_name, 8 ) ) // only accept alphaless 4:4:4
        csp = X264_CSP_I444;

    /* Set high bit depth from known extensions */
    if( sscanf( csp_name, "%d%*[pP]%d", bit_depth ) != 1 )
        *bit_depth = 8;

    return csp;
}

static int open_file( char *psz_filename, hnd_t *p_handle, video_info_t *info, cli_input_opt_t *opt )
{
    y4m_hnd_t *h = malloc( sizeof(y4m_hnd_t) );
    int i;
    uint32_t n, d;
    char header[MAX_YUV4_HEADER+10];
    char *tokend, *header_end;
    int colorspace = X264_CSP_NONE;
    int alt_colorspace = X264_CSP_NONE;
    int alt_bit_depth = 8;
    if( !h )
        return -1;

    h->next_frame = 0;
    info->vfr = 0;

    if( !strcmp( psz_filename, "-" ) )
        h->fh = stdin;
    else
        h->fh = x264_fopen(psz_filename, "rb");
    if( h->fh == NULL )
        return -1;

    h->frame_header_len = strlen( Y4M_FRAME_MAGIC )+1;

    /* Read header */
    //解析Y4M格式的文件头
    for( i = 0; i < MAX_YUV4_HEADER; i++ )
    {
        header[i] = fgetc( h->fh );
        if( header[i] == '\n' )
        {
            /* Add a space after last option. Makes parsing "444" vs
             "444alpha" easier. */
            header[i+1] = 0x20;
            header[i+2] = 0;
            break;
        }
    }
}
if( i == MAX_YUV4_HEADER || strcmp( header, Y4M_MAGIC, strlen( Y4M_MAGIC ) ) )

```

```

return -1;

/* Scan properties */
header_end = &header[i+1]; /* Include space */
h->seq_header_len = i+1;
for( char *tokstart = &header[strlen( Y4M_MAGIC )+1]; tokstart < header_end; tokstart++ )
{
    if( *tokstart == 0x20 )
        continue;
    switch( *tokstart++ )
    {
        case 'W': /* Width. Required. */
            info->width = strtol( tokstart, &tokend, 10 );
            tokstart=tokend;
            break;
        case 'H': /* Height. Required. */
            info->height = strtol( tokstart, &tokend, 10 );
            tokstart=tokend;
            break;
        case 'C': /* Color space */
            colorspace = parse_csp_and_depth( tokstart, &h->bit_depth );
            tokstart = strchr( tokstart, 0x20 );
            break;
        case 'I': /* Interlace type */
            switch( *tokstart++ )
            {
                case 't':
                    info->interlaced = 1;
                    info->tff = 1;
                    break;
                case 'b':
                    info->interlaced = 1;
                    info->tff = 0;
                    break;
                case 'm':
                    info->interlaced = 1;
                    break;
                /*case '?':
                //case 'p':
                default:
                    break;
            }
            break;
        case 'F': /* Frame rate - 0:0 if unknown */
            if( sscanf( tokstart, "%u:%u", &n, &d ) == 2 && n && d )
            {
                x264_reduce_fraction( &n, &d );
                info->fps_num = n;
                info->fps_den = d;
            }
            tokstart = strchr( tokstart, 0x20 );
            break;
        case 'A': /* Pixel aspect - 0:0 if unknown */
            /* Don't override the aspect ratio if sar has been explicitly set on the commandline. */
            if( sscanf( tokstart, "%u:%u", &n, &d ) == 2 && n && d )
            {
                x264_reduce_fraction( &n, &d );
                info->sar_width = n;
                info->sar_height = d;
            }
            tokstart = strchr( tokstart, 0x20 );
            break;
        case 'X': /* Vendor extensions */
            if( !strncmp( "YSCSS=", tokstart, 6 ) )
            {
                /* Older nonstandard pixel format representation */
                tokstart += 6;
                alt_colorspace = parse_csp_and_depth( tokstart, &alt_bit_depth );
            }
            tokstart = strchr( tokstart, 0x20 );
            break;
    }
}
}

```

```

/* colorspace == Y264_CSP_NONE */

```

```

if( colorspace == X264_CSP_NONE )
{
    colorspace = alt_colorspace;
    h->bit_depth = alt_bit_depth;
}

// default to 8bit 4:2:0 if nothing is specified
if( colorspace == X264_CSP_NONE )
{
    colorspace = X264_CSP_I420;
    h->bit_depth = 8;
}

FAIL_IF_ERROR( colorspace <= X264_CSP_NONE || colorspace >= X264_CSP_MAX, "colorspace unhandled\n" )
FAIL_IF_ERROR( h->bit_depth < 8 || h->bit_depth > 16, "unsupported bit depth `%d`\n", h->bit_depth );

info->thread_safe = 1;
info->num_frames = 0;
info->csp = colorspace;
h->frame_size = h->frame_header_len;

if( h->bit_depth > 8 )
    info->csp |= X264_CSP_HIGH_DEPTH;

const x264_cli_csp_t *csp = x264_cli_get_csp( info->csp );

for( i = 0; i < csp->planes; i++ )
{
    h->plane_size[i] = x264_cli_pic_plane_size( info->csp, info->width, info->height, i );
    h->frame_size += h->plane_size[i];
    /* x264_cli_pic_plane_size returns the size in bytes, we need the value in pixels from here on */
    h->plane_size[i] /= x264_cli_csp_depth_factor( info->csp );
}

/* Most common case: frame_header = "FRAME" */
if( x264_is_regular_file( h->fh ) )
{
    uint64_t init_pos = ftell( h->fh );
    fseek( h->fh, 0, SEEK_END );
    uint64_t i_size = ftell( h->fh );
    fseek( h->fh, init_pos, SEEK_SET );
    info->num_frames = (i_size - h->seq_header_len) / h->frame_size;
}

*p_handle = h;
return 0;
}

```

从源代码可以看出，y4m\_input中的open\_file()完成了Y4M文件的打开和文件头解析的功能。

## cli\_vid\_filter\_t

x264项目中cli\_vid\_filter\_t结构体相关的源代码都位于根目录的filters文件夹下。cli\_vid\_filter\_t的定义位于filters\videolvideo.h，如下所示。

```

struct cli_vid_filter_t
{
    /* name of the filter */
    const char *name;
    /* help: a short message on what the filter does and how to use it.
     * this should only be implemented by filters directly accessible by the user */
    void (*help)( int longhelp );
    /* init: initializes the filter given the input clip properties and parameter to adjust them as necessary
     * with the given options provided by the user.
     * returns 0 on success, nonzero on error. */
    int (*init)( hnd_t *handle, cli_vid_filter_t *filter, video_info_t *info, x264_param_t *param, char *opt_string );
    /* get_frame: given the storage for the output frame and desired frame number, generate the frame accordingly.
     * the image data returned by get_frame should be treated as const and not be altered.
     * returns 0 on success, nonzero on error. */
    int (*get_frame)( hnd_t handle, cli_pic_t *output, int frame );
    /* release_frame: frame is done being used and is signaled for cleanup.
     * returns 0 on success, nonzero on error. */
    int (*release_frame)( hnd_t handle, cli_pic_t *pic, int frame );
    /* free: run filter cleanup procedures. */
    void (*free)( hnd_t handle );
    /* next registered filter, unused by filters themselves */
    cli_vid_filter_t *next;
};

```

从源代码中可以看出，cli\_vid\_filter\_t中一共包含了help(), init(), get\_frame(), release\_frame(), free()几个接口。下面举例看两个Filter结构体：

source\_filter：不作任何处理。

resize\_filter：拉伸。

#### source\_filter（没有功能的cli\_vid\_filter\_t结构体）

source\_filter的定义位于filters\video\source.c，该文件内容如下所示。

```

#include "video.h"

/* This filter converts the demuxer API into the filtering API for video frames.
 * Backseeking is prohibited here as not all demuxers are capable of doing so. */

typedef struct
{
    cli_pic_t pic;
    hnd_t hin;
    int cur_frame;
} source_hnd_t;

cli_vid_filter_t source_filter;

static int init( hnd_t *handle, cli_vid_filter_t *filter, video_info_t *info, x264_param_t *param, char *opt_string )
{
    source_hnd_t *h = calloc( 1, sizeof(source_hnd_t) );
    if( !h )
        return -1;
    h->cur_frame = -1;

    if( cli_input.picture_alloc( &h->pic, info->csp, info->width, info->height ) )
        return -1;

    h->hin = *handle;
    *handle = h;
    *filter = source_filter;

    return 0;
}

static int get_frame( hnd_t handle, cli_pic_t *output, int frame )
{
    source_hnd_t *h = handle;
    /* do not allow requesting of frames from before the current position */
    if( frame <= h->cur_frame || cli_input.read_frame( &h->pic, h->hin, frame ) )
        return -1;
    h->cur_frame = frame;
    *output = h->pic;
    return 0;
}

static int release_frame( hnd_t handle, cli_pic_t *pic, int frame )
{
    source_hnd_t *h = handle;
    if( cli_input.release_frame && cli_input.release_frame( &h->pic, h->hin ) )
        return -1;
    return 0;
}

static void free_filter( hnd_t handle )
{
    source_hnd_t *h = handle;
    cli_input.picture_clean( &h->pic );
    cli_input.close_file( h->hin );
    free( h );
}

cli_vid_filter_t source_filter = { "source", NULL, init, get_frame, release_frame, free_filter, NULL };

```

从源代码中可以看出，source\_filter的get\_frame()直接调用了cli\_input\_t的read\_frame()；而它的release\_frame()也是直接调用了cli\_input\_t的release\_frame()。简而言之，source\_filter相当于是一个cli\_input\_t。

#### resize\_filter（拉伸功能对应的cli\_vid\_filter\_t结构体）

resize\_filter的定义位于filters\video\resize.c，该结构体定义如下。

```
cli_vid_filter_t resize_filter = { NAME, help, init, get_frame, release_frame, free_filter, NULL };
```

由于resize\_filter涉及到的代码比较多，在这里仅看一下它的get\_frame()的定义。

```

static int get_frame( hnd_t handle, cli_pic_t *output, int frame )
{
    resizer_hnd_t *h = handle;
    if( h->prev_filter.get_frame( h->prev_hnd, output, frame ) )
        return -1;
    if( h->variable_input && check_resizer( h, output ) )
        return -1;
    h->working = 1;
    if( h->pre_swap_chroma )
        XCHG( uint8_t*, output->img.plane[1], output->img.plane[2] );
    if( h->ctx )
    {
        sws_scale( h->ctx, (const uint8_t* const*)output->img.plane, output->img.stride,
            0, output->img.height, h->buffer.img.plane, h->buffer.img.stride );
        output->img = h->buffer.img; /* copy img data */
    }
    else
        output->img.csp = h->dst_csp;
    if( h->post_swap_chroma )
        XCHG( uint8_t*, output->img.plane[1], output->img.plane[2] );

    return 0;
}

```

可以看出resize\_filter中调用了libswscale类库中的sws\_scale()对图像完成了拉伸工作。  
注：拉伸滤镜需要libswscale类库的支持。

至此cli\_output\_t, cli\_input\_t, cli\_vid\_filter\_t这3个在x264中与输入输出有关的结构体的源代码就分析完毕了。有关x264命令行工具的源代码分析工作也就做完了。下一篇文章开始对libx264内部的源代码进行分析。

**雷霄骅**

**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/45583217>

文章标签： [x264](#) [H.264](#) [视频编码](#) [源代码](#) [YUV](#)

个人分类： [x264](#)

所属专栏： [开源多媒体项目源代码分析](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com