

原 最简单的视频编码器：基于libvpx（编码YUV为VP8）

2014年12月24日 00:24:22 阅读量：12966

最简单的视频编码器系列文章列表：

[最简单的视频编码器：编译](#)

[最简单的视频编码器：基于libx264（编码YUV为H.264）](#)

[最简单的视频编码器：基于libx265（编码YUV为H.265）](#)

[最简单的视频编码器：libvpx（编码YUV为VP8）](#)

本文记录一个最简单的基于libvpx的VP8视频编码器。这个例子是从官方的示例代码中精简出来的例子。我发现与H.264不同，VP8的裸流（即不包含封装格式的纯视频数据流）是不能播放的。换言之，VP8的裸流必须存放在容器中才可以播放。官方示例代码中存储VP8视频流的封装格式是IVF。IVF这种封装格式不是很常见，相关的资料可以查询有关的文档。

此外，这个工程中的libvpx也可以编码VP9格式的视频。但是封装格式那里有点问题目前还没有解决，所以暂时没有包含编码VP9的代码。编码VP9和编码VP8的函数调用是一模一样的。

流程图

调用libvpx进行视频编码的流程图如下所示。

流程图中主要的函数如下所示。

vpx_img_alloc()：为图像结构体vpx_image_t分配内存。
vpx_codec_enc_config_default()：设置参数集结构体vpx_codec_enc_cfg_t的缺省值。
vpx_codec_enc_init()：打开编码器。
vpx_codec_encode()：编码一帧图像。
vpx_codec_get_cx_data()：获取一帧压缩编码数据。
vpx_codec_destroy()：关闭编码器。

存储数据的结构体如下所示。

vpx_image_t：存储压缩编码前的像素数据。
vpx_codec_cx_pkt_t：存储压缩编码后的码流数据。

IVF封装格式处理的函数如下所示。

write_ivf_file_header()：写IVF封装格式的文件头。
write_ivf_frame_header()：写IVF封装格式中每帧数据的帧头。

此外流程图中还包括一个“flush_encoder”模块，该模块使用的函数和编码模块是一样的。唯一的不同在于不再输入视频像素数据。它的作用是输出编码器中剩余的码流数据。

源代码

```
[cpp]
1.  /**
2.   * 最简单的基于VPX的视频编码器
3.   * Simplest VPX Encoder
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.  * 本程序精简了libvpx中的一个示例代码。
12.  * 可以YUV格式的像素数据编码为VPx(VP8/VP9)码流，是最简单的
13.  * 基于libvpx的视频编码器
14.  * 需要注意的是，编码输出的封装格式是IVF
15.  *
16.  * This example modified from an example from vpx project.
17.  * It encode YUV data to VPX(VP8/VP9) bitstream.
18.  * It's the simplest encoder example based on libvpx.
19.  */
20. #include <stdio.h>
21. #include <stdlib.h>
22.
```

```

23.
24. #define VPX_CODEC_DISABLE_COMPAT 1
25.
26. #include "vpx/vpx_encoder.h"
27. #include "vpx/vp8cx.h"
28.
29. #define interface (&vpx_codec_vp8_cx_algo)
30.
31. #define fourcc    0x30385056
32.
33. #define IVF_FILE_HDR_SZ  (32)
34. #define IVF_FRAME_HDR_SZ (12)
35.
36. static void mem_put_le16(char *mem, unsigned int val) {
37.     mem[0] = val;
38.     mem[1] = val>>8;
39. }
40.
41. static void mem_put_le32(char *mem, unsigned int val) {
42.     mem[0] = val;
43.     mem[1] = val>>8;
44.     mem[2] = val>>16;
45.     mem[3] = val>>24;
46. }
47.
48.
49. static void write_ivf_file_header(FILE *outfile,
50.                                   const vpx_codec_enc_cfg_t *cfg,
51.                                   int frame_cnt) {
52.     char header[32];
53.
54.     if(cfg->g_pass != VPX_RC_ONE_PASS && cfg->g_pass != VPX_RC_LAST_PASS)
55.         return;
56.     header[0] = 'D';
57.     header[1] = 'K';
58.     header[2] = 'I';
59.     header[3] = 'F';
60.     mem_put_le16(header+4, 0);           /* version */
61.     mem_put_le16(header+6, 32);         /* headersize */
62.     mem_put_le32(header+8, fourcc);     /* headersize */
63.     mem_put_le16(header+12, cfg->g_w);   /* width */
64.     mem_put_le16(header+14, cfg->g_h);   /* height */
65.     mem_put_le32(header+16, cfg->g_timebase.den); /* rate */
66.     mem_put_le32(header+20, cfg->g_timebase.num); /* scale */
67.     mem_put_le32(header+24, frame_cnt); /* length */
68.     mem_put_le32(header+28, 0);         /* unused */
69.
70.     fwrite(header, 1, 32, outfile);
71. }
72.
73.
74. static void write_ivf_frame_header(FILE *outfile,
75.                                    const vpx_codec_cx_pkt_t *pkt)
76. {
77.     char header[12];
78.     vpx_codec_pts_t pts;
79.
80.     if(pkt->kind != VPX_CODEC_CX_FRAME_PKT)
81.         return;
82.
83.     pts = pkt->data.frame.pts;
84.     mem_put_le32(header, pkt->data.frame.sz);
85.     mem_put_le32(header+4, pts&0xFFFFFFFF);
86.     mem_put_le32(header+8, pts >> 32);
87.
88.     fwrite(header, 1, 12, outfile);
89. }
90.
91. int main(int argc, char **argv) {
92.
93.     FILE *infile, *outfile;
94.     vpx_codec_ctx_t codec;
95.     vpx_codec_enc_cfg_t cfg;
96.     int frame_cnt = 0;
97.     unsigned char file_hdr[IVF_FILE_HDR_SZ];
98.     unsigned char frame_hdr[IVF_FRAME_HDR_SZ];
99.     vpx_image_t raw;
100.    vpx_codec_err_t ret;
101.    int width,height;
102.    int y_size;
103.    int frame_avail;
104.    int got_data;
105.    int flags = 0;
106.
107.    width = 640;
108.    height = 360;
109.
110.    /* Open input file for this encoding pass */
111.    infile = fopen("./cuc_ieschool_640x360_yuv420p.yuv", "rb");
112.    outfile = fopen("cuc_ieschool.ivf", "wb");
113.

```

```

114.         if(infile==NULL||outfile==NULL){
115.             printf("Error open files.\n");
116.             return -1;
117.         }
118.
119.         if(!vpx_img_alloc(&raw, VPX_IMG_FMT_I420, width, height, 1)){
120.             printf("Fail to allocate image\n");
121.             return -1;
122.         }
123.
124.         printf("Using %s\n",vpx_codec_iface_name(interface));
125.
126.         /* Populate encoder configuration */
127.         ret = vpx_codec_enc_config_default(interface, &cfg, 0);
128.         if(ret) {
129.             printf("Failed to get config: %s\n", vpx_codec_err_to_string(ret));
130.             return -1;
131.         }
132.
133.         /* Update the default configuration with our settings */
134.         cfg.rc_target_bitrate = 800;
135.         cfg.g_w = width;
136.         cfg.g_h = height;
137.
138.         write_ivf_file_header(outfile, &cfg, 0);
139.
140.         /* Initialize codec */
141.         if(vpx_codec_enc_init(&codec, interface, &cfg, 0)){
142.             printf("Failed to initialize encoder\n");
143.             return -1;
144.         }
145.
146.         frame_avail = 1;
147.         got_data = 0;
148.
149.         y_size=cfg.g_w*cfg.g_h;
150.
151.         while(frame_avail || got_data) {
152.             vpx_codec_iter_t iter = NULL;
153.             const vpx_codec_cx_pkt_t *pkt;
154.
155.             if(fread(raw.planes[0], 1, y_size*3/2, infile)!=y_size*3/2){
156.                 frame_avail=0;
157.             }
158.
159.             if(frame_avail){
160.                 ret=vpx_codec_encode(&codec,&raw,frame_cnt,1,flags,VPX_DL_REALTIME);
161.             }else{
162.                 ret=vpx_codec_encode(&codec,NULL,frame_cnt,1,flags,VPX_DL_REALTIME);
163.             }
164.
165.             if(ret){
166.                 printf("Failed to encode frame\n");
167.                 return -1;
168.             }
169.             got_data = 0;
170.             while( (pkt = vpx_codec_get_cx_data(&codec, &iter)) ) {
171.                 got_data = 1;
172.                 switch(pkt->kind) {
173.                     case VPX_CODEC_CX_FRAME_PKT:
174.                         write_ivf_frame_header(outfile, pkt);
175.                         fwrite(pkt->data.frame.buf, 1, pkt->data.frame.sz,outfile);
176.                         break;
177.                     default:
178.                         break;
179.                 }
180.             }
181.             printf("Succeed encode frame: %5d\n",frame_cnt);
182.             frame_cnt++;
183.         }
184.
185.         fclose(infile);
186.
187.         vpx_codec_destroy(&codec);
188.
189.         /* Try to rewrite the file header with the actual frame count */
190.         if(!fseek(outfile, 0, SEEK_SET))
191.             write_ivf_file_header(outfile, &cfg, frame_cnt-1);
192.
193.         fclose(outfile);
194.
195.         return 0;
196.     }

```

运行结果

程序的输入为一个YUV文件（已经测试过YUV420P格式）。

输出为IVF封装格式的VP8码流文件。

VP8码流文件的信息如下所示。

下载

Simplest Encoder

项目主页

SourceForge：<https://sourceforge.net/projects/simplestencoder/>

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

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

CDSN下载地址：<http://download.csdn.net/detail/leixiaohua1020/8284105>

该解决方案包含了几个常见的编码器的使用示例：

simplest_vpx_encoder：最简单的基于libvpx的视频编码器

simplest_x264_encoder：最简单的基于libx264的视频编码器

simplest_x265_encoder：最简单的基于libx265的视频编码器

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/42079217>

文章标签：[libvpx](#) [vp8](#) [vp9](#) [视频](#) [编解码](#)

个人分类：[我的开源项目](#) [libvpx](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com