

音视频数据处理入门：H.264视频码流解析

2016年01月31日 00:19:50 阅读数：90820

音视频数据处理入门系列文章：

[音视频数据处理入门：RGB、YUV像素数据处理](#)

[音视频数据处理入门：PCM音频采样数据处理](#)

[音视频数据处理入门：H.264视频码流解析](#)

[音视频数据处理入门：AAC音频码流解析](#)

[音视频数据处理入门：FLV封装格式解析](#)

[音视频数据处理入门：UDP-RTP协议解析](#)

前两篇文章介绍的YUV/RGB处理程序以及PCM处理程序都属于音视频原始数据的处理程序。从本文开始介绍音视频码流的处理程序。本文介绍的程序是视频码流处理程序。视频码流在视频播放器中的位置如下所示。



本文中的程序是一个H.264码流解析程序。该程序可以从H.264码流中分析得到它的基本单元NALU，并且可以简单解析NALU首部的字段。通过修改该程序可以实现不同的H.264码流处理功能。

原理

H.264原始码流（又称为“裸流”）是由一个一个的NALU组成的。他们的结构如下图所示。





其中每个NALU之间通过startcode（起始码）进行分隔，起始码分成两种：0x000001（3Byte）或者0x00000001（4Byte）。如果NALU对应的Slice为一帧的开始就用0x00000001，否则就用0x000001。

H.264码流解析的步骤就是首先从码流中搜索0x000001和0x00000001，分离出NALU；然后再分析NALU的各个字段。本文的程序即实现了上述的两个步骤。

代码

整个程序位于simplest_h264_parser()函数中，如下所示。

```
[cpp]  
1.  /**
2.  * 最简单的视音频数据处理示例
3.  * Simplest MediaData Test
4.  *
5.  * 雷霄骅 Lei Xiaohua
6.  * leixiaohua1020@126.com
7.  * 中国传媒大学/数字电视技术
8.  * Communication University of China / Digital TV Technology
9.  * http://blog.csdn.net/leixiaohua1020
10. *
11. * 本项目包含如下几种视音频测试示例：
12. * (1) 像素数据处理程序。包含RGB和YUV像素格式处理的函数。
13. * (2) 音频采样数据处理程序。包含PCM音频采样格式处理的函数。
14. * (3) H.264码流分析程序。可以分离并解析NALU。
15. * (4) AAC码流分析程序。可以分离并解析ADTS帧。
16. * (5) FLV封装格式分析程序。可以将FLV中的MP3音频码流分离出来。
17. * (6) UDP-RTP协议分析程序。可以将分析UDP/RTP/MPEG-TS数据包。
18. *
19. * This project contains following samples to handling multimedia data:
20. * (1) Video pixel data handling program. It contains several examples to handle RGB and YUV data.
21. * (2) Audio sample data handling program. It contains several examples to handle PCM data.
22. * (3) H.264 stream analysis program. It can parse H.264 bitstream and analysis NALU of stream.
23. * (4) AAC stream analysis program. It can parse AAC bitstream and analysis ADTS frame of stream.
24. * (5) FLV format analysis program. It can analysis FLV file and extract MP3 audio stream.
25. * (6) UDP-RTP protocol analysis program. It can analysis UDP/RTP/MPEG-TS Packet.
26. *
27. */
28. #include <stdio.h>
29. #include <stdlib.h>
30. #include <string.h>
31.
32. typedef enum {
33.     NALU_TYPE_SLICE    = 1,
34.     NALU_TYPE_DPA      = 2,
35.     NALU_TYPE_DPB      = 3,
36.     NALU_TYPE_DPC      = 4,
37.     NALU_TYPE_IDR      = 5,
38.     NALU_TYPE_SEI      = 6,
39.     NALU_TYPE_SPS      = 7,
40.     NALU_TYPE_PPS      = 8,
41.     NALU_TYPE_AUD      = 9,
42.     NALU_TYPE_EOSEQ    = 10,
43.     NALU_TYPE_EOSTREAM = 11,
44.     NALU_TYPE_FILL     = 12,
45. } NaluType;
46.
47. typedef enum {
48.     NALU_PRIORITY_DISPOSABLE = 0,
49.     NALU_PRIORITY_LOW        = 1,
50.     NALU_PRIORITY_HIGH       = 2,
51.     NALU_PRIORITY_HIGHEST    = 3
52. } NaluPriority;
53.
54.
55. typedef struct
56. {
57.     int startcodeprefix_len;    //!< 4 for parameter sets and first slice in picture, 3 for everything else (suggested)
58.     unsigned len;              //!< Length of the NAL unit (Excluding the start code, which does not belong to the NALU)
59.     unsigned max_size;         //!< Nal Unit Buffer size
60.     int forbidden_bit;         //!< should be always FALSE
61.     int nal_reference_idc;      //!< NALU_PRIORITY_xxxx
62.     int nal_unit_type;          //!< NALU_TYPE_xxxx
63.     char *buf;                 //!< contains the first byte followed by the EBSP
64. } NALU_t;
65.
66. FILE *h264bitstream = NULL;    //!<< the bit stream file
67.
68. int info2=0, info3=0;
69.
70. static int FindStartCode2 (unsigned char *Buf){
71.     if(Buf[0]!=0 || Buf[1]!=0 || Buf[2] !=1) return 0; //0x000001?
72.     else return 1;
73. }
74.
75. static int FindStartCode3 (unsigned char *Buf){
76.     if(Buf[0]!=0 || Buf[1]!=0 || Buf[2] !=0 || Buf[3] !=1) return 0; //0x00000001?
77.     else return 1;
78. }
79.
80.
81. int GetAnnexbNALU (NALU_t *nalu){
82.     int pos = 0;
83.     int StartCodeFound, rewind;
84.     unsigned char *Buf;
85.
86.     if ((Buf = (unsigned char*)calloc (nalu->max_size , sizeof(char))) == NULL)
87.         printf ("GetAnnexbNALU: Could not allocate Buf memory\n");
88. }
```

```

88.
89.     nalu->startcodeprefix_len=3;
90.
91.     if (3 != fread (Buf, 1, 3, h264bitstream)){
92.         free(Buf);
93.         return 0;
94.     }
95.     info2 = FindStartCode2 (Buf);
96.     if(info2 != 1) {
97.         if(1 != fread(Buf+3, 1, 1, h264bitstream)){
98.             free(Buf);
99.             return 0;
100.        }
101.        info3 = FindStartCode3 (Buf);
102.        if (info3 != 1){
103.            free(Buf);
104.            return -1;
105.        }
106.        else {
107.            pos = 4;
108.            nalu->startcodeprefix_len = 4;
109.        }
110.    }
111.    else{
112.        nalu->startcodeprefix_len = 3;
113.        pos = 3;
114.    }
115.    StartCodeFound = 0;
116.    info2 = 0;
117.    info3 = 0;
118.
119.    while (!StartCodeFound){
120.        if (feof (h264bitstream)){
121.            nalu->len = (pos-1)-nalu->startcodeprefix_len;
122.            memcpy (nalu->buf, &Buf[nalu->startcodeprefix_len], nalu->len);
123.            nalu->forbidden_bit = nalu->buf[0] & 0x80; //1 bit
124.            nalu->nal_reference_idc = nalu->buf[0] & 0x60; // 2 bit
125.            nalu->nal_unit_type = (nalu->buf[0]) & 0x1f; // 5 bit
126.            free(Buf);
127.            return pos-1;
128.        }
129.        Buf[pos++] = fgetc (h264bitstream);
130.        info3 = FindStartCode3(&Buf[pos-4]);
131.        if(info3 != 1)
132.            info2 = FindStartCode2(&Buf[pos-3]);
133.        StartCodeFound = (info2 == 1 || info3 == 1);
134.    }
135.
136.    // Here, we have found another start code (and read length of startcode bytes more than we should
137.    // have. Hence, go back in the file
138.    rewind = (info3 == 1)? -4 : -3;
139.
140.    if (0 != fseek (h264bitstream, rewind, SEEK_CUR)){
141.        free(Buf);
142.        printf("GetAnnexbNALU: Cannot fseek in the bit stream file");
143.    }
144.
145.    // Here the Start code, the complete NALU, and the next start code is in the Buf.
146.    // The size of Buf is pos, pos+rewind are the number of bytes excluding the next
147.    // start code, and (pos+rewind)-startcodeprefix_len is the size of the NALU excluding the start code
148.
149.    nalu->len = (pos+rewind)-nalu->startcodeprefix_len;
150.    memcpy (nalu->buf, &Buf[nalu->startcodeprefix_len], nalu->len);//
151.    nalu->forbidden_bit = nalu->buf[0] & 0x80; //1 bit
152.    nalu->nal_reference_idc = nalu->buf[0] & 0x60; // 2 bit
153.    nalu->nal_unit_type = (nalu->buf[0]) & 0x1f; // 5 bit
154.    free(Buf);
155.
156.    return (pos+rewind);
157. }
158.
159. /**
160.  * Analysis H.264 Bitstream
161.  * @param url      Location of input H.264 bitstream file.
162.  */
163. int simplest_h264_parser(char *url){
164.
165.     NALU_t *n;
166.     int buffersize=100000;
167.
168.     //FILE *myout=fopen("output_log.txt","wb+");
169.     FILE *myout=stdout;
170.
171.     h264bitstream=fopen(url, "rb+");
172.     if (h264bitstream==NULL){
173.         printf("Open file error\n");
174.         return 0;
175.     }
176.
177.     n = (NALU_t*)calloc (1, sizeof (NALU_t));
178.     if (n == NULL){
179.         printf("Alloc NALU Error\n");

```

```

179.         printf( "Alloc NALU Error\n" );
180.         return 0;
181.     }
182.
183.     n->max_size=buffer_size;
184.     n->buf = (char*)calloc (buffer_size, sizeof (char));
185.     if (n->buf == NULL){
186.         free (n);
187.         printf ("AllocNALU: n->buf");
188.         return 0;
189.     }
190.
191.     int data_offset=0;
192.     int nal_num=0;
193.     printf("-----+----- NALU Table -----+-----+\n");
194.     printf(" NUM | POS | IDC | TYPE | LEN | \n");
195.     printf("-----+-----+-----+-----+\n");
196.
197.     while(!feof(h264bitstream))
198.     {
199.         int data_lenth;
200.         data_lenth=GetAnnexbNALU(n);
201.
202.         char type_str[20]={0};
203.         switch(n->nal_unit_type){
204.             case NALU_TYPE_SLICE:sprintf(type_str,"SLICE");break;
205.             case NALU_TYPE_DPA:sprintf(type_str,"DPA");break;
206.             case NALU_TYPE_DPB:sprintf(type_str,"DPB");break;
207.             case NALU_TYPE_DPC:sprintf(type_str,"DPC");break;
208.             case NALU_TYPE_IDR:sprintf(type_str,"IDR");break;
209.             case NALU_TYPE_SEI:sprintf(type_str,"SEI");break;
210.             case NALU_TYPE_SPS:sprintf(type_str,"SPS");break;
211.             case NALU_TYPE_PPS:sprintf(type_str,"PPS");break;
212.             case NALU_TYPE_AUD:sprintf(type_str,"AUD");break;
213.             case NALU_TYPE_EOSEQ:sprintf(type_str,"EOSEQ");break;
214.             case NALU_TYPE_EOSTREAM:sprintf(type_str,"EOSTREAM");break;
215.             case NALU_TYPE_FILL:sprintf(type_str,"FILL");break;
216.         }
217.         char idc_str[20]={0};
218.         switch(n->nal_reference_idc>5){
219.             case NALU_PRIORITY_DISPOSABLE:sprintf(idc_str,"DISPOS");break;
220.             case NALU_PRIORITY_LOW:sprintf(idc_str,"LOW");break;
221.             case NALU_PRIORITY_HIGH:sprintf(idc_str,"HIGH");break;
222.             case NALU_PRIORITY_HIGHEST:sprintf(idc_str,"HIGHEST");break;
223.         }
224.
225.         fprintf(myout,"%5d| %8d| %7s| %6s| %8d|\n",nal_num,data_offset,idc_str,type_str,n->len);
226.
227.         data_offset=data_offset+data_lenth;
228.
229.         nal_num++;
230.     }
231.
232.     //Free
233.     if (n){
234.         if (n->buf){
235.             free(n->buf);
236.             n->buf=NULL;
237.         }
238.         free (n);
239.     }
240.     return 0;
241. }

```

上文中的函数调用方法如下所示。

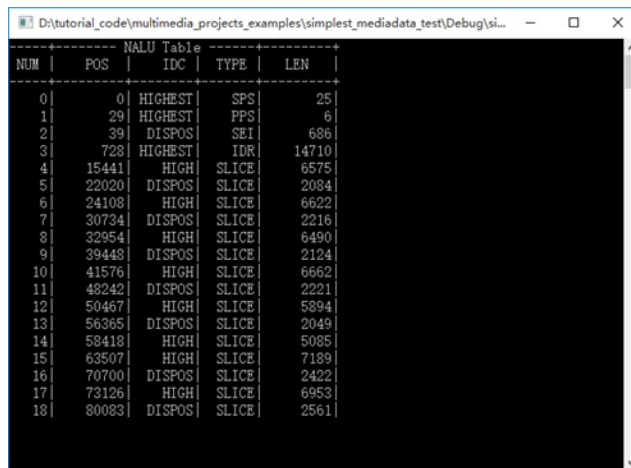
```

1. simplest_h264_parser("sintel.h264");

```

结果

本程序的输入为一个H.264原始码流（裸流）的文件路径，输出为该码流的NALU统计数据，如下图所示。



NUM	POS	IDC	TYPE	LEN
0	0	HIGHEST	SPS	25
1	29	HIGHEST	PPS	6
2	39	DISPOS	SEI	686
3	728	HIGHEST	IDR	14710
4	15441	HIGH	SLICE	6575
5	22020	DISPOS	SLICE	2084
6	24108	HIGH	SLICE	6622
7	30734	DISPOS	SLICE	2216
8	32954	HIGH	SLICE	6490
9	39448	DISPOS	SLICE	2124
10	41576	HIGH	SLICE	6662
11	48242	DISPOS	SLICE	2221
12	50467	HIGH	SLICE	5894
13	56365	DISPOS	SLICE	2049
14	58418	HIGH	SLICE	5085
15	63507	HIGH	SLICE	7189
16	70700	DISPOS	SLICE	2422
17	73126	HIGH	SLICE	6953
18	80083	DISPOS	SLICE	2561

下载

Simplest mediadata test

项目主页

SourceForge：<https://sourceforge.net/projects/simplest-mediadata-test/>

Github：https://github.com/leixiaohua1020/simplest_mediadata_test

开源中国：http://git.oschina.net/leixiaohua1020/simplest_mediadata_test

CSDN下载地址：<http://download.csdn.net/detail/leixiaohua1020/9422409>

本项目包含如下几种视音频数据解析示例：

- (1)像素数据处理程序。包含RGB和YUV像素格式处理的函数。
- (2)音频采样数据处理程序。包含PCM音频采样格式处理的函数。
- (3)H.264码流分析程序。可以分离并解析NALU。
- (4)AAC码流分析程序。可以分离并解析ADTS帧。
- (5)FLV封装格式分析程序。可以将FLV中的MP3音频码流分离出来。
- (6)UDP-RTP协议分析程序。可以将分析UDP/RTP/MPEG-TS数据包。

雷霄骅 (Lei Xiaohua)

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/50534369>

文章标签：[H.264](#) [视频码流](#) [分析](#)

个人分类：[我的开源项目](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com