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　阅读数：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）

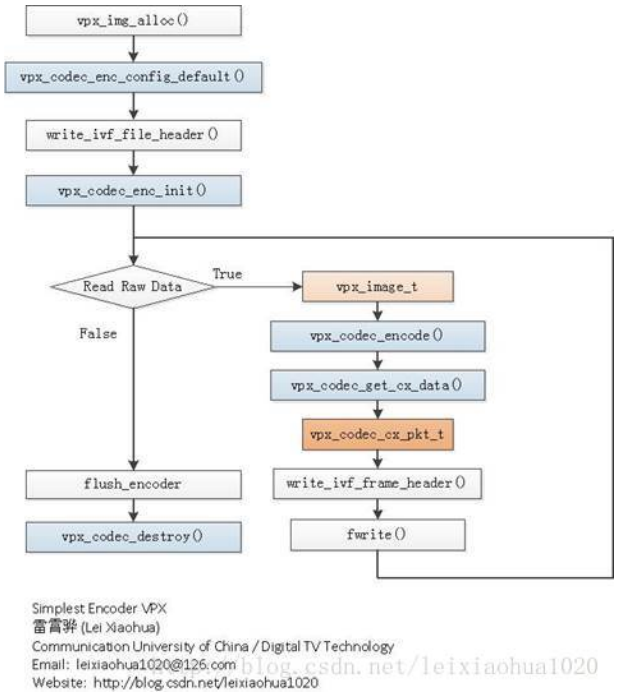
==========

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
/**
 * 最简单的基于VPX的视频编码器
 * Simplest VPX Encoder
 *
 * 雷霄骅 Lei Xiaohua
 * leixiaohua1020@126.com
 * 中国传媒大学/数字电视技术
 * Communication University of China / Digital TV Technology
 * http://blog.csdn.net/leixiaohua1020
 *
 * 本程序精简了libvpx中的一个示例代码。
 * 可以YUV格式的像素数据编码为VPx(VP8/VP9)码流，是最简单的
 * 基于libvpx的视频编码器
 * 需要注意的是，编码输出的封装格式是IVF
 *
 * This example modified from an example from vpx project.
 * It encode YUV data to VPX(VP8/VP9) bitstream.
 * It's the simplest encoder example based on libvpx.
 */
#include <stdio.h>
#include <stdlib.h>


#define VPX_CODEC_DISABLE_COMPAT 1

#include "vpx/vpx_encoder.h"
#include "vpx/vp8cx.h"

#define interface (&vpx_codec_vp8_cx_algo)

#define fourcc    0x30385056

#define IVF_FILE_HDR_SZ  (32)
#define IVF_FRAME_HDR_SZ (12)

static void mem_put_le16(char *mem, unsigned int val) {
    mem[0] = val;
    mem[1] = val>>8;
}

static void mem_put_le32(char *mem, unsigned int val) {
    mem[0] = val;
    mem[1] = val>>8;
    mem[2] = val>>16;
    mem[3] = val>>24;
}


static void write_ivf_file_header(FILE *outfile,
                                  const vpx_codec_enc_cfg_t *cfg,
                                  int frame_cnt) {
    char header[32];

    if(cfg->g_pass != VPX_RC_ONE_PASS && cfg->g_pass != VPX_RC_LAST_PASS)
        return;
    header[0] = 'D';
    header[1] = 'K';
    header[2] = 'I';
    header[3] = 'F';
    mem_put_le16(header+4,  0);                   /* version */
    mem_put_le16(header+6,  32);                  /* headersize */
    mem_put_le32(header+8,  fourcc);              /* headersize */
    mem_put_le16(header+12, cfg->g_w);            /* width */
    mem_put_le16(header+14, cfg->g_h);            /* height */
    mem_put_le32(header+16, cfg->g_timebase.den); /* rate */
    mem_put_le32(header+20, cfg->g_timebase.num); /* scale */
    mem_put_le32(header+24, frame_cnt);           /* length */
    mem_put_le32(header+28, 0);                   /* unused */

    fwrite(header, 1, 32, outfile);
}


static void write_ivf_frame_header(FILE *outfile,
                                   const vpx_codec_cx_pkt_t *pkt)
{
```

```
{
    char             header[12];
    vpx_codec_pts_t  pts;

    if(pkt->kind != VPX_CODEC_CX_FRAME_PKT)
        return;

    pts = pkt->data.frame.pts;
    mem_put_le32(header, pkt->data.frame.sz);
    mem_put_le32(header+4, pts&0xFFFFFFFF);
    mem_put_le32(header+8, pts >> 32);

    fwrite(header, 1, 12, outfile);
}

int main(int argc, char **argv) {

    FILE *infile, *outfile;
    vpx_codec_ctx_t codec;
    vpx_codec_enc_cfg_t cfg;
    int frame_cnt = 0;
    unsigned char file_hdr[IVF_FILE_HDR_SZ];
    unsigned char frame_hdr[IVF_FRAME_HDR_SZ];
    vpx_image_t raw;
    vpx_codec_err_t ret;
    int width,height;
         int y_size;
    int frame_avail;
    int got_data;
    int flags = 0;

    width = 640;
    height = 360;

         /* Open input file for this encoding pass */
         infile = fopen("../cuc_ieschool_640x360_yuv420p.yuv", "rb");
         outfile = fopen("cuc_ieschool.ivf", "wb");

         if(infile==NULL||outfile==NULL){
                  printf("Error open files.\n");
                  return -1;
         }

         if(!vpx_img_alloc(&raw, VPX_IMG_FMT_I420, width, height, 1)){
        printf("Fail to allocate image\n");
                  return -1;
         }

    printf("Using %s\n",vpx_codec_iface_name(interface));

    /* Populate encoder configuration */
    ret = vpx_codec_enc_config_default(interface, &cfg, 0);
    if(ret) {
        printf("Failed to get config: %s\n", vpx_codec_err_to_string(ret));
        return -1;
    }

    /* Update the default configuration with our settings */
    cfg.rc_target_bitrate =800;
    cfg.g_w = width;
    cfg.g_h = height;

    write_ivf_file_header(outfile, &cfg, 0);

    /* Initialize codec */
    if(vpx_codec_enc_init(&codec, interface, &cfg, 0)){
        printf("Failed to initialize encoder\n");
                  return -1;
         }

    frame_avail = 1;
    got_data = 0;

         y_size=cfg.g_w*cfg.g_h;

    while(frame_avail || got_data) {
        vpx_codec_iter_t iter = NULL;
        const vpx_codec_cx_pkt_t *pkt;

                  if(fread(raw.planes[0], 1, y_size*3/2, infile)!=y_size*3/2){
                           frame_avail=0;
                  }

                  if(frame_avail){
                           ret=vpx_codec_encode(&codec,&raw,frame_cnt,1,flags,VPX_DL_REALTIME);
                  }else{
                           ret=vpx_codec_encode(&codec,NULL,frame_cnt,1,flags,VPX_DL_REALTIME);
                  }

                  if(ret){
           printf("Failed to encode frame\n");
                           return -1;
```

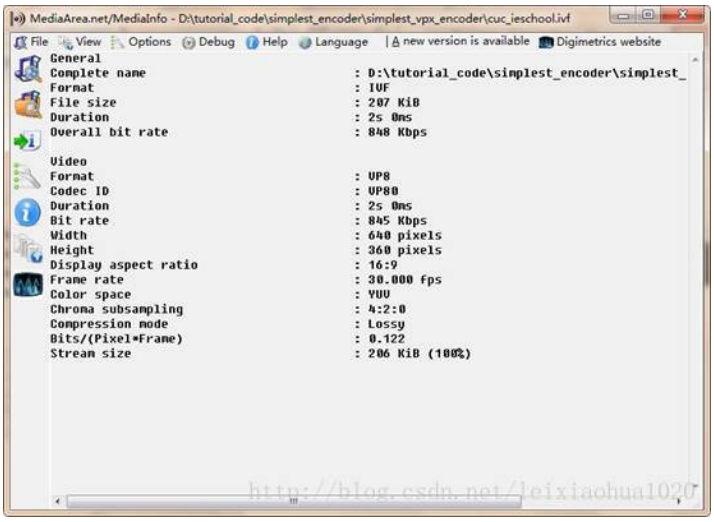
```
                    }
        got_data = 0;
        while( (pkt = vpx_codec_get_cx_data(&codec, &iter)) ) {
            got_data = 1;
            switch(pkt->kind) {
            case VPX_CODEC_CX_FRAME_PKT:
                write_ivf_frame_header(outfile, pkt);
                fwrite(pkt->data.frame.buf, 1, pkt->data.frame.sz,outfile);
                break;
            default:
                break;
            }
        }
                    printf("Succeed encode frame: %5d\n",frame_cnt);
        frame_cnt++;
    }

    fclose(infile);

    vpx_codec_destroy(&codec);

    /* Try to rewrite the file header with the actual frame count */
    if(!fseek(outfile, 0, SEEK_SET))
        write_ivf_file_header(outfile, &cfg, frame_cnt-1);

    fclose(outfile);

    return 0;
}
```

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

文章标签： libvpx vp8 vp9 视频 编解码
个人分类： 我的开源项目 libvpx