

## 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：11954

=====

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）](#)

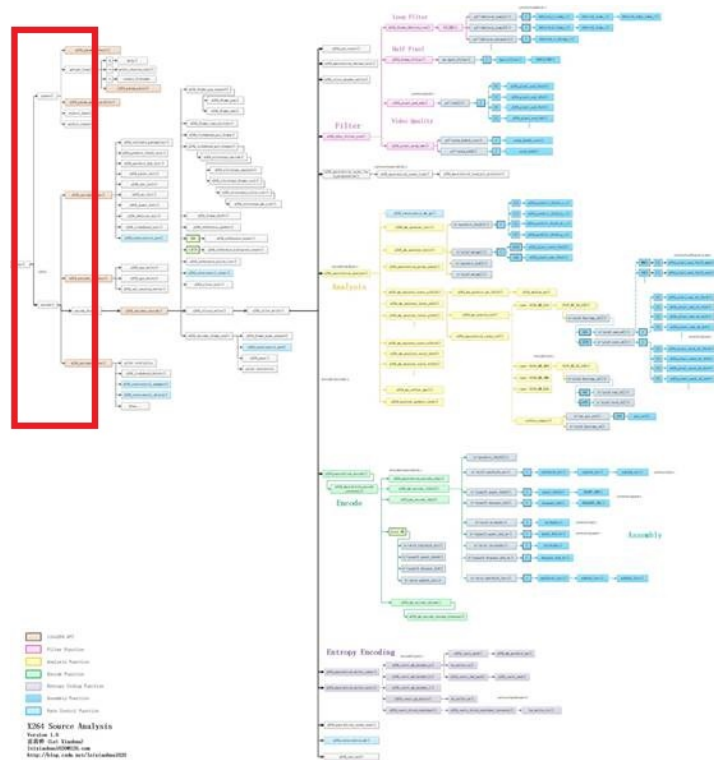
[FFmpeg的H.264解码器源代码简单分析：环路滤波（Loop Filter）部分](#)

=====

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。

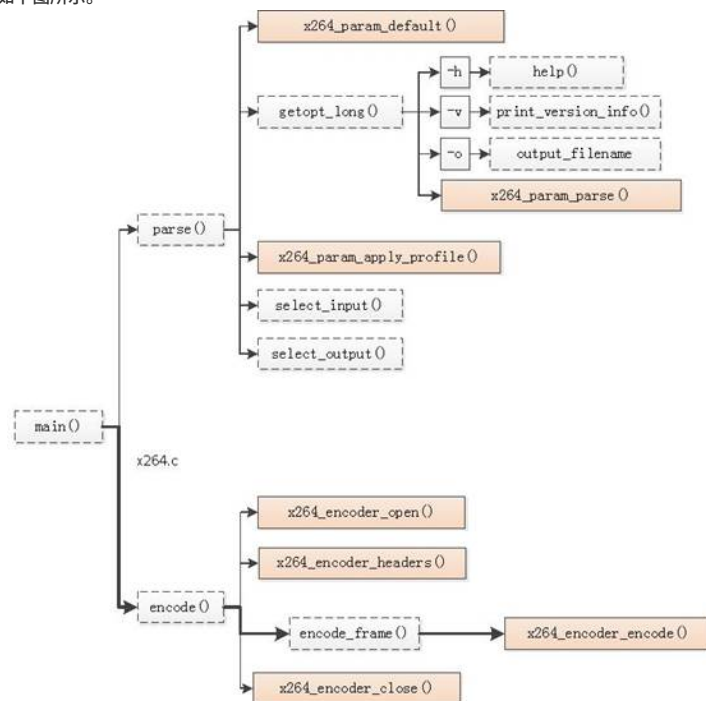
## 函数调用关系图

X264命令行工具的源代码在x264中的位置如下图所示。



单击查看更清晰的图片

X264命令行工具的源代码的调用关系如下图所示。



X264 Source Analysis - x264.c  
 雷霄骅 (Lei Xiaohua)  
 leixiaohual020@126.com  
<http://blog.csdn.net/leixiaohual020>

单击查看更清晰的图片

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s

elect\_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;
```

```

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;
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.     #if X264_CHROMA_FORMAT
181.         static const uint8_t output_csp_fix[] = { X264_CHROMA_FORMAT, X264_CSP_RGB };
182.     #else
183.         static const uint8_t output_csp_fix[] = { X264_CSP_I420, X264_CSP_I422, X264_CSP_I444, X264_CSP_RGB };
184.     #endif
185.         param->i_csp = output_csp = output_csp_fix[output_csp];
186.         break;

```

```

187.         case OPT_INPUT_RANGE:
188.             FAIL_IF_ERROR( parse_enum_value( optarg, range_names, &input_opt.input_range ), "Unknown input range '%s'\n", optarg
)
189.             input_opt.input_range += RANGE_AUTO;
190.             break;
191.         case OPT_RANGE:
192.             FAIL_IF_ERROR( parse_enum_value( optarg, range_names, ¶m->vui.b_fullrange ), "Unknown range '%s'\n", optarg );
193.             input_opt.output_range = param->vui.b_fullrange += RANGE_AUTO;
194.             break;
195.         default:
196. generic_option:
197.         {
198.             if( long_options_index < 0 )
199.             {
200.                 for( int i = 0; long_options[i].name; i++ )
201.                 {
202.                     if( long_options[i].val == c )
203.                     {
204.                         long_options_index = i;
205.                         break;
206.                     }
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
ename )
242.     //输入文件路径
243.     input_filename = argv[optind++];
244.     video_info_t info = {0};
245.     char demuxername[5];
246.
247.     /* set info flags to be overwritten by demuxer as necessary. */
248.     //设置info结构体
249.     info.csp = param->i_csp;
250.     info.fps_num = param->i_fps_num;
251.     info.fps_den = param->i_fps_den;
252.     info.fullrange = input_opt.input_range == RANGE_PC;
253.     info.interlaced = param->b_interlaced;
254.     if( param->vui.i_sar_width > 0 && param->vui.i_sar_height > 0 )
255.     {
256.         info.sar_width = param->vui.i_sar_width;
257.         info.sar_height = param->vui.i_sar_height;
258.     }
259.     info.tff = param->b_tff;
260.     info.vfr = param->b_vfr_input;
261.
262.     input_opt.seek = opt->i_seek;
263.     input_opt.progress = opt->b_progress;
264.     input_opt.output_csp = output_csp;
265.     //设置输入文件的格式(yuv, y4m...)
266.     if( select_input( demuxer, demuxername, input_filename, &opt->hin, &info, &input_opt ) )
267.         return -1;
268.
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 --cfl\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.     return 0;
400. }
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_openc1 = 0;
138.     param->i_openc1_device = 0;
139.     param->openc1_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
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( !strcmp( 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( !strcmp( 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( !strcmp( 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( !strcmp( 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( !strcmp( 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( !strcmp( preset, "medium" ) )
69. {
70.     /* Default is medium */
71. }
72. else if( !strcmp( 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( !strcmp( 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( !strcmp( 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。

```

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。

```

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 Avisynth if compiled with support (%s).\n"
17.         " " or libav* formats if compiled with lavf support (%s) or ffms support (%s).\n"
18.         "Outfile 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"

```

```

20.         output_bit_depth = (configured_at_compile_time)
21.
22.         "\n"
23.         "Options:\n"
24.         "\n"
25.         "  -h, --help           List basic options\n"
26.         "  --longhelp          List more options\n"
27.         "  --fullhelp         List all options\n"
28.         "\n",
29.         X264_BUILD, X264_VERSION,
30.
31. #if HAVE_AVS
32.         "yes",
33. #else
34.         "no",
35. #endif
36. #if HAVE_LAVF
37.         "yes",
38. #else
39.         "no",
40. #endif
41. #if HAVE_FFMS
42.         "yes",
43. #else
44.         "no",
45. #endif
46. #if HAVE_GPAC
47.         "gpac",
48. #elif HAVE_LSMASH
49.         "lsmash",
50. #else
51.         "no",
52. #endif
53.
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-buFSIZE 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.
79.     H2(
80. #if X264_CHROMA_FORMAT <= X264_CSP_I420
81. #if BIT_DEPTH==8
82.         "          - baseline:\n"
83.         "          --no-8x8dct --bframes 0 --no-cabac\n"
84.         "          --cqm flat --weightp 0\n"
85.         "          No interlaced.\n"
86.         "          No lossless.\n"
87.         "          - main:\n"
88.         "          --no-8x8dct --cqm flat\n"
89.         "          No lossless.\n"
90.         "          - high:\n"
91.         "          No lossless.\n"
92. #endif
93.         "          - high10:\n"
94.         "          No lossless.\n"
95.         "          Support for bit depth 8-10.\n"
96. #endif
97. #if X264_CHROMA_FORMAT <= X264_CSP_I422
98.         "          - high422:\n"
99.         "          No lossless.\n"
100.         "          Support for bit depth 8-10.\n"
101.         "          Support for 4:2:0/4:2:2 chroma subsampling.\n"
102. #endif
103.         "          - high444:\n"
104.         "          Support for bit depth 8-10.\n"
105.         "          Support for 4:2:0/4:2:2/4:4:4 chroma subsampling.\n" );
106.     else H0(
107.         "          - "
108.
109. #if X264_CHROMA_FORMAT <= X264_CSP_I420
110. #if BIT_DEPTH==8
111.         "baseline,main,high,"
112. #endif
113.         "high10,"
114. #endif
115. #if X264_CHROMA_FORMAT <= X264_CSP_I422
116.         "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. strttable_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) [%f]\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 [%f]\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 [%f]\n", defaults->rc.f_rate_tolerance );
267. H2( " --ipratio <float> QP factor between I and P [%f]\n", defaults->rc.f_ip_factor );
268. H2( " --pbratio <float> QP factor between P and B [%f]\n", defaults->rc.f_pb_factor );
269. H2( " --chroma-qp-offset <integer> QP difference between chroma and luma [%d]\n", defaults->analyse.i_chroma_qp_offset );
270. H2( " --aq-mode <integer> AQ method [%d]\n"
271. " - 0: Disabled\n"
272. " - 1: Variance AQ (complexity mask)\n"
273. " - 2: Auto-variance AQ (experimental)\n", defaults->rc.i_aq_mode );
274. H1( " --aq-strength <float> Reduces blocking and blurring in flat and\n"
275. " textured areas. [%f]\n", defaults->rc.f_aq_strength );
276. H1( "\n" );
277. H0( " -p, --pass <integer> Enable multipass ratecontrol\n"
278. " - 1: First pass, creates stats file\n"
279. " - 2: Last pass, does not overwrite stats file\n" );
280. H2( " - 3: Nth pass, overwrites 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 [%f]\n", defaults->rc.f_qcompress );
284. H2( " --cplxblur <float> Reduce fluctuations in QP (before curve compression) [%f]\n", defaults->rc.f_complexity_blur );
285. H2( " --qblur <float> Reduce fluctuations in QP (after curve compression) [%f]\n", defaults->rc.f_qblur );
286. H2( " --zones <zone0>/<zone1>/... Tweak the bitrate of regions of the video\n" );
287. H2( " 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: framenum 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.         strtable_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.         strtable_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( " --mvrange <integer> Maximum motion vector length [-1 (auto)]\n" );
322.     H2( " --mvrange-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->
353.         analyse.i_luma_deadzone[0] );
354.     H2( " --deadzone-intra <int> Set the size of the intra luma quantization deadzone [%d]\n", defaults->
355.         analyse.i_luma_deadzone[1] );
356.     H2( " Deadzones should be in the range 0 - 32.\n" );
357.     H2( " --cqm <string> Preset quant matrices [\"flat\"]\n"
358.         "                                - jvt, flat\n" );
359.     H1( " --cqmfile <string> Read custom quant matrices from a JM-compatible file\n" );
360.     H2( " Overrides any other --cqm* options.\n" );
361.     H2( " --cqm4 <list> Set all 4x4 quant matrices\n"
362.         "                                Takes a comma-separated list of 16 integers.\n" );
363.     H2( " --cqm8 <list> Set all 8x8 quant matrices\n"
364.         "                                Takes a comma-separated list of 64 integers.\n" );
365.     H2( " --cqm4i, --cqm4p, --cqm8i, --cqm8p <list>\n"
366.         "                                Set both luma and chroma quant matrices\n" );
367.     H2( " --cqm4iy, --cqm4ic, --cqm4py, --cqm4pc <list>\n"
368.         "                                Set individual quant matrices\n" );
369.     H2( "\n" );
370.     H2( "Video Usability Info (Annex E):\n" );
371.     H2( "The VUI settings are not used by the encoder but are merely suggestions to\n" );
372.     H2( "the playback equipment. See doc/vui.txt for details. Use at your own risk.\n" );
373.     H2( "\n" );
374.     H2( " --overscan <string> Specify crop overscan setting [\"%s\"]\n"
375.         "                                - undef, show, crop\n",
376.         strtable_lookup( x264_overscan_names, defaults->vui.i_overscan ) );
377.     H2( " --videoformat <string> Specify video format [\"%s\"]\n"
378.         "                                - component, pal, ntsc, secam, mac, undef\n",
379.         strtable_lookup( x264_vidformat_names, defaults->vui.i_vidformat ) );
380.     H2( " --range <string> Specify color range [\"%s\"]\n"
381.         "                                - %s\n", range_names[0], stringify_names( buf, range_names ) );
382.     H2( " --colorprim <string> Specify color primaries [\"%s\"]\n"
383.         "                                - undef, bt709, bt470m, bt470bg, smpte170m,\n"
384.         "                                smpte240m, film, bt2020\n",
385.         strtable_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. strtable_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. strtable_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-bufsize)\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 <intxint> 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( " --bluray-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", strtable_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%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。

```
[cpp]
1. //解析以字符串方式输入的参数
2. //即选项名称和选项值都是字符串
3. //实质就是通过strcmp()方法
4. int x264_param_parse( x264_param_t *p, const char *name, const char *value )
5. {
```

```

6.  char *name_buf = NULL;
7.  int b_error = 0;
8.  int name_was_bool;
9.  int value_was_null = !value;
10. int i;
11.
12. if( !name )
13.     return X264_PARAM_BAD_NAME;
14. if( !value )
15.     value = "true";
16.
17. if( value[0] == '=' )
18.     value++;
19.
20. if( strchr( name, '_' ) ) // s/_/-/g
21. {
22.     char *c;
23.     name_buf = strdup(name);
24.     while( ( c = strchr( name_buf, '_' ) ) )
25.         *c = '-';
26.     name = name_buf;
27. }
28.
29. if( (!strcmp( name, "no-", 3 ) && (i = 3)) ||
30.     (!strcmp( name, "no", 2 ) && (i = 2)) )
31. {
32.     name += i;
33.     value = atobool(value) ? "false" : "true";
34. }
35. name_was_bool = 0;
36.
37. #define OPT(STR) else if( !strcmp( name, STR ) )
38. #define OPT2(STR0, STR1) else if( !strcmp( name, STR0 ) || !strcmp( name, STR1 ) )
39. if(0);
40. //OPT()实际上就是strcmp()
41. OPT("asm")
42. {
43.     p->cpu = isdigit(value[0]) ? atoi(value) :
44.         !strcasecmp(value, "auto") || atobool(value) ? x264_cpu_detect() : 0;
45.     if( b_error )
46.     {
47.         char *buf = strdup(value);
48.         char *tok, UNUSED *saveptr=NULL, *init;
49.         b_error = 0;
50.         p->cpu = 0;
51.         for( init=buf; (tok=strtok_r(init, ",", &saveptr)); init=NULL )
52.         {
53.             for( i=0; x264_cpu_names[i].flags && strcmp(tok, x264_cpu_names[i].name); i++ );
54.             p->cpu |= x264_cpu_names[i].flags;
55.             if( !x264_cpu_names[i].flags )
56.                 b_error = 1;
57.         }
58.         free( buf );
59.         if( (p->cpu & X264_CPU_SSSE3) && !(p->cpu & X264_CPU_SSE2_IS_SLOW) )
60.             p->cpu |= X264_CPU_SSE2_IS_FAST;
61.     }
62. }
63. OPT("threads")
64. {
65.     if( !strcasecmp(value, "auto") )
66.         p->i_threads = X264_THREADS_AUTO;
67.     else
68.         p->i_threads = atoi(value);
69. }
70. OPT("lookahead-threads")
71. {
72.     if( !strcasecmp(value, "auto") )
73.         p->i_lookahead_threads = X264_THREADS_AUTO;
74.     else
75.         p->i_lookahead_threads = atoi(value);
76. }
77. OPT("sliced-threads")
78. p->b_sliced_threads = atobool(value);
79. OPT("sync-lookahead")
80. {
81.     if( !strcasecmp(value, "auto") )
82.         p->i_sync_lookahead = X264_SYNC_LOOKAHEAD_AUTO;
83.     else
84.         p->i_sync_lookahead = atoi(value);
85. }
86. OPT2("deterministic", "n-deterministic")
87. p->b_deterministic = atobool(value);
88. OPT("cpu-independent")
89. p->b_cpu_independent = atobool(value);
90. OPT2("level", "level-idc")
91. {
92.     if( !strcmp(value, "1b") )
93.         p->i_level_idc = 9;
94.     else if( atof(value) < 6 )
95.         p->i_level_idc = (int)(10*atof(value)+.5);
96.     else
97.         p->i_level_idc = atoi(value);

```

```

97.         p->i_level_luc = atoi(value);
98.     }
99.     OPT("bluray-compat")
100.     p->b_bluray_compat = atobool(value);
101.     OPT("avcintra-class")
102.     p->i_avcintra_class = atoi(value);
103.     OPT("sar")
104.     {
105.         b_error = ( 2 != sscanf( value, "%d:%d", &p->vui.i_sar_width, &p->vui.i_sar_height ) &&
106.                     2 != sscanf( value, "%d/%d", &p->vui.i_sar_width, &p->vui.i_sar_height ) );
107.     }
108.     OPT("overscan")
109.     b_error |= parse_enum( value, x264_overscan_names, &p->vui.i_overscan );
110.     OPT("videoformat")
111.     b_error |= parse_enum( value, x264_vidformat_names, &p->vui.i_vidformat );
112.     OPT("fullrange")
113.     b_error |= parse_enum( value, x264_fullrange_names, &p->vui.b_fullrange );
114.     OPT("colorprim")
115.     b_error |= parse_enum( value, x264_colorprim_names, &p->vui.i_colorprim );
116.     OPT("transfer")
117.     b_error |= parse_enum( value, x264_transfer_names, &p->vui.i_transfer );
118.     OPT("colormatrix")
119.     b_error |= parse_enum( value, x264_colmatrix_names, &p->vui.i_colmatrix );
120.     OPT("chromaloc")
121.     {
122.         p->vui.i_chroma_loc = atoi(value);
123.         b_error = ( p->vui.i_chroma_loc < 0 || p->vui.i_chroma_loc > 5 );
124.     }
125.     OPT("fps")
126.     {
127.         if( sscanf( value, "%u/%u", &p->i_fps_num, &p->i_fps_den ) == 2 )
128.             ;
129.         else
130.         {
131.             float fps = atof(value);
132.             if( fps > 0 && fps <= INT_MAX/1000 )
133.             {
134.                 p->i_fps_num = (int)(fps * 1000 + .5);
135.                 p->i_fps_den = 1000;
136.             }
137.             else
138.             {
139.                 p->i_fps_num = atoi(value);
140.                 p->i_fps_den = 1;
141.             }
142.         }
143.     }
144.     OPT2("ref", "frameref")
145.     p->i_frame_reference = atoi(value);
146.     OPT("dpb-size")
147.     p->i_dpb_size = atoi(value);
148.     OPT("keyint")
149.     {
150.         if( strstr( value, "infinite" ) )
151.             p->i_keyint_max = X264_KEYINT_MAX_INFINITE;
152.         else
153.             p->i_keyint_max = atoi(value);
154.     }
155.     OPT2("min-keyint", "keyint-min")
156.     {
157.         p->i_keyint_min = atoi(value);
158.         if( p->i_keyint_max < p->i_keyint_min )
159.             p->i_keyint_max = p->i_keyint_min;
160.     }
161.     OPT("scenecut")
162.     {
163.         p->i_scenecut_threshold = atobool(value);
164.         if( b_error || p->i_scenecut_threshold )
165.         {
166.             b_error = 0;
167.             p->i_scenecut_threshold = atoi(value);
168.         }
169.     }
170.     OPT("intra-refresh")
171.     p->b_intra_refresh = atobool(value);
172.     OPT("bframes")
173.     p->i_bframe = atoi(value);
174.     OPT("b-adapt")
175.     {
176.         p->i_bframe_adaptive = atobool(value);
177.         if( b_error )
178.         {
179.             b_error = 0;
180.             p->i_bframe_adaptive = atoi(value);
181.         }
182.     }
183.     OPT("b-bias")
184.     p->i_bframe_bias = atoi(value);
185.     OPT("b-pyramid")
186.     {
187.         b_error |= parse_enum( value, x264_b_pyramid_names, &p->i_bframe_pyramid );
188.         if( b_error )

```

```

189.     {
190.         b_error = 0;
191.         p->i_bframe_pyramid = atoi(value);
192.     }
193. }
194. OPT("open-gop")
195.     p->b_open_gop = atobool(value);
196. OPT("nfr")
197.     p->b_deblocking_filter = !atobool(value);
198. OPT2("filter", "deblock")
199.     {
200.         if( 2 == sscanf( value, "%d:%d", &p->i_deblocking_filter_alphac0, &p->i_deblocking_filter_beta ) ||
201.            2 == sscanf( value, "%d,%d", &p->i_deblocking_filter_alphac0, &p->i_deblocking_filter_beta ) )
202.         {
203.             p->b_deblocking_filter = 1;
204.         }
205.         else if( sscanf( value, "%d", &p->i_deblocking_filter_alphac0 ) )
206.         {
207.             p->b_deblocking_filter = 1;
208.             p->i_deblocking_filter_beta = p->i_deblocking_filter_alphac0;
209.         }
210.         else
211.             p->b_deblocking_filter = atobool(value);
212.     }
213. OPT("slice-max-size")
214.     p->i_slice_max_size = atoi(value);
215. OPT("slice-max-mbs")
216.     p->i_slice_max_mbs = atoi(value);
217. OPT("slice-min-mbs")
218.     p->i_slice_min_mbs = atoi(value);
219. OPT("slices")
220.     p->i_slice_count = atoi(value);
221. OPT("slices-max")
222.     p->i_slice_count_max = atoi(value);
223. OPT("cabac")
224.     p->b_cabac = atobool(value);
225. OPT("cabac-idc")
226.     p->i_cabac_init_idc = atoi(value);
227. OPT("interlaced")
228.     p->b_interlaced = atobool(value);
229. OPT("tff")
230.     p->b_interlaced = p->b_tff = atobool(value);
231. OPT("bff")
232.     {
233.         p->b_interlaced = atobool(value);
234.         p->b_tff = !p->b_interlaced;
235.     }
236. OPT("constrained-intra")
237.     p->b_constrained_intra = atobool(value);
238. OPT("cqm")
239.     {
240.         if( strstr( value, "flat" ) )
241.             p->i_cqm_preset = X264_CQM_FLAT;
242.         else if( strstr( value, "jvt" ) )
243.             p->i_cqm_preset = X264_CQM_JVT;
244.         else
245.             p->psz_cqm_file = strdup(value);
246.     }
247. OPT("cqmfile")
248.     p->psz_cqm_file = strdup(value);
249. OPT("cqm4")
250.     {
251.         p->i_cqm_preset = X264_CQM_CUSTOM;
252.         b_error |= parse_cqm( value, p->cqm_4iy, 16 );
253.         b_error |= parse_cqm( value, p->cqm_4py, 16 );
254.         b_error |= parse_cqm( value, p->cqm_4ic, 16 );
255.         b_error |= parse_cqm( value, p->cqm_4pc, 16 );
256.     }
257. OPT("cqm8")
258.     {
259.         p->i_cqm_preset = X264_CQM_CUSTOM;
260.         b_error |= parse_cqm( value, p->cqm_8iy, 64 );
261.         b_error |= parse_cqm( value, p->cqm_8py, 64 );
262.         b_error |= parse_cqm( value, p->cqm_8ic, 64 );
263.         b_error |= parse_cqm( value, p->cqm_8pc, 64 );
264.     }
265. OPT("cqm4i")
266.     {
267.         p->i_cqm_preset = X264_CQM_CUSTOM;
268.         b_error |= parse_cqm( value, p->cqm_4iy, 16 );
269.         b_error |= parse_cqm( value, p->cqm_4ic, 16 );
270.     }
271. OPT("cqm4p")
272.     {
273.         p->i_cqm_preset = X264_CQM_CUSTOM;
274.         b_error |= parse_cqm( value, p->cqm_4py, 16 );
275.         b_error |= parse_cqm( value, p->cqm_4pc, 16 );
276.     }
277. OPT("cqm4iy")
278.     {
279.         p->i_cqm_preset = X264_CQM_CUSTOM;

```

```

280.         b_error |= parse_cqm( value, p->cqm_4iy, 16 );
281.     }
282.     OPT("cqm4ic")
283.     {
284.         p->i_cqm_preset = X264_CQM_CUSTOM;
285.         b_error |= parse_cqm( value, p->cqm_4ic, 16 );
286.     }
287.     OPT("cqm4py")
288.     {
289.         p->i_cqm_preset = X264_CQM_CUSTOM;
290.         b_error |= parse_cqm( value, p->cqm_4py, 16 );
291.     }
292.     OPT("cqm4pc")
293.     {
294.         p->i_cqm_preset = X264_CQM_CUSTOM;
295.         b_error |= parse_cqm( value, p->cqm_4pc, 16 );
296.     }
297.     OPT("cqm8i")
298.     {
299.         p->i_cqm_preset = X264_CQM_CUSTOM;
300.         b_error |= parse_cqm( value, p->cqm_8iy, 64 );
301.         b_error |= parse_cqm( value, p->cqm_8ic, 64 );
302.     }
303.     OPT("cqm8p")
304.     {
305.         p->i_cqm_preset = X264_CQM_CUSTOM;
306.         b_error |= parse_cqm( value, p->cqm_8py, 64 );
307.         b_error |= parse_cqm( value, p->cqm_8pc, 64 );
308.     }
309.     OPT("log")
310.         p->i_log_level = atoi(value);
311.     OPT("dump-yuv")
312.         p->psz_dump_yuv = strdup(value);
313.     OPT2("analyse", "partitions")
314.     {
315.         p->analyse.inter = 0;
316.         if( strstr( value, "none" ) ) p->analyse.inter = 0;
317.         if( strstr( value, "all" ) ) p->analyse.inter = -0;
318.
319.         if( strstr( value, "i4x4" ) ) p->analyse.inter |= X264_ANALYSE_I4x4;
320.         if( strstr( value, "i8x8" ) ) p->analyse.inter |= X264_ANALYSE_I8x8;
321.         if( strstr( value, "p8x8" ) ) p->analyse.inter |= X264_ANALYSE_PSUB16x16;
322.         if( strstr( value, "p4x4" ) ) p->analyse.inter |= X264_ANALYSE_PSUB8x8;
323.         if( strstr( value, "b8x8" ) ) p->analyse.inter |= X264_ANALYSE_BSUB16x16;
324.     }
325.     OPT("8x8dct")
326.         p->analyse.b_transform_8x8 = atobool(value);
327.     OPT2("weightb", "weight-b")
328.         p->analyse.b_weighted_bipred = atobool(value);
329.     OPT("weightp")
330.         p->analyse.i_weighted_pred = atoi(value);
331.     OPT2("direct", "direct-pred")
332.         b_error |= parse_enum( value, x264_direct_pred_names, &p->analyse.i_direct_mv_pred );
333.     OPT("chroma-qp-offset")
334.         p->analyse.i_chroma_qp_offset = atoi(value);
335.     OPT("me")
336.         b_error |= parse_enum( value, x264_motion_est_names, &p->analyse.i_me_method );
337.     OPT2("merange", "me-range")
338.         p->analyse.i_me_range = atoi(value);
339.     OPT2("mvrange", "mv-range")
340.         p->analyse.i_mv_range = atoi(value);
341.     OPT2("mvrange-thread", "mv-range-thread")
342.         p->analyse.i_mv_range_thread = atoi(value);
343.     OPT2("subme", "subq")
344.         p->analyse.i_subpel_refine = atoi(value);
345.     OPT("psy-rd")
346.     {
347.         if( 2 == sscanf( value, "%f:%f", &p->analyse.f_psy_rd, &p->analyse.f_psy_trellis ) ||
348.            2 == sscanf( value, "%f,%f", &p->analyse.f_psy_rd, &p->analyse.f_psy_trellis ) ||
349.            2 == sscanf( value, "%f|%f", &p->analyse.f_psy_rd, &p->analyse.f_psy_trellis ) )
350.         {
351.             else if( sscanf( value, "%f", &p->analyse.f_psy_rd ) )
352.             {
353.                 p->analyse.f_psy_trellis = 0;
354.             }
355.             else
356.             {
357.                 p->analyse.f_psy_rd = 0;
358.                 p->analyse.f_psy_trellis = 0;
359.             }
360.         }
361.     OPT("psy")
362.         p->analyse.b_psy = atobool(value);
363.     OPT("chroma-me")
364.         p->analyse.b_chroma_me = atobool(value);
365.     OPT("mixed-refs")
366.         p->analyse.b_mixed_references = atobool(value);
367.     OPT("trellis")
368.         p->analyse.i_trellis = atoi(value);
369.     OPT("fast-pskip")
370.         p->analyse.b_fast_pskip = atobool(value);

```

```

371.     OPT("dct-decimate")
372.         p->analyse.b_dct_decimate = atobool(value);
373.     OPT("deadzone-inter")
374.         p->analyse.i_luma_deadzone[0] = atoi(value);
375.     OPT("deadzone-intra")
376.         p->analyse.i_luma_deadzone[1] = atoi(value);
377.     OPT("nr")
378.         p->analyse.i_noise_reduction = atoi(value);
379.     OPT("bitrate")
380.     {
381.         p->rc.i_bitrate = atoi(value);
382.         p->rc.i_rc_method = X264_RC_ABR;
383.     }
384.     OPT2("qp", "qp_constant")
385.     {
386.         p->rc.i_qp_constant = atoi(value);
387.         p->rc.i_rc_method = X264_RC_CQP;
388.     }
389.     OPT("crf")
390.     {
391.         p->rc.f_rf_constant = atof(value);
392.         p->rc.i_rc_method = X264_RC_CRF;
393.     }
394.     OPT("crf-max")
395.         p->rc.f_rf_constant_max = atof(value);
396.     OPT("rc-lookahead")
397.         p->rc.i_lookahead = atoi(value);
398.     OPT2("qpmin", "qp-min")
399.         p->rc.i_qp_min = atoi(value);
400.     OPT2("qpmax", "qp-max")
401.         p->rc.i_qp_max = atoi(value);
402.     OPT2("qpstep", "qp-step")
403.         p->rc.i_qp_step = atoi(value);
404.     OPT("ratetol")
405.         p->rc.f_rate_tolerance = !strncmp("inf", value, 3) ? 1e9 : atof(value);
406.     OPT("vbv-maxrate")
407.         p->rc.i_vbv_max_bitrate = atoi(value);
408.     OPT("vbv-bufsize")
409.         p->rc.i_vbv_buffer_size = atoi(value);
410.     OPT("vbv-init")
411.         p->rc.f_vbv_buffer_init = atof(value);
412.     OPT2("ipratio", "ip-factor")
413.         p->rc.f_ip_factor = atof(value);
414.     OPT2("pbratio", "pb-factor")
415.         p->rc.f_pb_factor = atof(value);
416.     OPT("aq-mode")
417.         p->rc.i_aq_mode = atoi(value);
418.     OPT("aq-strength")
419.         p->rc.f_aq_strength = atof(value);
420.     OPT("pass")
421.     {
422.         int pass = x264_clip3( atoi(value), 0, 3 );
423.         p->rc.b_stat_write = pass & 1;
424.         p->rc.b_stat_read = pass & 2;
425.     }
426.     OPT("stats")
427.     {
428.         p->rc.psz_stat_in = strdup(value);
429.         p->rc.psz_stat_out = strdup(value);
430.     }
431.     OPT("qcomp")
432.         p->rc.f_qcompress = atof(value);
433.     OPT("mbtree")
434.         p->rc.b_mb_tree = atobool(value);
435.     OPT("qblur")
436.         p->rc.f_qblur = atof(value);
437.     OPT2("cplxblur", "cplx-blur")
438.         p->rc.f_complexity_blur = atof(value);
439.     OPT("zones")
440.         p->rc.psz_zones = strdup(value);
441.     OPT("crop-rect")
442.         b_error |= sscanf( value, "%u,%u,%u,%u", &p->crop_rect.i_left, &p->crop_rect.i_top,
443.                             &p->crop_rect.i_right, &p->crop_rect.i_bottom ) != 4;
444.     OPT("psnr")
445.         p->analyse.b_psnr = atobool(value);
446.     OPT("ssim")
447.         p->analyse.b_ssim = atobool(value);
448.     OPT("aud")
449.         p->b_aud = atobool(value);
450.     OPT("sps-id")
451.         p->i_sps_id = atoi(value);
452.     OPT("global-header")
453.         p->b_repeat_headers = !atobool(value);
454.     OPT("repeat-headers")
455.         p->b_repeat_headers = atobool(value);
456.     OPT("annexb")
457.         p->b_annexb = atobool(value);
458.     OPT("force-cfr")
459.         p->b_vfr_input = !atobool(value);
460.     OPT("nal-hrd")
461.         b_error |= parse_enum( value, x264_nal_hrd_names, &p->i_nal_hrd );

```



```

462.     OPT("filler")
463.         p->rc.b_filler = atobool(value);
464.     OPT("pic-struct")
465.         p->b_pic_struct = atobool(value);
466.     OPT("fake-interlaced")
467.         p->b_fake_interlaced = atobool(value);
468.     OPT("frame-packing")
469.         p->i_frame_packing = atoi(value);
470.     OPT("stitchable")
471.         p->b_stitchable = atobool(value);
472.     OPT("opencl")
473.         p->b_opencl = atobool( value );
474.     OPT("opencl-clbin")
475.         p->psz_clbin_file = strdup( value );
476.     OPT("opencl-device")
477.         p->i_opencl_device = atoi( value );
478.     else
479.         return X264_PARAM_BAD_NAME;
480. #undef OPT
481. #undef OPT2
482. #undef atobool
483. #undef atoi
484. #undef atof
485.
486.     if( name_buf )
487.         free( name_buf );
488.
489.     b_error |= value_was_null && !name_was_bool;
490.     return b_error ? X264_PARAM_BAD_VALUE : 0;
491. }

```

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。

```

1.  /*      (can be NULL, in which case the function will do nothing)
2.  *
3.  *      Does NOT guarantee that the given profile will be used: if the restrictions
4.  *      of "High" are applied to settings that are already Baseline-compatible, the
5.  *      stream will remain baseline. In short, it does not increase settings, only
6.  *      decrease them.
7.  *
8.  *      returns 0 on success, negative on failure (e.g. invalid profile name). */
9.  int x264_param_apply_profile( x264_param_t *, const char *profile );

```

x264\_param\_apply\_profile()的定义如下所示。

```

1. //设置profile
2. int x264_param_apply_profile( x264_param_t *param, const char *profile )
3. {
4.     if( !profile )
5.         return 0;
6.     //字符串到整型
7.     int p = profile_string_to_int( profile );
8.     //检查profile设置是否正确
9.     if( p < 0 )
10.    {
11.        x264_log( NULL, X264_LOG_ERROR, "invalid profile: %s\n", profile );
12.        return -1;
13.    }
14.    if( p < PROFILE_HIGH444_PREDICTIVE && ((param->rc.i_rc_method == X264_RC_CQP && param->rc.i_qp_constant <= 0) ||
15.        (param->rc.i_rc_method == X264_RC_CRF && (int)(param->rc.f_rf_constant + QP_BD_OFFSET) <= 0)) )
16.    {
17.        x264_log( NULL, X264_LOG_ERROR, "%s profile doesn't support lossless\n", profile );
18.        return -1;
19.    }
20.    if( p < PROFILE_HIGH444_PREDICTIVE && (param->i_csp & X264_CSP_MASK) >= X264_CSP_I444 )
21.    {
22.        x264_log( NULL, X264_LOG_ERROR, "%s profile doesn't support 4:4:4\n", profile );
23.        return -1;
24.    }
25.    if( p < PROFILE_HIGH422 && (param->i_csp & X264_CSP_MASK) >= X264_CSP_I422 )
26.    {
27.        x264_log( NULL, X264_LOG_ERROR, "%s profile doesn't support 4:2:2\n", profile );
28.        return -1;
29.    }
30.    if( p < PROFILE_HIGH10 && BIT_DEPTH > 8 )
31.    {
32.        x264_log( NULL, X264_LOG_ERROR, "%s profile doesn't support a bit depth of %d\n", profile, BIT_DEPTH );
33.        return -1;
34.    }
35.    //根据不同的Profile做设置
36.    //Baseline基本型
37.    if( p == PROFILE_BASELINE )
38.    {
39.        //不支持DCT8x8
40.        param->analyse.b_transform_8x8 = 0;
41.        //不使用CABAC
42.        param->b_cabac = 0;
43.        param->i_cqm_preset = X264_CQM_FLAT;
44.        param->psz_cqm_file = NULL;
45.        //没有B帧
46.        param->i_bframe = 0;
47.        //没有加权
48.        param->analyse.i_weighted_pred = X264_WEIGHTP_NONE;
49.        //不支持隔行扫描
50.        if( param->b_interlaced )
51.        {
52.            x264_log( NULL, X264_LOG_ERROR, "baseline profile doesn't support interlacing\n" );
53.            return -1;
54.        }
55.        if( param->b_fake_interlaced )
56.        {
57.            x264_log( NULL, X264_LOG_ERROR, "baseline profile doesn't support fake interlacing\n" );
58.            return -1;
59.        }
60.    }
61.    //Main主型
62.    else if( p == PROFILE_MAIN )
63.    {
64.        //不支持DCT8x8
65.        param->analyse.b_transform_8x8 = 0;
66.        param->i_cqm_preset = X264_CQM_FLAT;
67.        param->psz_cqm_file = NULL;
68.    }
69.    return 0;
70. }

```

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。

```
[cpp]
1. static int profile_string_to_int( const char *str )
2. {
3.     if( !strcasecmp( str, "baseline" ) )
4.         return PROFILE_BASELINE;
5.     if( !strcasecmp( str, "main" ) )
6.         return PROFILE_MAIN;
7.     if( !strcasecmp( str, "high" ) )
8.         return PROFILE_HIGH;
9.     if( !strcasecmp( str, "high10" ) )
10.        return PROFILE_HIGH10;
11.    if( !strcasecmp( str, "high422" ) )
12.        return PROFILE_HIGH422;
13.    if( !strcasecmp( str, "high444" ) )
14.        return PROFILE_HIGH444_PREDICTIVE;
15.    return -1;
16. }
```



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。

```
[cpp]
1. //根据文件名的后缀确定输出的文件格式 (raw H264, flv, mp4...)
2. static int select_output( const char *muxer, char *filename, x264_param_t *param )
3. {
4.     //从文件路径字符串中解析出扩展名, 存入ext
5.     //解析的方式就是反向搜索字符"."
6.     const char *ext = get_filename_extension( filename );
7.
8.     //strcasecmp(char *s1, char *s2)用于忽略大小写比较字符串。
9.     //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。
10.
11.    if( !strcmp( filename, "-" ) || strcmp( muxer, "auto" ) )
12.        ext = muxer;
13.    //后缀为"mp4"
14.    if( !strcasecmp( ext, "mp4" ) )
15.    {
16.        #if HAVE_GOPAC || HAVE_LSMASH
17.            cli_output = mp4_output;
18.            param->b_annexb = 0;
19.            param->b_repeat_headers = 0;
20.            if( param->i_nal_hrd == X264_NAL_HRD_CBR )
21.            {
22.                x264_cli_log( "x264", X264_LOG_WARNING, "cbr nal-hrd is not compatible with mp4\n" );
23.                param->i_nal_hrd = X264_NAL_HRD_VBR;
24.            }
25.        #else
26.            x264_cli_log( "x264", X264_LOG_ERROR, "not compiled with MP4 output support\n" );
27.            return -1;
28.        #endif
29.    }
30.    else if( !strcasecmp( ext, "mkv" ) )
31.    {
32.        //设定cli_output_t
33.        cli_output = mkv_output;
34.        //不加起始码0x00000001
35.        param->b_annexb = 0;
36.        //不再每个Keyframe前面加SPS和PPS
37.        param->b_repeat_headers = 0;
38.    }
39.    else if( !strcasecmp( ext, "flv" ) )
40.    {
41.        cli_output = flv_output;
42.        param->b_annexb = 0;
43.        param->b_repeat_headers = 0;
44.    }
45.    else
46.        cli_output = raw_output; //不符合上述后缀, 则输出裸流
47.    return 0;
48. }
```

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。

```
[cpp]    
1. //根据“.”确定文件后缀  
2. static inline char *get_filename_extension( char *filename )  
3. {  
4.     char *ext = filename + strlen( filename );  
5.     while( *ext != '.' && ext > filename )  
6.         ext--;  
7.     ext += *ext == '.';  
8.     return ext;  
9. }
```

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。

```

1. //设置输入文件的格式 (yuv, y4m...)
2. static int select_input( const char *demuxer, char *used_demuxer, char *filename,
3.                         hnd_t *p_handle, video_info_t *info, cli_input_opt_t *opt )
4. {
5.     int b_auto = !strcasecmp( demuxer, "auto" );
6.     //从文件路径字符串中解析出扩展名, 存入ext
7.     //解析的方式就是反向搜索字符"."
8.     const char *ext = b_auto ? get_filename_extension( filename ) : "";
9.     int b_regular = strcmp( filename, "-" );
10.    if( !b_regular && b_auto )
11.        ext = "raw";
12.    b_regular = b_regular && x264_is_regular_file_path( filename );
13.    if( b_regular )
14.    {
15.        FILE *f = x264_fopen( filename, "r" );
16.        if( f )
17.        {
18.            b_regular = x264_is_regular_file( f );
19.            fclose( f );
20.        }
21.    }
22.    const char *module = b_auto ? ext : demuxer;
23.
24.    //strcasecmp(char *s1, char *s2)用于忽略大小写比较字符串.
25.    //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.
26.
27.    if( !strcasecmp( module, "avs" ) || !strcasecmp( ext, "d2v" ) || !strcasecmp( ext, "dga" ) )
28.    {
29.        #if HAVE_AVS
30.            cli_input = avs_input;
31.            module = "avs";
32.        #else
33.            x264_cli_log( "x264", X264_LOG_ERROR, "not compiled with AVS input support\n" );
34.            return -1;
35.        #endif
36.    }
37.    else if( !strcasecmp( module, "y4m" ) )
38.        cli_input = y4m_input;
39.    else if( !strcasecmp( module, "raw" ) || !strcasecmp( ext, "yuv" ) )
40.        cli_input = raw_input;
41.    else
42.    {
43.        #if HAVE_FFMS
44.            if( b_regular && ( b_auto || !strcasecmp( demuxer, "ffms" ) ) &&
45.                !ffms_input.open_file( filename, p_handle, info, opt ) )
46.            {
47.                module = "ffms";
48.                b_auto = 0;
49.                cli_input = ffms_input;
50.            }
51.        #endif
52.        #if HAVE_LAVF
53.            if( ( b_auto || !strcasecmp( demuxer, "lavf" ) ) &&
54.                !lavf_input.open_file( filename, p_handle, info, opt ) )
55.            {
56.                module = "lavf";
57.                b_auto = 0;
58.                cli_input = lavf_input;
59.            }
60.        #endif
61.        #if HAVE_AVS
62.            if( b_regular && ( b_auto || !strcasecmp( demuxer, "avs" ) ) &&
63.                !avs_input.open_file( filename, p_handle, info, opt ) )
64.            {
65.                module = "avs";
66.                b_auto = 0;
67.                cli_input = avs_input;
68.            }
69.        #endif
70.        if( b_auto && !raw_input.open_file( filename, p_handle, info, opt ) )
71.        {
72.            module = "raw";
73.            b_auto = 0;
74.            cli_input = raw_input;
75.        }
76.
77.        FAIL_IF_ERROR( !( *p_handle ), "could not open input file '%s' via any method!\n", filename )
78.    }
79.    strcpy( used_demuxer, module );
80.
81.    return 0;
82. }

```

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。

```
[cpp]  
1. //编码（在内部有一个循环用于一帧一帧编码）
2. static int encode( x264_param_t *param, cli_opt_t *opt )
3. {
4.     x264_t *h = NULL;
5.     x264_picture_t pic;
6.     cli_pic_t cli_pic;
7.     const cli_pulldown_t *pulldown = NULL; // shut up gcc
8.
9.     int i_frame = 0;
10.    int i_frame_output = 0;
11.    int64_t i_end, i_previous = 0, i_start = 0;
12.    int64_t i_file = 0;
13.    int i_frame_size;
14.    int64_t last_dts = 0;
15.    int64_t prev_dts = 0;
16.    int64_t first_dts = 0;
17.    # define MAX_PTS_WARNING 3 /* arbitrary */
18.    int pts_warning_cnt = 0;
19.    int64_t largest_pts = -1;
20.    int64_t second_largest_pts = -1;
21.    int64_t ticks_per_frame;
22.    double duration;
23.    double pulldown_pts = 0;
24.    int retval = 0;
25.
26.    opt->b_progress &= param->i_log_level < X264_LOG_DEBUG;
27.
28.    /* set up pulldown */
29.    if( opt->i_pulldown && !param->b_vfr_input )
30.    {
31.        param->b_pulldown = 1;
32.        param->b_pic_struct = 1;
33.        pulldown = &pulldown_values[opt->i_pulldown];
34.        param->i_timebase_num = param->i_fps_den;
35.        FAIL_IF_ERROR2( fmod( param->i_fps_num * pulldown->fps_factor, 1 ),
36.            "unsupported framerate for chosen pulldown\n" );
37.        param->i_timebase_den = param->i_fps_num * pulldown->fps_factor;
38.    }
39.    //打开编码器
40.    h = x264_encoder_open( param );
41.    FAIL_IF_ERROR2( !h, "x264_encoder_open failed\n" );
42.    //获得参数
43.    x264_encoder_parameters( h, param );
44.    //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
45.    //cli_output_t是代表输出媒体文件的结构体
46.    FAIL_IF_ERROR2( cli_output.set_param( opt->hout, param ), "can't set outfile param\n" );
47.    //计时
48.    i_start = x264_mdate();
49.
50.    /* ticks/frame = ticks/second / frames/second */
51.    ticks_per_frame = (int64_t)param->i_timebase_den * param->i_fps_den / param->i_timebase_num / param->i_fps_num;
52.    FAIL_IF_ERROR2( ticks_per_frame < 1 && !param->b_vfr_input, "ticks_per_frame invalid: %\"PRIu64\"\\n\", ticks_per_frame )
53.    ticks_per_frame = X264_MAX( ticks_per_frame, 1 );
54.
55.    //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
56.    //Header指的就是SPS/PPS/SEI
57.    if( !param->b_repeat_headers )
58.    {
59.        // Write SPS/PPS/SEI
60.        x264_nal_t *headers;
61.        int i_nal;
62.        //获得文件头（SPS、PPS、SEI）
63.        FAIL_IF_ERROR2( x264_encoder_headers( h, &headers, &i_nal ) < 0, "x264_encoder_headers failed\n" );
64.        //把文件头写入输出文件
65.        FAIL_IF_ERROR2( (i_file = cli_output.write_headers( opt->hout, headers )) < 0, "error writing headers to output file\n" );
66.    }
67.
68.    if( opt->tcfile_out )
69.        fprintf( opt->tcfile_out, "# timecode format v2\\n" );
70.
71.    /* Encode frames */
72.    //循环进行编码
73.    for( ; !b_ctrl_c && (i_frame < param->i_frame_total || !param->i_frame_total); i_frame++ )
74.    {
75.        //从输入源中获取1帧YUV数据，存于cli_pic
76.        //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。
77.        //原本代表输入源的结构体是cli_input_t
78.        if( filter.get_frame( opt->hin, &cli_pic, i_frame + opt->i_seek ) )
79.            break;
80.        //初始化x264_picture_t结构体pic
81.        x264_picture_init( &pic );
82.        //cli_pic到pic
83.        convert_cli_to_lib_pic( &pic, &cli_pic );
```

```

84.
85.     if( !param->b_vfr_input )
86.         pic.i_pts = i_frame;
87.
88.     if( opt->i_pulldown && !param->b_vfr_input )
89.     {
90.         pic.i_pic_struct = pulldown->pattern[ i_frame % pulldown->mod ];
91.         pic.i_pts = (int64_t)( pulldown_pts + 0.5 );
92.         pulldown_pts += pulldown_frame_duration[pic.i_pic_struct];
93.     }
94.     else if( opt->timebase_convert_multiplier )
95.         pic.i_pts = (int64_t)( pic.i_pts * opt->timebase_convert_multiplier + 0.5 );
96.
97.     if( pic.i_pts <= largest_pts )
98.     {
99.         if( cli_log_level >= X264_LOG_DEBUG || pts_warning_cnt < MAX_PTS_WARNING )
100.             x264_cli_log( "x264", X264_LOG_WARNING, "non-strictly-monotonic pts at frame %d (%\"PRId64\" <= %\"PRId64\")\n",
101.                 i_frame, pic.i_pts, largest_pts );
102.         else if( pts_warning_cnt == MAX_PTS_WARNING )
103.             x264_cli_log( "x264", X264_LOG_WARNING, "too many nonmonotonic pts warnings, suppressing further ones\n" );
104.         pts_warning_cnt++;
105.         pic.i_pts = largest_pts + ticks_per_frame;
106.     }
107.
108.     second_largest_pts = largest_pts;
109.     largest_pts = pic.i_pts;
110.     if( opt->tcfile_out )
111.         fprintf( opt->tcfile_out, "%.6f\n", pic.i_pts * ((double)param->i_timebase_num / param->i_timebase_den) * 1e3 );
112.
113.     if( opt->qpfile )
114.         parse_qpfile( opt, &pic, i_frame + opt->i_seek );
115.
116.     prev_dts = last_dts;
117.     //编码pic中存储的1帧YUV数据
118.     i_frame_size = encode_frame( h, opt->hout, &pic, &last_dts );
119.     if( i_frame_size < 0 )
120.     {
121.         b_ctrl_c = 1; /* lie to exit the loop */
122.         retval = -1;
123.     }
124.     else if( i_frame_size )
125.     {
126.         i_file += i_frame_size;
127.         i_frame_output++;
128.         if( i_frame_output == 1 )
129.             first_dts = prev_dts = last_dts;
130.     }
131.     //释放处理完的YUV数据
132.     if( filter.release_frame( opt->hin, &cli_pic, i_frame + opt->i_seek ) )
133.         break;
134.
135.     /* update status line (up to 1000 times per input file) */
136.     if( opt->b_progress && i_frame_output )
137.         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 );
138.     }
139.     /* Flush delayed frames */
140.     //输出编码器中剩余的帧
141.     //x264_encoder_delayed_frames()返回剩余的帧的个数
142.     while( !b_ctrl_c && x264_encoder_delayed_frames( h ) )
143.     {
144.         prev_dts = last_dts;
145.         //编码
146.         //注意第3个参数为NULL
147.         i_frame_size = encode_frame( h, opt->hout, NULL, &last_dts );
148.         if( i_frame_size < 0 )
149.         {
150.             b_ctrl_c = 1; /* lie to exit the loop */
151.             retval = -1;
152.         }
153.         else if( i_frame_size )
154.         {
155.             i_file += i_frame_size;
156.             i_frame_output++;
157.             if( i_frame_output == 1 )
158.                 first_dts = prev_dts = last_dts;
159.         }
160.         //输出一些统计信息
161.         if( opt->b_progress && i_frame_output )
162.             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 );
163.     }
164. fail:
165.     if( pts_warning_cnt >= MAX_PTS_WARNING && cli_log_level < X264_LOG_DEBUG )
166.         x264_cli_log( "x264", X264_LOG_WARNING, "%d suppressed nonmonotonic pts warnings\n", pts_warning_cnt-MAX_PTS_WARNING );
167.
168.     /* duration algorithm fails when only 1 frame is output */
169.     if( i_frame_output == 1 )
170.         duration = (double)param->i_fps_den / param->i_fps_num;
171.     else if( b_ctrl_c )
172.         duration = (double)(2 * last_dts - prev_dts - first_dts) * param->i_timebase_num / param->i_timebase_den;

```

```

173.         else
174.             duration = (double)(2 * largest_pts - second_largest_pts) * param->i_timebase_num / param->i_timebase_den;
175.             //计时
176.             i_end = x264_mdate();
177.             /* Erase progress indicator before printing encoding stats. */
178.             if( opt->b_progress )
179.                 fprintf( stderr, "                                \r" );
180.             //关闭编码器
181.             if( h )
182.                 x264_encoder_close( h );
183.             fprintf( stderr, "\n" );
184.
185.             if( b_ctrl_c )
186.                 fprintf( stderr, "aborted at input frame %d, output frame %d\n", opt->i_seek + i_frame, i_frame_output );
187.             //关闭输出文件
188.             cli_output.close_file( opt->hout, largest_pts, second_largest_pts );
189.             opt->hout = NULL;
190.
191.             if( i_frame_output > 0 )
192.             {
193.                 double fps = (double)i_frame_output * (double)1000000 /
194.                     (double)( i_end - i_start );
195.
196.                 fprintf( stderr, "encoded %d frames, %.2f fps, %.2f kb/s\n", i_frame_output, fps,
197.                     (double) i_file * 8 / ( 1000 * duration ) );
198.             }
199.
200.             return retval;
201.     }

```

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。



```

1. //编码1帧
2. static int encode_frame( x264_t *h, hnd_t hout, x264_picture_t *pic, int64_t *last_dts )
3. {
4.     x264_picture_t pic_out;
5.     x264_nal_t *nal;
6.     int i_nal;
7.     int i_frame_size = 0;
8.     //编码API
9.     //编码x264_picture_t为x264_nal_t
10.    i_frame_size = x264_encoder_encode( h, &nal, &i_nal, pic, &pic_out );
11.
12.    FAIL_IF_ERROR( i_frame_size < 0, "x264_encoder_encode failed\n" );
13.
14.    if( i_frame_size )
15.    {
16.        //通过cli_output_t中的方法输出
17.        //输出raw H.264流的话, 等同于直接fwrite()
18.        //其他封装格式, 则还需进行一定的封装
19.        i_frame_size = cli_output.write_frame( hout, nal[0].p_payload, i_frame_size, &pic_out );
20.        *last_dts = pic_out.i_dts;
21.    }
22.
23.    return i_frame_size;
24. }

```

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cli\_output\_t结构体的知识将在后文中记录。

## print\_status()

print\_status()的定义如下。

```

1. //打印一些和时间有关的统计信息
2. 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 )
3. {
4.     char buf[200];
5.     int64_t i_time = x264_mdate();
6.     if( i_previous && i_time - i_previous < UPDATE_INTERVAL )
7.         return i_previous;
8.     int64_t i_elapsed = i_time - i_start;
9.     double fps = i_elapsed > 0 ? i_frame * 1000000. / i_elapsed : 0;
10.    double bitrate;
11.    if( last_ts )
12.        bitrate = (double) i_file * 8 / ( (double) last_ts * 1000 * param->i_timebase_num / param->i_timebase_den );
13.    else
14.        bitrate = (double) i_file * 8 / ( (double) 1000 * param->i_fps_den / param->i_fps_num );
15.    if( i_frame_total )
16.    {
17.        //形成输出的字符串
18.        int eta = i_elapsed * ( i_frame_total - i_frame ) / ((int64_t)i_frame * 1000000);
19.        sprintf( buf, "x264 [%1f%%] %d/%d frames, %.2f fps, %.2f kb/s, eta %d:%02d:%02d",
20.                100. * i_frame / i_frame_total, i_frame, i_frame_total, fps, bitrate,
21.                eta/3600, (eta/60)%60, eta%60 );
22.    }
23.    else
24.        sprintf( buf, "x264 %d frames: %.2f fps, %.2f kb/s", i_frame, fps, bitrate );
25.    //输出到stderr
26.    fprintf( stderr, "%s \r", buf+5 );
27.    //设置到标题栏?
28.    x264_cli_set_console_title( buf );
29.    fflush( stderr ); // needed in windows
30.    return i_time;
31. }

```

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。



<http://blog.csdn.net/leixiaohua1020>

## 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。

```
[cpp]    
1.  typedef struct  
2.  {  
3.      int (*open_file)( char *psz_filename, hnd_t *p_handle, cli_output_opt_t *opt );  
4.      int (*set_param)( hnd_t handle, x264_param_t *p_param );  
5.      int (*write_headers)( hnd_t handle, x264_nal_t *p_nal );  
6.      int (*write_frame)( hnd_t handle, uint8_t *p_nal, int i_size, x264_picture_t *p_picture );  
7.      int (*close_file)( hnd_t handle, int64_t largest_pts, int64_t second_largest_pts );  
8.  } cli_output_t;  
9.  
10. extern const cli_output_t raw_output;  
11. extern const cli_output_t mkv_output;  
12. extern const cli_output_t mp4_output;  
13. 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\_output, 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。

```
[cpp]    
1.  #include "output.h"  
2.  
3.  static int open_file( char *psz_filename, hnd_t *p_handle, cli_output_opt_t *opt )  
4.  {  
5.      if( !strcmp( psz_filename, "-" ) )  
6.          *p_handle = stdout;  
7.      else if( !(*p_handle = x264_fopen( psz_filename, "w+b" )) )  
8.          return -1;  
9.  
10.     return 0;  
11. }  
12.  
13. static int set_param( hnd_t handle, x264_param_t *p_param )  
14. {  
15.     return 0;  
16. }  
17.  
18. static int write_headers( hnd_t handle, x264_nal_t *p_nal )  
19. {  
20.     int size = p_nal[0].i_payload + p_nal[1].i_payload + p_nal[2].i_payload;  
21.  
22.     if( fwrite( p_nal[0].p_payload, size, 1, (FILE*)handle ) )  
23.         return size;  
24.     return -1;  
25. }  
26.  
27. static int write_frame( hnd_t handle, uint8_t *p_nalu, int i_size, x264_picture_t *p_picture )  
28. {  
29.     if( fwrite( p_nalu, i_size, 1, (FILE*)handle ) )  
30.         return i_size;  
31.     return -1;  
32. }  
33.  
34. static int close_file( hnd_t handle, int64_t largest_pts, int64_t second_largest_pts )  
35. {  
36.     if( !handle || handle == stdout )  
37.         return 0;  
38.  
39.     return fclose( (FILE*)handle );  
40. }  
41.  
42. 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。

```
[cpp]    
1. 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。

```
[cpp]    
1. static int write_header( flv_buffer *c )  
2. {  
3.     flv_put_tag( c, "FLV" ); // Signature  
4.     flv_put_byte( c, 1 );    // Version  
5.     flv_put_byte( c, 1 );    // Video Only  
6.     flv_put_be32( c, 9 );    // DataOffset  
7.     flv_put_be32( c, 0 );    // PreviousTagSize0  
8.  
9.     return flv_flush_data( c );  
10. }  
11.  
12. static int open_file( char *psz_filename, hnd_t *p_handle, cli_output_opt_t *opt )  
13. {  
14.     *p_handle = NULL;  
15.     flv_hnd_t *p_flv = calloc( 1, sizeof(flv_hnd_t) );  
16.     if( !p_flv )  
17.         return -1;  
18.  
19.     p_flv->b_dts_compress = opt->use_dts_compress;  
20.  
21.     p_flv->c = flv_create_writer( psz_filename );  
22.     if( !p_flv->c )  
23.         return -1;  
24.  
25.     CHECK( write_header( p_flv->c ) );  
26.     *p_handle = p_flv;  
27.  
28.     return 0;  
29. }
```

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。

```

1. static int write_frame( hnd_t handle, uint8_t *p_nalu, int i_size, x264_picture_t *p_picture )
2. {
3.     flv_hnd_t *p_flv = handle;
4.     flv_buffer *c = p_flv->c;
5.
6.     #define convert_timebase_ms( timestamp, timebase ) (int64_t)((timestamp) * (timebase) * 1000 + 0.5)
7.
8.     if( !p_flv->i_framenum )
9.     {
10.         p_flv->i_delay_time = p_picture->i_dts * -1;
11.         if( !p_flv->b_dts_compress && p_flv->i_delay_time )
12.             x264_cli_log( "flv", X264_LOG_INFO, "initial delay %"PRIu64" ms\n",
13.                 convert_timebase_ms( p_picture->i_pts + p_flv->i_delay_time, p_flv->d_timebase ) );
14.     }
15.
16.     int64_t dts;
17.     int64_t cts;
18.     int64_t offset;
19.
20.     if( p_flv->b_dts_compress )
21.     {
22.         if( p_flv->i_framenum == 1 )
23.             p_flv->i_init_delta = convert_timebase_ms( p_picture->i_dts + p_flv->i_delay_time, p_flv->d_timebase );
24.         dts = p_flv->i_framenum > p_flv->i_delay_frames
25.             ? convert_timebase_ms( p_picture->i_dts, p_flv->d_timebase )
26.             : p_flv->i_framenum * p_flv->i_init_delta / (p_flv->i_delay_frames + 1);
27.         cts = convert_timebase_ms( p_picture->i_pts, p_flv->d_timebase );
28.     }
29.     else
30.     {
31.         dts = convert_timebase_ms( p_picture->i_dts + p_flv->i_delay_time, p_flv->d_timebase );
32.         cts = convert_timebase_ms( p_picture->i_pts + p_flv->i_delay_time, p_flv->d_timebase );
33.     }
34.     offset = cts - dts;
35.
36.     if( p_flv->i_framenum )
37.     {
38.         if( p_flv->i_prev_dts == dts )
39.             x264_cli_log( "flv", X264_LOG_WARNING, "duplicate DTS %"PRIu64" generated by rounding\n",
40.                 "          decoding framerate cannot exceed 1000fps\n", dts );
41.         if( p_flv->i_prev_cts == cts )
42.             x264_cli_log( "flv", X264_LOG_WARNING, "duplicate CTS %"PRIu64" generated by rounding\n",
43.                 "          composition framerate cannot exceed 1000fps\n", cts );
44.     }
45.     p_flv->i_prev_dts = dts;
46.     p_flv->i_prev_cts = cts;
47.
48.     // A new frame - write packet header
49.     flv_put_byte( c, FLV_TAG_TYPE_VIDEO );
50.     flv_put_be24( c, 0 ); // calculated later
51.     flv_put_be24( c, dts );
52.     flv_put_byte( c, dts >> 24 );
53.     flv_put_be24( c, 0 );
54.
55.     p_flv->start = c->d_cur;
56.     flv_put_byte( c, p_picture->b_keyframe ? FLV_FRAME_KEY : FLV_FRAME_INTER );
57.     flv_put_byte( c, 1 ); // AVC NALU
58.     flv_put_be24( c, offset );
59.
60.     if( p_flv->sei )
61.     {
62.         flv_append_data( c, p_flv->sei, p_flv->sei_len );
63.         free( p_flv->sei );
64.         p_flv->sei = NULL;
65.     }
66.     flv_append_data( c, p_nalu, i_size );
67.
68.     unsigned length = c->d_cur - p_flv->start;
69.     flv_rewrite_amf_be24( c, length, p_flv->start - 10 );
70.     flv_put_be32( c, 11 + length ); // Last tag size
71.     CHECK( flv_flush_data( c ) );
72.
73.     p_flv->i_framenum++;
74.
75.     return i_size;
76. }

```

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。

```

1.  typedef struct
2.  {
3.      int (*open_file)( char *psz_filename, hnd_t *p_handle, video_info_t *info, cli_input_opt_t *opt );
4.      int (*picture_alloc)( cli_pic_t *pic, int csp, int width, int height );
5.      int (*read_frame)( cli_pic_t *pic, hnd_t handle, int i_frame );
6.      int (*release_frame)( cli_pic_t *pic, hnd_t handle );
7.      void (*picture_clean)( cli_pic_t *pic );
8.      int (*close_file)( hnd_t handle );
9.  } cli_input_t;
10.
11. extern const cli_input_t raw_input;
12. extern const cli_input_t y4m_input;
13. extern const cli_input_t avs_input;
14. extern const cli_input_t lavf_input;
15. 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\_input\_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inputraw.c，该文件内容如下所示。

```

1.  #include "input.h"
2.  #define FAIL_IF_ERROR( cond, ... ) FAIL_IF_ERR( cond, __VA_ARGS__ )
3.
4.  typedef struct
5.  {
6.      FILE *fh;
7.      int next_frame;
8.      uint64_t plane_size[4];
9.      uint64_t frame_size;
10.     int bit_depth;
11. } raw_hnd_t;
12.
13. //打开raw YUV格式文件
14. static int open_file( char *psz_filename, hnd_t *p_handle, video_info_t *info, cli_input_opt_t *opt )
15. {
16.     raw_hnd_t *h = calloc( 1, sizeof(raw_hnd_t) );
17.     if( !h )
18.         return -1;
19.
20.     if( !opt->resolution )
21.     {
22.         //如果没有设置分辨率
23.         //尝试从文件名中解析分辨率
24.         /* try to parse the file name */
25.         for( char *p = psz_filename; *p; p++ )
26.             if( *p >= '0' && *p <= '9' && sscanf( p, "%dx%d", &info->width, &info->height ) == 2 )
27.                 break;
28.     }
29.     else
30.         sscanf( opt->resolution, "%dx%d", &info->width, &info->height );
31.     //没有分辨率信息的话，会弹出错误信息
32.     FAIL_IF_ERROR( !info->width || !info->height, "raw input requires a resolution.\n" )
33.     //设置颜色空间
34.     if( opt->colorspace )
35.     {
36.         for( info->csp = X264_CSP_CLI_MAX-1; info->csp > X264_CSP_NONE; info->csp-- )
37.         {
38.             if( x264_cli_csps[info->csp].name && !strcasecmp( x264_cli_csps[info->csp].name, opt->colorspace ) )
39.                 break;
40.         }
41.         FAIL_IF_ERROR( info->csp == X264_CSP_NONE, "unsupported colorspace '%s'\n", opt->colorspace );
42.     }
43.     else /* default */
44.         info->csp = X264_CSP_I420; //默认为YUV420P
45.     //颜色位深
46.     h->bit_depth = opt->bit_depth;
47.     FAIL_IF_ERROR( h->bit_depth < 8 || h->bit_depth > 16, "unsupported bit depth '%d'\n", h->bit_depth );
48.     if( h->bit_depth > 8 )
49.         info->csp |= X264_CSP_HIGH_DEPTH;
50.
51.     if( !strcmp( psz_filename, "-" ) )
52.         h->fh = stdin; //从管道输入
53.     else
54.         h->fh = x264_fopen( psz_filename, "rb" ); //打开文件
55.     if( h->fh == NULL )
56.         return -1;
57.
58.     info->thread_safe = 1;
59.     info->num_frames = 0;
60.     info->vfr = 0;
61.
62.     const x264_cli_csp_t *csp = x264_cli_get_csp( info->csp );

```

```

63.     for( int i = 0; i < csp->planes; i++ )
64.     {
65.         h->plane_size[i] = x264_cli_pic_plane_size( info->csp, info->width, info->height, i );
66.         h->frame_size += h->plane_size[i];
67.         /* x264_cli_pic_plane_size returns the size in bytes, we need the value in pixels from here on */
68.         h->plane_size[i] /= x264_cli_csp_depth_factor( info->csp );
69.     }
70.
71.     if( x264_is_regular_file( h->fh ) )
72.     {
73.         fseek( h->fh, 0, SEEK_END );
74.         uint64_t size = ftell( h->fh );
75.         fseek( h->fh, 0, SEEK_SET );
76.         info->num_frames = size / h->frame_size;
77.     }
78.
79.     *p_handle = h;
80.     return 0;
81. }
82.
83. //读取一帧数据-内部
84. static int read_frame_internal( cli_pic_t *pic, raw_hnd_t *h, int bit_depth_uc )
85. {
86.     int error = 0;
87.     int pixel_depth = x264_cli_csp_depth_factor( pic->img.csp );
88.     //一个分量一个分量读
89.     for( int i = 0; i < pic->img.planes && !error; i++ )
90.     {
91.         //fread()读取
92.         error |= fread( pic->img.plane[i], pixel_depth, h->plane_size[i], h->fh ) != h->plane_size[i];
93.         if( bit_depth_uc )
94.         {
95.             /* upconvert non 16bit high depth planes to 16bit using the same
96.              * algorithm as used in the depth filter. */
97.             uint16_t *plane = (uint16_t*)pic->img.plane[i];
98.             uint64_t pixel_count = h->plane_size[i];
99.             int lshift = 16 - h->bit_depth;
100.            for( uint64_t j = 0; j < pixel_count; j++ )
101.                plane[j] = plane[j] << lshift;
102.        }
103.    }
104.    return error;
105. }
106. //读取一帧数据
107. static int read_frame( cli_pic_t *pic, hnd_t handle, int i_frame )
108. {
109.     raw_hnd_t *h = handle;
110.
111.     if( i_frame > h->next_frame )
112.     {
113.         if( x264_is_regular_file( h->fh ) )
114.             fseek( h->fh, i_frame * h->frame_size, SEEK_SET ); //fseek()。偏移量=帧序号*帧大小。
115.         else
116.             while( i_frame > h->next_frame )
117.             {
118.                 //读取一帧数据-内部
119.                 if( read_frame_internal( pic, h, 0 ) )
120.                     return -1;
121.                 h->next_frame++;
122.             }
123.     }
124.
125.     if( read_frame_internal( pic, h, h->bit_depth & 7 ) )
126.         return -1;
127.
128.     h->next_frame = i_frame+1;
129.     return 0;
130. }
131.
132. //关闭文件
133. static int close_file( hnd_t handle )
134. {
135.     raw_hnd_t *h = handle;
136.     if( !h || !h->fh )
137.         return 0;
138.     //fclose()关闭文件
139.     fclose( h->fh );
140.     free( h );
141.     return 0;
142. }
143. //raw格式对应的数组
144. 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。

```
[cpp] 1. 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。

```
[cpp] 1. typedef struct
2. {
3.     FILE *fh;
4.     int next_frame;
5.     int seq_header_len;
6.     int frame_header_len;
7.     uint64_t frame_size;
8.     uint64_t plane_size[3];
9.     int bit_depth;
10. } y4m_hnd_t;
11.
12. #define Y4M_MAGIC "YUV4MPEG2"
13. #define MAX_YUV4_HEADER 80
14. #define Y4M_FRAME_MAGIC "FRAME"
15. #define MAX_FRAME_HEADER 80
16.
17. static int parse_csp_and_depth( char *csp_name, int *bit_depth )
18. {
19.     int csp = X264_CSP_MAX;
20.
21.     /* Set colorspace from known variants */
22.     if( !strcmp( "420", csp_name, 3 ) )
23.         csp = X264_CSP_I420;
24.     else if( !strcmp( "422", csp_name, 3 ) )
25.         csp = X264_CSP_I422;
26.     else if( !strcmp( "444", csp_name, 3 ) && strcmp( "444alpha", csp_name, 8 ) ) // only accept alphaless 4:4:4
27.         csp = X264_CSP_I444;
28.
29.     /* Set high bit depth from known extensions */
30.     if( sscanf( csp_name, "%d*[pP]%d", bit_depth ) != 1 )
31.         *bit_depth = 8;
32.
33.     return csp;
34. }
35.
36. static int open_file( char *psz_filename, hnd_t *p_handle, video_info_t *info, cli_input_opt_t *opt )
37. {
38.     y4m_hnd_t *h = malloc( sizeof(y4m_hnd_t) );
39.     int i;
40.     uint32_t n, d;
41.     char header[MAX_YUV4_HEADER+10];
42.     char *tokend, *header_end;
43.     int colorspace = X264_CSP_NONE;
44.     int alt_colorspace = X264_CSP_NONE;
45.     int alt_bit_depth = 8;
46.     if( !h )
47.         return -1;
48.
49.     h->next_frame = 0;
50.     info->vfr = 0;
51.
52.     if( !strcmp( psz_filename, "-" ) )
53.         h->fh = stdin;
54.     else
55.         h->fh = x264_fopen(psz_filename, "rb");
56.     if( h->fh == NULL )
57.         return -1;
58.
59.     h->frame_header_len = strlen( Y4M_FRAME_MAGIC )+1;
60.
61.     /* Read header */
62.     //解析Y4M格式的文件头
63.     for( i = 0; i < MAX_YUV4_HEADER; i++ )
64.     {
65.         header[i] = fgetc( h->fh );
66.         if( header[i] == '\n' )
67.         {
68.             /* Add a space after last option. Makes parsing "444" vs
69.             "444alpha" easier. */
70.             header[i+1] = 0x20;
71.             header[i+2] = 0;
72.             break;
73.         }
74.     }
75.     if( i == MAX_YUV4_HEADER || strcmp( header, Y4M_MAGIC, strlen( Y4M_MAGIC ) ) )
76.         return -1;
77.
78.     /* Scan properties */
79.     header_end = &header[i+1]; /* Include space */
80.     h->seq_header_len = i+1;
```

```

81. for( char *tokstart = &header[strlen( Y4M_MAGIC )+1]; tokstart < header_end; tokstart++ )
82. {
83.     if( *tokstart == 0x20 )
84.         continue;
85.     switch( *tokstart++ )
86.     {
87.         case 'W': /* Width. Required. */
88.             info->width = strtol( tokstart, &tokend, 10 );
89.             tokstart=tokend;
90.             break;
91.         case 'H': /* Height. Required. */
92.             info->height = strtol( tokstart, &tokend, 10 );
93.             tokstart=tokend;
94.             break;
95.         case 'C': /* Color space */
96.             colorspace = parse_csp_and_depth( tokstart, &h->bit_depth );
97.             tokstart = strchr( tokstart, 0x20 );
98.             break;
99.         case 'I': /* Interlace type */
100.            switch( *tokstart++ )
101.            {
102.                case 't':
103.                    info->interlaced = 1;
104.                    info->tff = 1;
105.                    break;
106.                case 'b':
107.                    info->interlaced = 1;
108.                    info->tff = 0;
109.                    break;
110.                case 'm':
111.                    info->interlaced = 1;
112.                    break;
113.                //case '?':
114.                //case 'p':
115.                default:
116.                    break;
117.            }
118.            break;
119.         case 'F': /* Frame rate - 0:0 if unknown */
120.             if( sscanf( tokstart, "%u:%u", &n, &d ) == 2 && n && d )
121.             {
122.                 x264_reduce_fraction( &n, &d );
123.                 info->fps_num = n;
124.                 info->fps_den = d;
125.             }
126.             tokstart = strchr( tokstart, 0x20 );
127.             break;
128.         case 'A': /* Pixel aspect - 0:0 if unknown */
129.             /* Don't override the aspect ratio if sar has been explicitly set on the commandline. */
130.             if( sscanf( tokstart, "%u:%u", &n, &d ) == 2 && n && d )
131.             {
132.                 x264_reduce_fraction( &n, &d );
133.                 info->sar_width = n;
134.                 info->sar_height = d;
135.             }
136.             tokstart = strchr( tokstart, 0x20 );
137.             break;
138.         case 'X': /* Vendor extensions */
139.             if( !strcmp( "YSCSS=", tokstart, 6 ) )
140.             {
141.                 /* Older nonstandard pixel format representation */
142.                 tokstart += 6;
143.                 alt_colorspace = parse_csp_and_depth( tokstart, &alt_bit_depth );
144.             }
145.             tokstart = strchr( tokstart, 0x20 );
146.             break;
147.     }
148. }
149.
150. if( colorspace == X264_CSP_NONE )
151. {
152.     colorspace = alt_colorspace;
153.     h->bit_depth = alt_bit_depth;
154. }
155.
156. // default to 8bit 4:2:0 if nothing is specified
157. if( colorspace == X264_CSP_NONE )
158. {
159.     colorspace = X264_CSP_I420;
160.     h->bit_depth = 8;
161. }
162.
163. FAIL_IF_ERROR( colorspace <= X264_CSP_NONE || colorspace >= X264_CSP_MAX, "colorspace unhandled\n" )
164. FAIL_IF_ERROR( h->bit_depth < 8 || h->bit_depth > 16, "unsupported bit depth `%d'\n", h->bit_depth );
165.
166. info->thread_safe = 1;
167. info->num_frames = 0;
168. info->csp = colorspace;
169. h->frame_size = h->frame_header_len;
170.
171. if( h->bit_depth > 8 )

```



```

172.         info->csp |= X264_CSP_HIGH_DEPTH;
173.
174.         const x264_cli_csp_t *csp = x264_cli_get_csp( info->csp );
175.
176.         for( i = 0; i < csp->planes; i++ )
177.         {
178.             h->plane_size[i] = x264_cli_pic_plane_size( info->csp, info->width, info->height, i );
179.             h->frame_size += h->plane_size[i];
180.             /* x264_cli_pic_plane_size returns the size in bytes, we need the value in pixels from here on */
181.             h->plane_size[i] /= x264_cli_csp_depth_factor( info->csp );
182.         }
183.
184.         /* Most common case: frame_header = "FRAME" */
185.         if( x264_is_regular_file( h->fh ) )
186.         {
187.             uint64_t init_pos = ftell( h->fh );
188.             fseek( h->fh, 0, SEEK_END );
189.             uint64_t i_size = ftell( h->fh );
190.             fseek( h->fh, init_pos, SEEK_SET );
191.             info->num_frames = (i_size - h->seq_header_len) / h->frame_size;
192.         }
193.
194.         *p_handle = h;
195.         return 0;
196.     }

```

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项目中和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\video\video.h，如下所示。

```

1.  struct cli_vid_filter_t
2.  {
3.      /* name of the filter */
4.      const char *name;
5.      /* help: a short message on what the filter does and how to use it.
6.       * this should only be implemented by filters directly accessible by the user */
7.      void (*help)( int longhelp );
8.      /* init: initializes the filter given the input clip properties and parameter to adjust them as necessary
9.       * with the given options provided by the user.
10.       * returns 0 on success, nonzero on error. */
11.      int (*init)( hnd_t *handle, cli_vid_filter_t *filter, video_info_t *info, x264_param_t *param, char *opt_string );
12.      /* get_frame: given the storage for the output frame and desired frame number, generate the frame accordingly.
13.       * the image data returned by get_frame should be treated as const and not be altered.
14.       * returns 0 on success, nonzero on error. */
15.      int (*get_frame)( hnd_t handle, cli_pic_t *output, int frame );
16.      /* release_frame: frame is done being used and is signaled for cleanup.
17.       * returns 0 on success, nonzero on error. */
18.      int (*release_frame)( hnd_t handle, cli_pic_t *pic, int frame );
19.      /* free: run filter cleanup procedures. */
20.      void (*free)( hnd_t handle );
21.      /* next registered filter, unused by filters themselves */
22.      cli_vid_filter_t *next;
23.  };

```

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。

```

1. #include "video.h"
2.
3. /* This filter converts the demuxer API into the filtering API for video frames.
4.  * Backseeking is prohibited here as not all demuxers are capable of doing so. */
5.
6. typedef struct
7. {
8.     cli_pic_t pic;
9.     hnd_t hin;
10.    int cur_frame;
11. } source_hnd_t;
12.
13. cli_vid_filter_t source_filter;
14.
15. static int init( hnd_t *handle, cli_vid_filter_t *filter, video_info_t *info, x264_param_t *param, char *opt_string )
16. {
17.     source_hnd_t *h = calloc( 1, sizeof(source_hnd_t) );
18.     if( !h )
19.         return -1;
20.     h->cur_frame = -1;
21.
22.     if( cli_input.picture_alloc( &h->pic, info->csp, info->width, info->height ) )
23.         return -1;
24.
25.     h->hin = *handle;
26.     *handle = h;
27.     *filter = source_filter;
28.
29.     return 0;
30. }
31.
32. static int get_frame( hnd_t handle, cli_pic_t *output, int frame )
33. {
34.     source_hnd_t *h = handle;
35.     /* do not allow requesting of frames from before the current position */
36.     if( frame <= h->cur_frame || cli_input.read_frame( &h->pic, h->hin, frame ) )
37.         return -1;
38.     h->cur_frame = frame;
39.     *output = h->pic;
40.     return 0;
41. }
42.
43. static int release_frame( hnd_t handle, cli_pic_t *pic, int frame )
44. {
45.     source_hnd_t *h = handle;
46.     if( cli_input.release_frame && cli_input.release_frame( &h->pic, h->hin ) )
47.         return -1;
48.     return 0;
49. }
50.
51. static void free_filter( hnd_t handle )
52. {
53.     source_hnd_t *h = handle;
54.     cli_input.picture_clean( &h->pic );
55.     cli_input.close_file( h->hin );
56.     free( h );
57. }
58.
59. 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。

```

1. 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。

```

1. static int get_frame( hnd_t handle, cli_pic_t *output, int frame )
2. {
3.     resizer_hnd_t *h = handle;
4.     if( h->prev_filter.get_frame( h->prev_hnd, output, frame ) )
5.         return -1;
6.     if( h->variable_input && check_resizer( h, output ) )
7.         return -1;
8.     h->working = 1;
9.     if( h->pre_swap_chroma )
10.        XCHG( uint8_t*, output->img.plane[1], output->img.plane[2] );
11.     if( h->ctx )
12.     {
13.         sws_scale( h->ctx, (const uint8_t* const*)output->img.plane, output->img.stride,
14.                    0, output->img.height, h->buffer.img.plane, h->buffer.img.stride );
15.         output->img = h->buffer.img; /* copy img data */
16.     }
17.     else
18.         output->img.csp = h->dst_csp;
19.     if( h->post_swap_chroma )
20.        XCHG( uint8_t*, output->img.plane[1], output->img.plane[2] );
21.
22.     return 0;
23. }

```

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