原 最简单的基于FFMPEG的转码程序

2014年05月25日 00:01:33　阅读数：65417

本文介绍一个简单的基于FFmpeg的转码器。它可以将一种视频格式（包括封转格式和编码格式）转换为另一种视频格式。转码器在视音频编解码处理的程序中，属于一个比较复杂的东西。因为它结合了视频的解码和编码。一个视频播放器，一般只包含解码功能；一个视频编码工具，一般只包含编码功能；而一个视频转码器，则需要先对视频进行解码，然后再对视频进行编码，因而相当于解码器和编码器的结合。下图例举了一个视频的转码流程。输入视频的封装格式是FLV，视频编码标准是H.264，音频编码标准是AAC；输出视频的封装格式是AVI，视频编码标准是MPEG2，音频编码标准是MP3。从流程中可以看出，首先从输入视频中分离出视频码流和音频压缩码流，然后分别将视频码流和音频码流进行解码，获取到非压缩的像素数据/音频采样数据，接着将非压缩的像素数据/音频采样数据重新进行编码，获得重新编码后的视频码流和音频码流，最后将视频码流和音频码流重新封装成一个文件。

本文介绍的视频转码器正是使用FFMPEG类库从编程的角度实现了上述流程。该例子是从FFmpeg的例子改编的，平台是VC2010，类库版本是2014.5.6。

流程图（2014.9.29更新）

下面附两张使用FFmpeg转码视频的流程图。图中使用浅绿色标出了视频的编码、解码函数。从代码中可以看出，使用了AVFilter的不少东西，因此建议先学习AVFilter的内容后再看这个转码器的源代码。

PS：实际上，转码器不是一定依赖AVFilter的。因此打算有时间对这个转码器进行进一步的简化，使学习的人无需AVFilter的基础也可以理解转码器。

简单介绍一下流程中各个函数的意义：
open_input_file()：打开输入文件，并初始化相关的结构体。
open_output_file()：打开输出文件，并初始化相关的结构体。
init_filters()：初始化AVFilter相关的结构体。
av_read_frame()：从输入文件中读取一个AVPacket。
avcodec_decode_video2()：解码一个视频AVPacket（存储H.264等压缩码流数据）为AVFrame（存储YUV等非压缩的像素数据）。
avcodec_decode_video4()：解码一个音频AVPacket（存储MP3等压缩码流数据）为AVFrame（存储PCM采样数据）。
filter_encode_write_frame()：编码一个AVFrame。
flush_encoder()：输入文件读取完毕后，输出编码器中剩余的AVPacket。

以上函数中open_input_file()，open_output_file()，init_filters()中的函数在其他文章中都有所叙述，在这里不再重复：

open_input_file()可参考： 100行代码实现最简单的基于FFMPEG+SDL的视频播放器（SDL1.x）

open_output_file()可参考： 最简单的基于FFMPEG的视频编码器（YUV编码为H.264）

init_filters()可参考： 最简单的基于FFmpeg的AVfilter例子（水印叠加）

在这里介绍一下其中编码的函数filter_encode_write_frame()。filter_encode_write_frame()函数的流程如下图所示，它完成了视频/音频的编码功能。

PS：视频和音频的编码流程中除了编码函数avcodec_encode_video2()和avcodec_encode_audio2()不一样之外，其他部分几乎完全一样。

简单介绍一下filter_encode_write_frame()中各个函数的意义：

av_buffersrc_add_frame()：将解码后的AVFrame加入Filtergraph。

av_buffersink_get_buffer_ref()：从Filtergraph中取一个AVFrame。

avcodec_encode_video2()：编码一个视频AVFrame为AVPacket。

avcodec_encode_audio2()：编码一个音频AVFrame为AVPacket。

av_interleaved_write_frame()：将编码后的AVPacket写入文件。

代码

贴上代码

```
[cpp]
```

```
/*
 *最简单的基于FFmpeg的转码器
 *Simplest FFmpeg Transcoder
 *
 *雷霄骅 Lei Xiaohua
 *leixiaohua1020@126.com
 *中国传媒大学/数字电视技术
 *Communication University of China / DigitalTV Technology
 *http://blog.csdn.net/leixiaohua1020
 *
 *本程序实现了视频格式之间的转换。是一个最简单的视频转码程序。
 *
 */

#include "stdafx.h"
extern "C"
{
#include "libavcodec/avcodec.h"
#include "libavformat/avformat.h"
#include "libavfilter/avfiltergraph.h"
#include "libavfilter/avcodec.h"
#include "libavfilter/buffersink.h"
#include "libavfilter/buffersrc.h"
#include "libavutil/avutil.h"
#include "libavutil/opt.h"
#include "libavutil/pixdesc.h"
};



static AVFormatContext *ifmt_ctx;
static AVFormatContext *ofmt_ctx;
typedef struct FilteringContext{
    AVFilterContext*buffersink_ctx;
    AVFilterContext*buffersrc_ctx;
    AVFilterGraph*filter_graph;
} FilteringContext;
static FilteringContext *filter_ctx;
static int open_input_file(const char *filename)
{
    int ret;
    unsigned int i;
    ifmt_ctx =NULL;
    if ((ret = avformat_open_input(&ifmt_ctx,filename, NULL, NULL)) < 0) {
       av_log(NULL, AV_LOG_ERROR, "Cannot openinput file\n");
        return ret;
    }
    if ((ret = avformat_find_stream_info(ifmt_ctx, NULL))< 0) {
       av_log(NULL, AV_LOG_ERROR, "Cannot findstream information\n");
        return ret;
    }
    for (i = 0; i < ifmt_ctx->nb_streams; i++) {
        AVStream*stream;
       AVCodecContext *codec_ctx;
        stream =ifmt_ctx->streams[i];
        codec_ctx =stream->codec;
        /* Reencode video & audio and remux subtitles etc. */
        if (codec_ctx->codec_type == AVMEDIA_TYPE_VIDEO
                ||codec_ctx->codec_type == AVMEDIA_TYPE_AUDIO) {
            /* Open decoder */
            ret =avcodec_open2(codec_ctx,
                    avcodec_find_decoder(codec_ctx->codec_id), NULL);
            if (ret < 0) {
               av_log(NULL, AV_LOG_ERROR, "Failed toopen decoder for stream #%u\n", i);
                return ret;
            }
        }
    }
   av_dump_format(ifmt_ctx, 0, filename, 0);
    return 0;
}
static int open_output_file(const char *filename)
{
    AVStream*out_stream;
    AVStream*in_stream;
    AVCodecContext*dec_ctx, *enc_ctx;
    AVCodec*encoder;
    int ret;
    unsigned int i;
    ofmt_ctx =NULL;
   avformat_alloc_output_context2(&ofmt_ctx, NULL, NULL, filename);
    if (!ofmt_ctx) {
       av_log(NULL, AV_LOG_ERROR, "Could notcreate output context\n");
        return AVERROR_UNKNOWN;
    }
    for (i = 0; i < ifmt_ctx->nb_streams; i++) {
        out_stream= avformat_new_stream(ofmt_ctx, NULL);
        if (!out_stream) {
           av_log(NULL, AV_LOG_ERROR, "Failedallocating output stream\n");
            return AVERROR_UNKNOWN;
        }
```

```
        in_stream =ifmt_ctx->streams[i];
        dec_ctx =in_stream->codec;
        enc_ctx =out_stream->codec;
        if (dec_ctx->codec_type == AVMEDIA_TYPE_VIDEO
                ||dec_ctx->codec_type == AVMEDIA_TYPE_AUDIO) {
            /* in this example, we choose transcoding to same codec */
            encoder= avcodec_find_encoder(dec_ctx->codec_id);
            /* In this example, we transcode to same properties(picture size,
            * sample rate etc.). These properties can be changed for output
            * streams easily using filters */
            if (dec_ctx->codec_type == AVMEDIA_TYPE_VIDEO) {
               enc_ctx->height = dec_ctx->height;
               enc_ctx->width = dec_ctx->width;
               enc_ctx->sample_aspect_ratio = dec_ctx->sample_aspect_ratio;
                /* take first format from list of supported formats */
               enc_ctx->pix_fmt = encoder->pix_fmts[0];
                /* video time_base can be set to whatever is handy andsupported by encoder */
               enc_ctx->time_base = dec_ctx->time_base;
            } else {
               enc_ctx->sample_rate = dec_ctx->sample_rate;
               enc_ctx->channel_layout = dec_ctx->channel_layout;
               enc_ctx->channels = av_get_channel_layout_nb_channels(enc_ctx->channel_layout);
                /* take first format from list of supported formats */
               enc_ctx->sample_fmt = encoder->sample_fmts[0];
                AVRationaltime_base={1, enc_ctx->sample_rate};
               enc_ctx->time_base = time_base;
            }
            /* Third parameter can be used to pass settings to encoder*/
            ret =avcodec_open2(enc_ctx, encoder, NULL);
            if (ret < 0) {
               av_log(NULL, AV_LOG_ERROR, "Cannot openvideo encoder for stream #%u\n", i);
                return ret;
            }
        } else if(dec_ctx->codec_type == AVMEDIA_TYPE_UNKNOWN) {
           av_log(NULL, AV_LOG_FATAL, "Elementarystream #%d is of unknown type, cannot proceed\n", i);
            return AVERROR_INVALIDDATA;
        } else {
            /* if this stream must be remuxed */
            ret =avcodec_copy_context(ofmt_ctx->streams[i]->codec,
                   ifmt_ctx->streams[i]->codec);
            if (ret < 0) {
               av_log(NULL, AV_LOG_ERROR, "Copyingstream context failed\n");
                return ret;
            }
        }
        if (ofmt_ctx->oformat->flags &AVFMT_GLOBALHEADER)
           enc_ctx->flags |= CODEC_FLAG_GLOBAL_HEADER;
    }
   av_dump_format(ofmt_ctx, 0, filename, 1);
    if (!(ofmt_ctx->oformat->flags &AVFMT_NOFILE)) {
        ret =avio_open(&ofmt_ctx->pb, filename, AVIO_FLAG_WRITE);
        if (ret < 0) {
           av_log(NULL, AV_LOG_ERROR, "Could notopen output file '%s'", filename);
            return ret;
        }
    }
    /* init muxer, write output file header */
    ret =avformat_write_header(ofmt_ctx, NULL);
    if (ret < 0) {
        av_log(NULL,AV_LOG_ERROR, "Error occurred when openingoutput file\n");
        return ret;
    }
    return 0;
}
static intinit_filter(FilteringContext* fctx, AVCodecContext *dec_ctx,
       AVCodecContext *enc_ctx, const char *filter_spec)
{
    char args[512];
    int ret = 0;
    AVFilter*buffersrc = NULL;
    AVFilter*buffersink = NULL;
    AVFilterContext*buffersrc_ctx = NULL;
    AVFilterContext*buffersink_ctx = NULL;
    AVFilterInOut*outputs = avfilter_inout_alloc();
    AVFilterInOut*inputs  = avfilter_inout_alloc();
    AVFilterGraph*filter_graph = avfilter_graph_alloc();
    if (!outputs || !inputs || !filter_graph) {
        ret =AVERROR(ENOMEM);
        goto end;
    }
    if (dec_ctx->codec_type == AVMEDIA_TYPE_VIDEO) {
        buffersrc =avfilter_get_by_name("buffer");
        buffersink= avfilter_get_by_name("buffersink");
        if (!buffersrc || !buffersink) {
           av_log(NULL, AV_LOG_ERROR, "filteringsource or sink element not found\n");
            ret = AVERROR_UNKNOWN;
            goto end;
        }
       _snprintf(args, sizeof(args),
                "video_size=%dx%d:pix_fmt=%d:time_base=%d/%d:pixel_aspect=%d/%d",
               dec ctx->width, dec ctx->height, dec ctx->pix fmt,
```

```
                dec_ctx->time_base.num,dec_ctx->time_base.den,
                dec_ctx->sample_aspect_ratio.num,
                dec_ctx->sample_aspect_ratio.den);
        ret =avfilter_graph_create_filter(&buffersrc_ctx, buffersrc, "in",
                args, NULL, filter_graph);
        if (ret < 0) {
           av_log(NULL, AV_LOG_ERROR, "Cannotcreate buffer source\n");
            goto end;
        }
        ret =avfilter_graph_create_filter(&buffersink_ctx, buffersink, "out",
               NULL, NULL, filter_graph);
        if (ret < 0) {
           av_log(NULL, AV_LOG_ERROR, "Cannotcreate buffer sink\n");
            goto end;
        }
        ret =av_opt_set_bin(buffersink_ctx, "pix_fmts",
               (uint8_t*)&enc_ctx->pix_fmt, sizeof(enc_ctx->pix_fmt),
               AV_OPT_SEARCH_CHILDREN);
        if (ret < 0) {
           av_log(NULL, AV_LOG_ERROR, "Cannot setoutput pixel format\n");
            goto end;
        }
    } else if(dec_ctx->codec_type == AVMEDIA_TYPE_AUDIO) {
        buffersrc = avfilter_get_by_name("abuffer");
        buffersink= avfilter_get_by_name("abuffersink");
        if (!buffersrc || !buffersink) {
           av_log(NULL, AV_LOG_ERROR, "filteringsource or sink element not found\n");
            ret =AVERROR_UNKNOWN;
            goto end;
        }
        if (!dec_ctx->channel_layout)
           dec_ctx->channel_layout =
               av_get_default_channel_layout(dec_ctx->channels);
       _snprintf(args, sizeof(args),
                "time_base=%d/%d:sample_rate=%d:sample_fmt=%s:channel_layout=0x%I64x",
               dec_ctx->time_base.num, dec_ctx->time_base.den,dec_ctx->sample_rate,
               av_get_sample_fmt_name(dec_ctx->sample_fmt),
               dec_ctx->channel_layout);
        ret =avfilter_graph_create_filter(&buffersrc_ctx, buffersrc, "in",
               args, NULL, filter_graph);
        if (ret < 0) {
           av_log(NULL, AV_LOG_ERROR, "Cannotcreate audio buffer source\n");
            goto end;
        }
        ret =avfilter_graph_create_filter(&buffersink_ctx, buffersink, "out",
               NULL, NULL, filter_graph);
        if (ret < 0) {
           av_log(NULL, AV_LOG_ERROR, "Cannotcreate audio buffer sink\n");
            goto end;
        }
        ret = av_opt_set_bin(buffersink_ctx, "sample_fmts",
               (uint8_t*)&enc_ctx->sample_fmt, sizeof(enc_ctx->sample_fmt),
               AV_OPT_SEARCH_CHILDREN);
        if (ret < 0) {
           av_log(NULL, AV_LOG_ERROR, "Cannot setoutput sample format\n");
            goto end;
        }
        ret =av_opt_set_bin(buffersink_ctx, "channel_layouts",
               (uint8_t*)&enc_ctx->channel_layout,
                sizeof(enc_ctx->channel_layout),AV_OPT_SEARCH_CHILDREN);
        if (ret < 0) {
           av_log(NULL, AV_LOG_ERROR, "Cannot setoutput channel layout\n");
            goto end;
        }
        ret =av_opt_set_bin(buffersink_ctx, "sample_rates",
               (uint8_t*)&enc_ctx->sample_rate, sizeof(enc_ctx->sample_rate),
               AV_OPT_SEARCH_CHILDREN);
        if (ret < 0) {
           av_log(NULL, AV_LOG_ERROR, "Cannot setoutput sample rate\n");
            goto end;
        }
    } else {
        ret =AVERROR_UNKNOWN;
        goto end;
    }
    /* Endpoints for the filter graph. */
   outputs->name       =av_strdup("in");
   outputs->filter_ctx = buffersrc_ctx;
   outputs->pad_idx    = 0;
   outputs->next       = NULL;
   inputs->name       = av_strdup("out");
   inputs->filter_ctx = buffersink_ctx;
   inputs->pad_idx    = 0;
   inputs->next       = NULL;
    if (!outputs->name || !inputs->name) {
        ret =AVERROR(ENOMEM);
        goto end;
    }
    if ((ret = avfilter_graph_parse_ptr(filter_graph,filter_spec,
                &inputs, &outputs, NULL)) < 0)
        goto end;
```

```
    if ((ret = avfilter_graph_config(filter_graph, NULL))< 0)
        goto end;
    /* Fill FilteringContext */
   fctx->buffersrc_ctx = buffersrc_ctx;
   fctx->buffersink_ctx = buffersink_ctx;
    fctx->filter_graph= filter_graph;
end:
   avfilter_inout_free(&inputs);
   avfilter_inout_free(&outputs);
    return ret;
}
static int init_filters(void)
{
    const char*filter_spec;
    unsigned int i;
    int ret;
    filter_ctx =(FilteringContext *)av_malloc_array(ifmt_ctx->nb_streams, sizeof(*filter_ctx));
    if (!filter_ctx)
        return AVERROR(ENOMEM);
    for (i = 0; i < ifmt_ctx->nb_streams; i++) {
       filter_ctx[i].buffersrc_ctx  =NULL;
        filter_ctx[i].buffersink_ctx= NULL;
       filter_ctx[i].filter_graph   =NULL;
        if(!(ifmt_ctx->streams[i]->codec->codec_type == AVMEDIA_TYPE_AUDIO
                ||ifmt_ctx->streams[i]->codec->codec_type == AVMEDIA_TYPE_VIDEO))
            continue;
        if (ifmt_ctx->streams[i]->codec->codec_type== AVMEDIA_TYPE_VIDEO)
           filter_spec = "null"; /* passthrough (dummy) filter for video */
        else
           filter_spec = "anull"; /* passthrough (dummy) filter for audio */
        ret =init_filter(&filter_ctx[i], ifmt_ctx->streams[i]->codec,
             ofmt_ctx->streams[i]->codec, filter_spec);
        if (ret)
            return ret;
    }
    return 0;
}
static intencode_write_frame(AVFrame *filt_frame, unsignedint stream_index, int*got_frame) {
    int ret;
    int got_frame_local;
    AVPacketenc_pkt;
    int (*enc_func)(AVCodecContext *, AVPacket *, const AVFrame *, int*) =
       (ifmt_ctx->streams[stream_index]->codec->codec_type ==
        AVMEDIA_TYPE_VIDEO) ? avcodec_encode_video2 : avcodec_encode_audio2;
    if (!got_frame)
        got_frame =&got_frame_local;
    av_log(NULL,AV_LOG_INFO, "Encoding frame\n");
    /* encode filtered frame */
    enc_pkt.data =NULL;
    enc_pkt.size =0;
    av_init_packet(&enc_pkt);
    ret =enc_func(ofmt_ctx->streams[stream_index]->codec, &enc_pkt,
           filt_frame, got_frame);
   av_frame_free(&filt_frame);
    if (ret < 0)
        return ret;
    if (!(*got_frame))
        return 0;
    /* prepare packet for muxing */
   enc_pkt.stream_index = stream_index;
    enc_pkt.dts =av_rescale_q_rnd(enc_pkt.dts,
           ofmt_ctx->streams[stream_index]->codec->time_base,
           ofmt_ctx->streams[stream_index]->time_base,
           (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
    enc_pkt.pts =av_rescale_q_rnd(enc_pkt.pts,
           ofmt_ctx->streams[stream_index]->codec->time_base,
           ofmt_ctx->streams[stream_index]->time_base,
           (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
   enc_pkt.duration = av_rescale_q(enc_pkt.duration,
           ofmt_ctx->streams[stream_index]->codec->time_base,
           ofmt_ctx->streams[stream_index]->time_base);
    av_log(NULL,AV_LOG_DEBUG, "Muxing frame\n");
    /* mux encoded frame */
    ret =av_interleaved_write_frame(ofmt_ctx, &enc_pkt);
    return ret;
}
static intfilter_encode_write_frame(AVFrame *frame, unsignedint stream_index)
{
    int ret;
    AVFrame*filt_frame;
    av_log(NULL,AV_LOG_INFO, "Pushing decoded frame tofilters\n");
    /* push the decoded frame into the filtergraph */
    ret =av_buffersrc_add_frame_flags(filter_ctx[stream_index].buffersrc_ctx,
            frame,0);
    if (ret < 0) {
       av_log(NULL, AV_LOG_ERROR, "Error whilefeeding the filtergraph\n");
        return ret;
    }
    /* pull filtered frames from the filtergraph */
    while (1) {
        filt_frame= av_frame_alloc();
```

```
        if (!filt_frame) {
            ret =AVERROR(ENOMEM);
            break;
        }
       av_log(NULL, AV_LOG_INFO, "Pullingfiltered frame from filters\n");
        ret =av_buffersink_get_frame(filter_ctx[stream_index].buffersink_ctx,
               filt_frame);
        if (ret < 0) {
            /* if nomore frames for output - returns AVERROR(EAGAIN)
            * if flushed and no more frames for output - returns AVERROR_EOF
            * rewrite retcode to 0 to show it as normal procedure completion
            */
            if (ret == AVERROR(EAGAIN) || ret == AVERROR_EOF)
                ret= 0;
           av_frame_free(&filt_frame);
            break;
        }
       filt_frame->pict_type = AV_PICTURE_TYPE_NONE;
        ret =encode_write_frame(filt_frame, stream_index, NULL);
        if (ret < 0)
            break;
    }
    return ret;
}
static int flush_encoder(unsigned intstream_index)
{
    int ret;
    int got_frame;
    if(!(ofmt_ctx->streams[stream_index]->codec->codec->capabilities&
                CODEC_CAP_DELAY))
        return 0;
    while (1) {
       av_log(NULL, AV_LOG_INFO, "Flushingstream #%u encoder\n", stream_index);
        ret =encode_write_frame(NULL, stream_index, &got_frame);
        if (ret < 0)
            break;
        if (!got_frame)
            return 0;
    }
    return ret;
}

int_tmain(int argc, _TCHAR* argv[])
{
    int ret;
    AVPacketpacket;
    AVFrame *frame= NULL;
    enum AVMediaType type;
    unsigned intstream_index;
    unsigned int i;
    int got_frame;
    int (*dec_func)(AVCodecContext *, AVFrame *, int *, const AVPacket*);
    if (argc != 3) {
       av_log(NULL, AV_LOG_ERROR, "Usage: %s<input file> <output file>\n", argv[0]);
        return 1;
    }
   av_register_all();
   avfilter_register_all();
    if ((ret = open_input_file(argv[1])) < 0)
        goto end;
    if ((ret = open_output_file(argv[2])) < 0)
        goto end;
    if ((ret = init_filters()) < 0)
        goto end;
    /* read all packets */
    while (1) {
        if ((ret= av_read_frame(ifmt_ctx, &packet)) < 0)
            break;
       stream_index = packet.stream_index;
        type =ifmt_ctx->streams[packet.stream_index]->codec->codec_type;
       av_log(NULL, AV_LOG_DEBUG, "Demuxergave frame of stream_index %u\n",
               stream_index);
        if (filter_ctx[stream_index].filter_graph) {
           av_log(NULL, AV_LOG_DEBUG, "Going toreencode&filter the frame\n");
            frame =av_frame_alloc();
            if (!frame) {
                ret = AVERROR(ENOMEM);
                break;
            }
            packet.dts = av_rescale_q_rnd(packet.dts,
                    ifmt_ctx->streams[stream_index]->time_base,
                    ifmt_ctx->streams[stream_index]->codec->time_base,
                     (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
            packet.pts = av_rescale_q_rnd(packet.pts,
                    ifmt_ctx->streams[stream_index]->time_base,
                    ifmt_ctx->streams[stream_index]->codec->time_base,
                    (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
            dec_func = (type == AVMEDIA_TYPE_VIDEO) ? avcodec_decode_video2 :
                avcodec_decode_audio4;
            ret =dec_func(ifmt_ctx->streams[stream_index]->codec, frame,
                    &got_frame, &packet);
```

```
            if (ret < 0) {
               av_frame_free(&frame);
               av_log(NULL, AV_LOG_ERROR, "Decodingfailed\n");
                break;
            }
            if (got_frame) {
               frame->pts = av_frame_get_best_effort_timestamp(frame);
                ret= filter_encode_write_frame(frame, stream_index);
               av_frame_free(&frame);
                if (ret< 0)
                   goto end;
            } else {
               av_frame_free(&frame);
            }
        } else {
            /* remux this frame without reencoding */
           packet.dts = av_rescale_q_rnd(packet.dts,
                   ifmt_ctx->streams[stream_index]->time_base,
                   ofmt_ctx->streams[stream_index]->time_base,
                    (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
           packet.pts = av_rescale_q_rnd(packet.pts,
                   ifmt_ctx->streams[stream_index]->time_base,
                   ofmt_ctx->streams[stream_index]->time_base,
                    (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
            ret =av_interleaved_write_frame(ofmt_ctx, &packet);
            if (ret < 0)
                goto end;
        }
       av_free_packet(&packet);
    }
    /* flush filters and encoders */
    for (i = 0; i < ifmt_ctx->nb_streams; i++) {
        /* flush filter */
        if (!filter_ctx[i].filter_graph)
            continue;
        ret =filter_encode_write_frame(NULL, i);
        if (ret < 0) {
           av_log(NULL, AV_LOG_ERROR, "Flushingfilter failed\n");
            goto end;
        }
        /* flush encoder */
        ret = flush_encoder(i);
        if (ret < 0) {
           av_log(NULL, AV_LOG_ERROR, "Flushingencoder failed\n");
            goto end;
        }
    }
   av_write_trailer(ofmt_ctx);
end:
   av_free_packet(&packet);
   av_frame_free(&frame);
    for (i = 0; i < ifmt_ctx->nb_streams; i++) {
       avcodec_close(ifmt_ctx->streams[i]->codec);
        if (ofmt_ctx && ofmt_ctx->nb_streams >i && ofmt_ctx->streams[i] &&ofmt_ctx->streams[i]->codec)
           avcodec_close(ofmt_ctx->streams[i]->codec);
        if(filter_ctx && filter_ctx[i].filter_graph)
           avfilter_graph_free(&filter_ctx[i].filter_graph);
    }
   av_free(filter_ctx);
   avformat_close_input(&ifmt_ctx);
    if (ofmt_ctx &&!(ofmt_ctx->oformat->flags & AVFMT_NOFILE))
        avio_close(ofmt_ctx->pb);
   avformat_free_context(ofmt_ctx);
    if (ret < 0)
       av_log(NULL, AV_LOG_ERROR, "Erroroccurred\n");
    return (ret? 1:0);
}
```

程序运行截图：

默认情况下运行程序，会将“cuc_ieschool.ts”转换为“cuc_ieschool.avi”。调试的时候，可以修改“配置属性->调试->命令参数”中的参数，即可改变转码的输入输出文件。

工程下载地址（VC2010）：http://download.csdn.net/detail/leixiaohua1020/7394649

文章标签： ffmpeg 视频 音频 转码 VC

个人分类： FFMPEG

所属专栏： FFmpeg