[**FFMPEG结构体分析：AVFrame**](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

标签： [ffmpeg](http://www.csdn.net/tag/ffmpeg" \t "_blank)[AVFrame](http://www.csdn.net/tag/AVFrame)[源代码](http://www.csdn.net/tag/%e6%ba%90%e4%bb%a3%e7%a0%81)[解码](http://www.csdn.net/tag/%e8%a7%a3%e7%a0%81)[视频](http://www.csdn.net/tag/%e8%a7%86%e9%a2%91)

2013-11-06 21:15 50989人阅读 [评论](http://blog.csdn.net/leixiaohua1020/article/details/14214577#comments)(53) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/leixiaohua1020/article/details/14214577#report)

本文章已收录于：

[[](http://lib.csdn.net/base/57?source=blogtop) **直播技术知识库**](http://lib.csdn.net/base/57?source=blogtop)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

FFMPEG（137） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

注：写了一系列的结构体的分析的文章，在这里列一个列表：

[FFMPEG结构体分析：AVFrame](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

[FFMPEG结构体分析：AVFormatContext](http://blog.csdn.net/leixiaohua1020/article/details/14214705)

[FFMPEG结构体分析：AVCodecContext](http://blog.csdn.net/leixiaohua1020/article/details/14214859)

[FFMPEG结构体分析：AVIOContext](http://blog.csdn.net/leixiaohua1020/article/details/14215369)

[FFMPEG结构体分析：AVCodec](http://blog.csdn.net/leixiaohua1020/article/details/14215833)

[FFMPEG结构体分析：AVStream](http://blog.csdn.net/leixiaohua1020/article/details/14215821)  
[FFMPEG结构体分析：AVPacket](http://blog.csdn.net/leixiaohua1020/article/details/14215755)

FFMPEG有几个最重要的结构体，包含了解协议，解封装，解码操作，此前已经进行过分析：

[FFMPEG中最关键的结构体之间的关系](http://blog.csdn.net/leixiaohua1020/article/details/11693997)

在此不再详述，其中AVFrame是包含码流参数较多的结构体。本文将会详细分析一下该结构体里主要变量的含义和作用。

首先看一下结构体的定义（位于avcodec.h）：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. /\*
2. \*雷霄骅
3. \*leixiaohua1020@126.com
4. \*中国传媒大学/数字电视技术
5. \*/
6. /\*\*
7. \* Audio Video Frame.
8. \* New fields can be added to the end of AVFRAME with minor version
9. \* bumps. Similarly fields that are marked as to be only accessed by
10. \* av\_opt\_ptr() can be reordered. This allows 2 forks to add fields
11. \* without breaking compatibility with each other.
12. \* Removal, reordering and changes in the remaining cases require
13. \* a major version bump.
14. \* sizeof(AVFrame) must not be used outside libavcodec.
15. \*/
16. **typedef** **struct** AVFrame {
17. #define AV\_NUM\_DATA\_POINTERS 8
18. /\*\*图像数据
19. \* pointer to the picture/channel planes.
20. \* This might be different from the first allocated byte
21. \* - encoding: Set by user
22. \* - decoding: set by AVCodecContext.get\_buffer()
23. \*/
24. uint8\_t \*data[AV\_NUM\_DATA\_POINTERS];
26. /\*\*
27. \* Size, in bytes, of the data for each picture/channel plane.
28. \*
29. \* For audio, only linesize[0] may be set. For planar audio, each channel
30. \* plane must be the same size.
31. \*
32. \* - encoding: Set by user
33. \* - decoding: set by AVCodecContext.get\_buffer()
34. \*/
35. **int** linesize[AV\_NUM\_DATA\_POINTERS];
37. /\*\*
38. \* pointers to the data planes/channels.
39. \*
40. \* For video, this should simply point to data[].
41. \*
42. \* For planar audio, each channel has a separate data pointer, and
43. \* linesize[0] contains the size of each channel buffer.
44. \* For packed audio, there is just one data pointer, and linesize[0]
45. \* contains the total size of the buffer for all channels.
46. \*
47. \* Note: Both data and extended\_data will always be set by get\_buffer(),
48. \* but for planar audio with more channels that can fit in data,
49. \* extended\_data must be used by the decoder in order to access all
50. \* channels.
51. \*
52. \* encoding: unused
53. \* decoding: set by AVCodecContext.get\_buffer()
54. \*/
55. uint8\_t \*\*extended\_data;
57. /\*\*宽高
58. \* width and height of the video frame
59. \* - encoding: unused
60. \* - decoding: Read by user.
61. \*/
62. **int** width, height;
64. /\*\*
65. \* number of audio samples (per channel) described by this frame
66. \* - encoding: Set by user
67. \* - decoding: Set by libavcodec
68. \*/
69. **int** nb\_samples;
71. /\*\*
72. \* format of the frame, -1 if unknown or unset
73. \* Values correspond to enum AVPixelFormat for video frames,
74. \* enum AVSampleFormat for audio)
75. \* - encoding: unused
76. \* - decoding: Read by user.
77. \*/
78. **int** format;
80. /\*\*是否是关键帧
81. \* 1 -> keyframe, 0-> not
82. \* - encoding: Set by libavcodec.
83. \* - decoding: Set by libavcodec.
84. \*/
85. **int** key\_frame;
87. /\*\*帧类型（I,B,P）
88. \* Picture type of the frame, see ?\_TYPE below.
89. \* - encoding: Set by libavcodec. for coded\_picture (and set by user for input).
90. \* - decoding: Set by libavcodec.
91. \*/
92. **enum** AVPictureType pict\_type;
94. /\*\*
95. \* pointer to the first allocated byte of the picture. Can be used in get\_buffer/release\_buffer.
96. \* This isn't used by libavcodec unless the default get/release\_buffer() is used.
97. \* - encoding:
98. \* - decoding:
99. \*/
100. uint8\_t \*base[AV\_NUM\_DATA\_POINTERS];
102. /\*\*
103. \* sample aspect ratio for the video frame, 0/1 if unknown/unspecified
104. \* - encoding: unused
105. \* - decoding: Read by user.
106. \*/
107. AVRational sample\_aspect\_ratio;
109. /\*\*
110. \* presentation timestamp in time\_base units (time when frame should be shown to user)
111. \* If AV\_NOPTS\_VALUE then frame\_rate = 1/time\_base will be assumed.
112. \* - encoding: MUST be set by user.
113. \* - decoding: Set by libavcodec.
114. \*/
115. int64\_t pts;
117. /\*\*
118. \* reordered pts from the last AVPacket that has been input into the decoder
119. \* - encoding: unused
120. \* - decoding: Read by user.
121. \*/
122. int64\_t pkt\_pts;
124. /\*\*
125. \* dts from the last AVPacket that has been input into the decoder
126. \* - encoding: unused
127. \* - decoding: Read by user.
128. \*/
129. int64\_t pkt\_dts;
131. /\*\*
132. \* picture number in bitstream order
133. \* - encoding: set by
134. \* - decoding: Set by libavcodec.
135. \*/
136. **int** coded\_picture\_number;
137. /\*\*
138. \* picture number in display order
139. \* - encoding: set by
140. \* - decoding: Set by libavcodec.
141. \*/
142. **int** display\_picture\_number;
144. /\*\*
145. \* quality (between 1 (good) and FF\_LAMBDA\_MAX (bad))
146. \* - encoding: Set by libavcodec. for coded\_picture (and set by user for input).
147. \* - decoding: Set by libavcodec.
148. \*/
149. **int** quality;
151. /\*\*
152. \* is this picture used as reference
153. \* The values for this are the same as the MpegEncContext.picture\_structure
154. \* variable, that is 1->top field, 2->bottom field, 3->frame/both fields.
155. \* Set to 4 for delayed, non-reference frames.
156. \* - encoding: unused
157. \* - decoding: Set by libavcodec. (before get\_buffer() call)).
158. \*/
159. **int** reference;
161. /\*\*QP表
162. \* QP table
163. \* - encoding: unused
164. \* - decoding: Set by libavcodec.
165. \*/
166. int8\_t \*qscale\_table;
167. /\*\*
168. \* QP store stride
169. \* - encoding: unused
170. \* - decoding: Set by libavcodec.
171. \*/
172. **int** qstride;
174. /\*\*
175. \*
176. \*/
177. **int** qscale\_type;
179. /\*\*跳过宏块表
180. \* mbskip\_table[mb]>=1 if MB didn't change
181. \* stride= mb\_width = (width+15)>>4
182. \* - encoding: unused
183. \* - decoding: Set by libavcodec.
184. \*/
185. uint8\_t \*mbskip\_table;
187. /\*\*运动矢量表
188. \* motion vector table
189. \* @code
190. \* example:
191. \* int mv\_sample\_log2= 4 - motion\_subsample\_log2;
192. \* int mb\_width= (width+15)>>4;
193. \* int mv\_stride= (mb\_width << mv\_sample\_log2) + 1;
194. \* motion\_val[direction][x + y\*mv\_stride][0->mv\_x, 1->mv\_y];
195. \* @endcode
196. \* - encoding: Set by user.
197. \* - decoding: Set by libavcodec.
198. \*/
199. int16\_t (\*motion\_val[2])[2];
201. /\*\*宏块类型表
202. \* macroblock type table
203. \* mb\_type\_base + mb\_width + 2
204. \* - encoding: Set by user.
205. \* - decoding: Set by libavcodec.
206. \*/
207. uint32\_t \*mb\_type;
209. /\*\*DCT系数
210. \* DCT coefficients
211. \* - encoding: unused
212. \* - decoding: Set by libavcodec.
213. \*/
214. **short** \*dct\_coeff;
216. /\*\*参考帧列表
217. \* motion reference frame index
218. \* the order in which these are stored can depend on the codec.
219. \* - encoding: Set by user.
220. \* - decoding: Set by libavcodec.
221. \*/
222. int8\_t \*ref\_index[2];
224. /\*\*
225. \* for some private data of the user
226. \* - encoding: unused
227. \* - decoding: Set by user.
228. \*/
229. **void** \*opaque;
231. /\*\*
232. \* error
233. \* - encoding: Set by libavcodec. if flags&CODEC\_FLAG\_PSNR.
234. \* - decoding: unused
235. \*/
236. uint64\_t error[AV\_NUM\_DATA\_POINTERS];
238. /\*\*
239. \* type of the buffer (to keep track of who has to deallocate data[\*])
240. \* - encoding: Set by the one who allocates it.
241. \* - decoding: Set by the one who allocates it.
242. \* Note: User allocated (direct rendering) & internal buffers cannot coexist currently.
243. \*/
244. **int** type;
246. /\*\*
247. \* When decoding, this signals how much the picture must be delayed.
248. \* extra\_delay = repeat\_pict / (2\*fps)
249. \* - encoding: unused
250. \* - decoding: Set by libavcodec.
251. \*/
252. **int** repeat\_pict;
254. /\*\*
255. \* The content of the picture is interlaced.
256. \* - encoding: Set by user.
257. \* - decoding: Set by libavcodec. (default 0)
258. \*/
259. **int** interlaced\_frame;
261. /\*\*
262. \* If the content is interlaced, is top field displayed first.
263. \* - encoding: Set by user.
264. \* - decoding: Set by libavcodec.
265. \*/
266. **int** top\_field\_first;
268. /\*\*
269. \* Tell user application that palette has changed from previous frame.
270. \* - encoding: ??? (no palette-enabled encoder yet)
271. \* - decoding: Set by libavcodec. (default 0).
272. \*/
273. **int** palette\_has\_changed;
275. /\*\*
276. \* codec suggestion on buffer type if != 0
277. \* - encoding: unused
278. \* - decoding: Set by libavcodec. (before get\_buffer() call)).
279. \*/
280. **int** buffer\_hints;
282. /\*\*
283. \* Pan scan.
284. \* - encoding: Set by user.
285. \* - decoding: Set by libavcodec.
286. \*/
287. AVPanScan \*pan\_scan;
289. /\*\*
290. \* reordered opaque 64bit (generally an integer or a double precision float
291. \* PTS but can be anything).
292. \* The user sets AVCodecContext.reordered\_opaque to represent the input at
293. \* that time,
294. \* the decoder reorders values as needed and sets AVFrame.reordered\_opaque
295. \* to exactly one of the values provided by the user through AVCodecContext.reordered\_opaque
296. \* @deprecated in favor of pkt\_pts
297. \* - encoding: unused
298. \* - decoding: Read by user.
299. \*/
300. int64\_t reordered\_opaque;
302. /\*\*
303. \* hardware accelerator private data (FFmpeg-allocated)
304. \* - encoding: unused
305. \* - decoding: Set by libavcodec
306. \*/
307. **void** \*hwaccel\_picture\_private;
309. /\*\*
310. \* the AVCodecContext which ff\_thread\_get\_buffer() was last called on
311. \* - encoding: Set by libavcodec.
312. \* - decoding: Set by libavcodec.
313. \*/
314. **struct** AVCodecContext \*owner;
316. /\*\*
317. \* used by multithreading to store frame-specific info
318. \* - encoding: Set by libavcodec.
319. \* - decoding: Set by libavcodec.
320. \*/
321. **void** \*thread\_opaque;
323. /\*\*
324. \* log2 of the size of the block which a single vector in motion\_val represents:
325. \* (4->16x16, 3->8x8, 2-> 4x4, 1-> 2x2)
326. \* - encoding: unused
327. \* - decoding: Set by libavcodec.
328. \*/
329. uint8\_t motion\_subsample\_log2;
331. /\*\*（音频）采样率
332. \* Sample rate of the audio data.
333. \*
334. \* - encoding: unused
335. \* - decoding: read by user
336. \*/
337. **int** sample\_rate;
339. /\*\*
340. \* Channel layout of the audio data.
341. \*
342. \* - encoding: unused
343. \* - decoding: read by user.
344. \*/
345. uint64\_t channel\_layout;
347. /\*\*
348. \* frame timestamp estimated using various heuristics, in stream time base
349. \* Code outside libavcodec should access this field using:
350. \* av\_frame\_get\_best\_effort\_timestamp(frame)
351. \* - encoding: unused
352. \* - decoding: set by libavcodec, read by user.
353. \*/
354. int64\_t best\_effort\_timestamp;
356. /\*\*
357. \* reordered pos from the last AVPacket that has been input into the decoder
358. \* Code outside libavcodec should access this field using:
359. \* av\_frame\_get\_pkt\_pos(frame)
360. \* - encoding: unused
361. \* - decoding: Read by user.
362. \*/
363. int64\_t pkt\_pos;
365. /\*\*
366. \* duration of the corresponding packet, expressed in
367. \* AVStream->time\_base units, 0 if unknown.
368. \* Code outside libavcodec should access this field using:
369. \* av\_frame\_get\_pkt\_duration(frame)
370. \* - encoding: unused
371. \* - decoding: Read by user.
372. \*/
373. int64\_t pkt\_duration;
375. /\*\*
376. \* metadata.
377. \* Code outside libavcodec should access this field using:
378. \* av\_frame\_get\_metadata(frame)
379. \* - encoding: Set by user.
380. \* - decoding: Set by libavcodec.
381. \*/
382. AVDictionary \*metadata;
384. /\*\*
385. \* decode error flags of the frame, set to a combination of
386. \* FF\_DECODE\_ERROR\_xxx flags if the decoder produced a frame, but there
387. \* were errors during the decoding.
388. \* Code outside libavcodec should access this field using:
389. \* av\_frame\_get\_decode\_error\_flags(frame)
390. \* - encoding: unused
391. \* - decoding: set by libavcodec, read by user.
392. \*/
393. **int** decode\_error\_flags;
394. #define FF\_DECODE\_ERROR\_INVALID\_BITSTREAM   1
395. #define FF\_DECODE\_ERROR\_MISSING\_REFERENCE   2
397. /\*\*
398. \* number of audio channels, only used for audio.
399. \* Code outside libavcodec should access this field using:
400. \* av\_frame\_get\_channels(frame)
401. \* - encoding: unused
402. \* - decoding: Read by user.
403. \*/
404. int64\_t channels;
405. } AVFrame;

AVFrame结构体一般用于存储原始数据（即非压缩数据，例如对视频来说是YUV，RGB，对音频来说是PCM），此外还包含了一些相关的信息。比如说，解码的时候存储了宏块类型表，QP表，运动矢量表等数据。编码的时候也存储了相关的数据。因此在使用FFMPEG进行码流分析的时候，AVFrame是一个很重要的结构体。

下面看几个主要变量的作用（在这里考虑解码的情况）：

uint8\_t \*data[AV\_NUM\_DATA\_POINTERS]：解码后原始数据（对视频来说是YUV，RGB，对音频来说是PCM）

int linesize[AV\_NUM\_DATA\_POINTERS]：data中“一行”数据的大小。注意：未必等于图像的宽，一般大于图像的宽。

int width, height：视频帧宽和高（1920x1080,1280x720...）

int nb\_samples：音频的一个AVFrame中可能包含多个音频帧，在此标记包含了几个

int format：解码后原始数据类型（YUV420，YUV422，RGB24...）

int key\_frame：是否是关键帧

enum AVPictureType pict\_type：帧类型（I,B,P...）

AVRational sample\_aspect\_ratio：宽高比（16:9，4:3...）

int64\_t pts：显示时间戳

int coded\_picture\_number：编码帧序号

int display\_picture\_number：显示帧序号

int8\_t \*qscale\_table：QP表

uint8\_t \*mbskip\_table：跳过宏块表

int16\_t (\*motion\_val[2])[2]：运动矢量表

uint32\_t \*mb\_type：宏块类型表

short \*dct\_coeff：DCT系数，这个没有提取过

int8\_t \*ref\_index[2]：运动估计参考帧列表（貌似H.264这种比较新的标准才会涉及到多参考帧）

int interlaced\_frame：是否是隔行扫描

uint8\_t motion\_subsample\_log2：一个宏块中的运动矢量采样个数，取log的

其他的变量不再一一列举，源代码中都有详细的说明。在这里重点分析一下几个需要一定的理解的变量：

**1.data[]**

对于packed格式的数据（例如RGB24），会存到data[0]里面。

对于planar格式的数据（例如YUV420P），则会分开成data[0]，data[1]，data[2]...（YUV420P中data[0]存Y，data[1]存U，data[2]存V）

具体参见：[FFMPEG 实现 YUV，RGB各种图像原始数据之间的转换（swscale）](http://blog.csdn.net/leixiaohua1020/article/details/14215391" \t "_blank)

**2.pict\_type**

包含以下类型：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. **enum** AVPictureType {
2. AV\_PICTURE\_TYPE\_NONE = 0, ///< Undefined
3. AV\_PICTURE\_TYPE\_I,     ///< Intra
4. AV\_PICTURE\_TYPE\_P,     ///< Predicted
5. AV\_PICTURE\_TYPE\_B,     ///< Bi-dir predicted
6. AV\_PICTURE\_TYPE\_S,     ///< S(GMC)-VOP MPEG4
7. AV\_PICTURE\_TYPE\_SI,    ///< Switching Intra
8. AV\_PICTURE\_TYPE\_SP,    ///< Switching Predicted
9. AV\_PICTURE\_TYPE\_BI,    ///< BI type
10. };

**3.sample\_aspect\_ratio**

宽高比是一个分数，FFMPEG中用AVRational表达分数：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. /\*\*
2. \* rational number numerator/denominator
3. \*/
4. **typedef** **struct** AVRational{
5. **int** num; ///< numerator
6. **int** den; ///< denominator
7. } AVRational;

**4.qscale\_table**

QP表指向一块内存，里面存储的是每个宏块的QP值。宏块的标号是从左往右，一行一行的来的。每个宏块对应1个QP。

qscale\_table[0]就是第1行第1列宏块的QP值；qscale\_table[1]就是第1行第2列宏块的QP值；qscale\_table[2]就是第1行第3列宏块的QP值。以此类推...

宏块的个数用下式计算：

注：宏块大小是16x16的。

每行宏块数：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. **int** mb\_stride = pCodecCtx->width/16+1

宏块的总数：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. **int** mb\_sum = ((pCodecCtx->height+15)>>4)\*(pCodecCtx->width/16+1)

**5.motion\_subsample\_log2**

1个运动矢量所能代表的画面大小（用宽或者高表示，单位是像素），注意，这里取了log2。

代码注释中给出以下数据：

4->16x16, 3->8x8, 2-> 4x4, 1-> 2x2

即1个运动矢量代表16x16的画面的时候，该值取4；1个运动矢量代表8x8的画面的时候，该值取3...以此类推

**6.motion\_val**

运动矢量表存储了一帧视频中的所有运动矢量。

该值的存储方式比较特别：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. int16\_t (\*motion\_val[2])[2];

为了弄清楚该值究竟是怎么存的，花了我好一阵子功夫...

注释中给了一段代码：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. **int** mv\_sample\_log2= 4 - motion\_subsample\_log2;
2. **int** mb\_width= (width+15)>>4;
3. **int** mv\_stride= (mb\_width << mv\_sample\_log2) + 1;
4. motion\_val[direction][x + y\*mv\_stride][0->mv\_x, 1->mv\_y];

大概知道了该数据的结构：

1.首先分为两个列表L0和L1

2.每个列表（L0或L1）存储了一系列的MV（每个MV对应一个画面，大小由**motion\_subsample\_log2**决定）

3.每个MV分为横坐标和纵坐标（x,y）

注意，在FFMPEG中MV和MB在存储的结构上是没有什么关联的，第1个MV是屏幕上左上角画面的MV（画面的大小取决于**motion\_subsample\_log2**），第2个MV是屏幕上第1行第2列的画面的MV，以此类推。因此在一个宏块（16x16）的运动矢量很有可能如下图所示（line代表一行运动矢量的个数）：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. //例如8x8划分的运动矢量与宏块的关系：
2. //-------------------------
3. //|          |            |
4. //|mv[x]     |mv[x+1]     |
5. //-------------------------
6. //|          |            |
7. //|mv[x+line]|mv[x+line+1]|
8. //-------------------------

**7.mb\_type**

宏块类型表存储了一帧视频中的所有宏块的类型。其存储方式和QP表差不多。只不过其是uint32类型的，而QP表是uint8类型的。每个宏块对应一个宏块类型变量。

宏块类型如下定义所示：

**[cpp]** [view plain](http://blog.csdn.net/leixiaohua1020/article/details/14214577) [copy](http://blog.csdn.net/leixiaohua1020/article/details/14214577)

1. //The following defines may change, don't expect compatibility if you use them.
2. #define MB\_TYPE\_INTRA4x4   0x0001
3. #define MB\_TYPE\_INTRA16x16 0x0002 //FIXME H.264-specific
4. #define MB\_TYPE\_INTRA\_PCM  0x0004 //FIXME H.264-specific
5. #define MB\_TYPE\_16x16      0x0008
6. #define MB\_TYPE\_16x8       0x0010
7. #define MB\_TYPE\_8x16       0x0020
8. #define MB\_TYPE\_8x8        0x0040
9. #define MB\_TYPE\_INTERLACED 0x0080
10. #define MB\_TYPE\_DIRECT2    0x0100 //FIXME
11. #define MB\_TYPE\_ACPRED     0x0200
12. #define MB\_TYPE\_GMC        0x0400
13. #define MB\_TYPE\_SKIP       0x0800
14. #define MB\_TYPE\_P0L0       0x1000
15. #define MB\_TYPE\_P1L0       0x2000
16. #define MB\_TYPE\_P0L1       0x4000
17. #define MB\_TYPE\_P1L1       0x8000
18. #define MB\_TYPE\_L0         (MB\_TYPE\_P0L0 | MB\_TYPE\_P1L0)
19. #define MB\_TYPE\_L1         (MB\_TYPE\_P0L1 | MB\_TYPE\_P1L1)
20. #define MB\_TYPE\_L0L1       (MB\_TYPE\_L0   | MB\_TYPE\_L1)
21. #define MB\_TYPE\_QUANT      0x00010000
22. #define MB\_TYPE\_CBP        0x00020000
23. //Note bits 24-31 are reserved for codec specific use (h264 ref0, mpeg1 0mv, ...)

一个宏块如果包含上述定义中的一种或两种类型，则其对应的宏块变量的对应位会被置1。  
注：一个宏块可以包含好几种类型，但是有些类型是不能重复包含的，比如说一个宏块不可能既是16x16又是8x8。

**8.ref\_index**

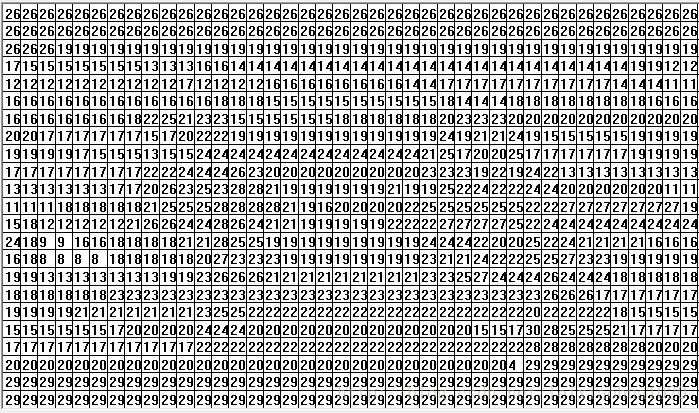
运动估计参考帧列表存储了一帧视频中所有宏块的参考帧索引。这个列表其实在比较早的压缩编码标准中是没有什么用的。只有像H.264这样的编码标准才有多参考帧的概念。但是这个字段目前我还没有研究透。只是知道每个宏块包含有4个该值，该值反映的是参考帧的索引。以后有机会再进行细研究吧。

**在这里展示一下自己做的码流分析软件的运行结果。将上文介绍的几个列表图像化显示了出来（在这里是使用MFC的绘图函数画出来的）**

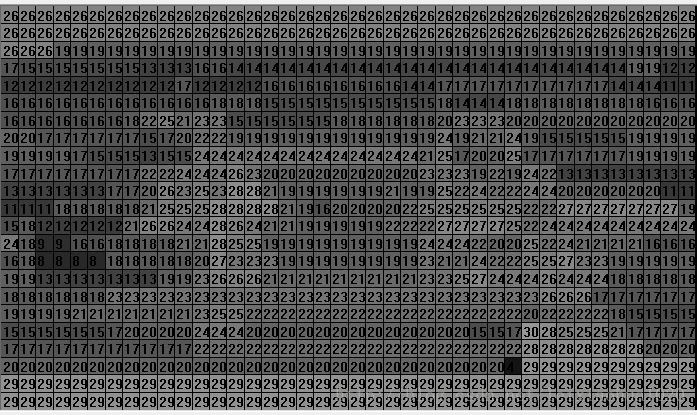
视频帧：



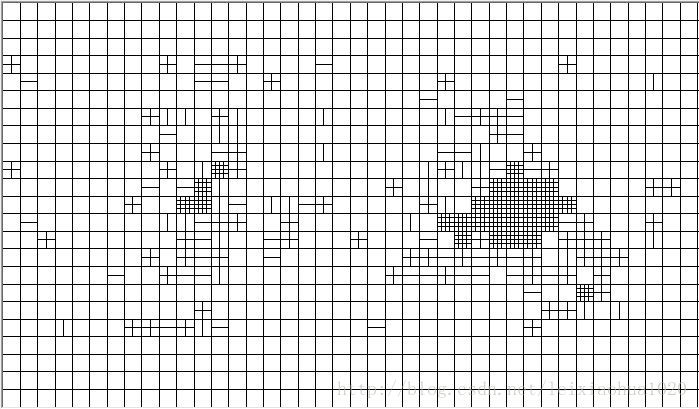
QP参数提取的结果：



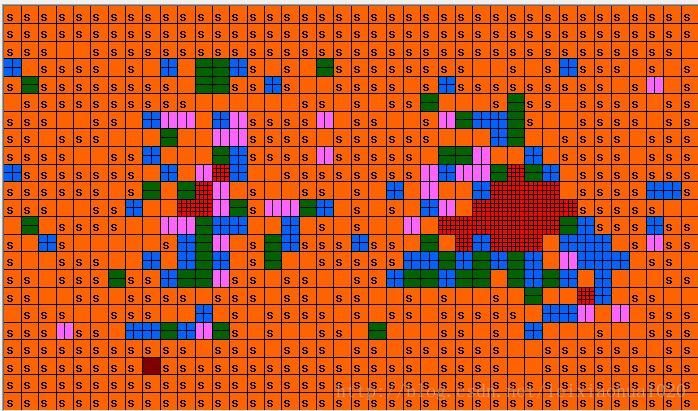
美化过的（加上了颜色）：



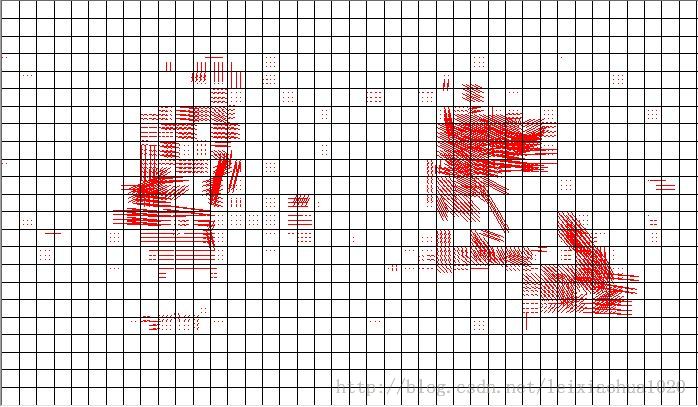
宏块类型参数提取的结果：



美化过的（加上了颜色，更清晰一些，s代表skip宏块）：



运动矢量参数提取的结果（在这里是List0）：



运动估计参考帧参数提取的结果：

