原 ffdshow 源代码分析 7： libavcodec视频解码器类（TvideoCodecLibavcodec）

2013年11月12日 00:36:32 阅读数：6552

==========

ffdshow源代码分析系列文章列表：

==========

前文已经介绍了ffdshow中对libavcodec封装的类Tlibavcodec：

ffdshow 源代码分析 6： 对解码器的dll的封装（libavcodec）

在这里我们进一步介绍一下其libavcodec解码器类。注意前一篇文章介绍的类Tlibavcodec仅仅是对libavcodec所在的“ffmpeg.dll”的函数进行封装的类。但Tlibavcodec并不是一个解码器类，其没有继承任何类，还不能为ffdshow所用。本文介绍的TvideoCodecLibavcodec才是libavcodec解码器类，其继承了TvideoCodecDec。

先来看一看TvideoCodecLibavcodec的定义吧，位于codecs-> TvideoCodecLibavcodec.h中。

```cpp
/*
 *雷霄骅
 *leixiaohua1020@126.com
 *中国传媒大学/数字电视技术
 */
#ifndef _TVIDEOCODECLIBAVCODEC_H_
#define _TVIDEOCODECLIBAVCODEC_H_

#include "TvideoCodec.h"
#include "ffmpeg/Tlibavcodec.h"
#include "ffmpeg/libavcodec/avcodec.h"

#define MAX_THREADS 8 // FIXME: This is defined in mpegvideo.h.

struct Textradata;
class TccDecoder;
//libavcodec解码器（视频）
struct TlibavcodecExt {
private:
    static int get_buffer(AVCodecContext *s, AVFrame *pic);
    int (*default_get_buffer)(AVCodecContext *s, AVFrame *pic);
    static void release_buffer(AVCodecContext *s, AVFrame *pic);
    void (*default_release_buffer)(AVCodecContext *s, AVFrame *pic);
    static int reget_buffer(AVCodecContext *s, AVFrame *pic);
    int (*default_reget_buffer)(AVCodecContext *s, AVFrame *pic);
    static void handle_user_data0(AVCodecContext *c, const uint8_t *buf, int buf_len);
public:
    virtual ~TlibavcodecExt() {}
    void connectTo(AVCodecContext *ctx, Tlibavcodec *libavcodec);
    virtual void onGetBuffer(AVFrame *pic) {}
    virtual void onRegetBuffer(AVFrame *pic) {}
    virtual void onReleaseBuffer(AVFrame *pic) {}
    virtual void handle_user_data(const uint8_t *buf, int buf_len) {}
```

```
};
//libavcodec解码，不算是Filter？
class TvideoCodecLibavcodec : public TvideoCodecDec, public TvideoCodecEnc, public TlibavcodecExt
{
    friend class TDXVADecoderVC1;
    friend class TDXVADecoderH264;
protected:
    //各种信息（源自AVCodecContext中）
    Tlibavcodec *libavcodec;
    void create(void);
    AVCodec *avcodec;
    mutable char_t codecName[100];
    AVCodecContext *avctx;
    uint32_t palette[AVPALETTE_COUNT];
    int palette_size;
    AVFrame *frame;
    FOURCC fcc;
    FILE *statsfile;
    int cfgcomode;
    int psnr;
    bool isAdaptive;
    int threadcount;
    bool dont_use_rtStop_from_upper_stream; // and reordering of timpestams is justified.
    bool theorart;
    bool codecinited, ownmatrices;
    REFERENCE_TIME rtStart, rtStop, avgTimePerFrame, segmentTimeStart;
    REFERENCE_TIME prior_in_rtStart, prior_in_rtStop;
    REFERENCE_TIME prior_out_rtStart, prior_out_rtStop;

    struct {
        REFERENCE_TIME rtStart, rtStop;
        unsigned int srcSize;
    } b[MAX_THREADS + 1];
    int inPosB;

    Textradata *extradata;
    bool sendextradata;
    unsigned int mb_width, mb_height, mb_count;
    static void line(unsigned char *dst, unsigned int _x0, unsigned int _y0, unsigned int _x1, unsigned int _y1, stride_t strideY);
    static void draw_arrow(uint8_t *buf, int sx, int sy, int ex, int ey, stride_t stride, int mulx, int muly, int dstdx, int dstdy);

    unsigned char *ffbuf;
    unsigned int ffbuflen;
    bool wasKey;
    virtual void handle_user_data(const uint8_t *buf, int buf_len);
    TccDecoder *ccDecoder;
    bool autoSkipingLoopFilter;
    enum AVDiscard initialSkipLoopFilter;
    int got_picture;
    bool firstSeek; // firstSeek means start of palyback.
    bool mpeg2_in_doubt;
    bool mpeg2_new_sequence;
    bool bReorderBFrame;
    //时长（AVCodecContext中）
    REFERENCE_TIME getDuration();
    int isReallyMPEG2(const unsigned char *src, size_t srcLen);
protected:
    virtual LRESULT beginCompress(int cfgcomode, uint64_t csp, const Trect &r);
    virtual bool beginDecompress(TffPictBase &pict, FOURCC infcc, const CMediaType &mt, int sourceFlags);
    virtual HRESULT flushDec(void);
    AVCodecParserContext *parser;
public:
    TvideoCodecLibavcodec(IffdshowBase *Ideci, IdecVideoSink *IsinkD);
    TvideoCodecLibavcodec(IffdshowBase *Ideci, IencVideoSink *IsinkE);
    virtual ~TvideoCodecLibavcodec();
    virtual int getType(void) const {
        return IDFF_MOVIE_LAVC;
    }
    virtual const char_t* getName(void) const;
    virtual int caps(void) const {
        return CAPS::VIS_MV | CAPS::VIS_QUANTS;
    }

    virtual void end(void);

    virtual void getCompressColorspaces(Tcsps &csps, unsigned int outDx, unsigned int outDy);
    virtual bool supExtradata(void);
    //获得ExtraData（AVCodecContext中）
    virtual bool getExtradata(const void* *ptr, size_t *len);
    virtual HRESULT compress(const TffPict &pict, TencFrameParams ¶ms);
    virtual HRESULT flushEnc(const TffPict &pict, TencFrameParams ¶ms) {
        return compress(pict, params);
    }

    virtual HRESULT decompress(const unsigned char *src, size_t srcLen, IMediaSample *pIn);
    virtual void onGetBuffer(AVFrame *pic);
    virtual bool onSeek(REFERENCE_TIME segmentStart);
    virtual bool onDiscontinuity(void);
    //画出运动矢量（AVCodecContext中）
    virtual bool drawMV(unsigned char *dst, unsigned int dx, stride_t stride, unsigned int dy) const;
    //编码器信息（AVCodecContext中）
```

```
    virtual void getEncoderInfo(char_t *buf, size_t buflen) const;
    virtual const char* get_current_idct(void);
    virtual HRESULT BeginFlush();
    bool isReorderBFrame() {
        return bReorderBFrame;
    };
    virtual void reorderBFrames(REFERENCE_TIME& rtStart, REFERENCE_TIME& rtStop);

    class Th264RandomAccess
    {
        friend class TvideoCodecLibavcodec;
    private:
        TvideoCodecLibavcodec* parent;
        int recovery_mode;  // 0:OK, 1:searching 2: found, 3:waiting for frame_num decoded, 4:waiting for POC outputed
        int recovery_frame_cnt;
        int recovery_poc;
        int thread_delay;

    public:
        Th264RandomAccess(TvideoCodecLibavcodec* Iparent);
        int search(uint8_t* buf, int buf_size);
        void onSeek(void);
        void judgeUsability(int *got_picture_ptr);
    } h264RandomAccess;
};

#endif
```

这里有一个类TlibavcodecExt，我觉得应该是扩展了Tlibavcodec的一些功能，在这里我们先不管它，直接看看TvideoCodecLibavcodec都包含了什么变量：

Tlibavcodec *libavcodec：该类封装了libavcodec的各种函数，在前一篇文章中已经做过介绍，在此不再重复叙述了。可以认为该变量是TvideoCodecLibavcodec类的灵魂，所有libavcodec中的函数都是通过该类调用的。

AVCodec *avcodec：FFMPEG中的结构体，解码器
AVCodecContext *avctx：FFMPEG中的结构体，解码器上下文
AVFrame *frame FFMPEG中的结构体，视频帧
mutable char_t codecName[100]：解码器名称
FOURCC fcc：FourCC
Textradata *extradata：附加数据

…

再来看一下TvideoCodecLibavcodec都包含什么方法：

create()：创建解码器的时候调用
getDuration()：获得时长
getExtradata()：获得附加数据
drawMV()：画运动矢量
getEncoderInfo()：获得编码器信息
此外还包括一些有关解码的方法【这个是最关键的】：
beginDecompress()：解码初始化

decompress()：解码

下面我们来详细看看这些函数的实现吧：

先来看一下TvideoCodecLibavcodec的构造函数：

```cpp
//libavcodec解码器（视频）
//内容大部分都很熟悉，因为是FFmpeg的API
TvideoCodecLibavcodec::TvideoCodecLibavcodec(IffdshowBase *Ideci, IdecVideoSink *IsinkD):
    Tcodec(Ideci), TcodecDec(Ideci, IsinkD),
    TvideoCodec(Ideci),
    TvideoCodecDec(Ideci, IsinkD),
    TvideoCodecEnc(Ideci, NULL),
    h264RandomAccess(this),
    bReorderBFrame(true)
{
    create();
}
```

可见构造函数调用了Create()，我们再来看看Create()：

```cpp
void TvideoCodecLibavcodec::create(void)
{
    ownmatrices = false;
    deci->getLibavcodec(&libavcodec);
    ok = libavcodec ? libavcodec->ok : false;
    avctx = NULL;
    avcodec = NULL;
    frame = NULL;
    quantBytes = 1;
    statsfile = NULL;
    threadcount = 0;
    codecinited = false;
    extradata = NULL;
    theorart = false;
    ffbuf = NULL;
    ffbuflen = 0;
    codecName[0] = '\0';
    ccDecoder = NULL;
    autoSkipingLoopFilter = false;
    inPosB = 1;
    firstSeek = true;
    mpeg2_new_sequence = true;
    parser = NULL;
}
```

从Create()函数我们可以看出，其完成了各种变量的初始化工作。其中有一行代码：

```cpp
deci->getLibavcodec(&libavcodec);
```

完成了Tlibavcodec*libavcodec的初始化工作。

再来看几个函数。

getDuration()，用于从AVCodecContext中获取时长：

```cpp
REFERENCE_TIME TvideoCodecLibavcodec::getDuration()
{
    REFERENCE_TIME duration = REF_SECOND_MULT / 100;
    if (avctx && avctx->time_base.num && avctx->time_base.den) {
        duration = REF_SECOND_MULT * avctx->time_base.num / avctx->time_base.den;
        if (codecId == AV_CODEC_ID_H264) {
            duration *= 2;
        }
    }
    if (duration == 0) {
        return REF_SECOND_MULT / 100;
    }
    return duration;
}
```

getExtradata()用于从AVCodecContext中获取附加信息：

```cpp
bool TvideoCodecLibavcodec::getExtradata(const void* *ptr, size_t *len)
{
    if (!avctx || !len) {
        return false;
    }
    *len = avctx->extradata_size;
    if (ptr) {
        *ptr = avctx->extradata;
    }
    return true;
}
```

drawMV()用于从AVFrame中获取运动矢量信息，并画出来（这个函数用于一个名为“可视化”的滤镜里面，用于显示视频的运动矢量信息）。

```cpp
//画出运动矢量
bool TvideoCodecLibavcodec::drawMV(unsigned char *dst, unsigned int dstdx, stride_t stride, unsigned int dstdy) const
{
    if (!frame->motion_val || !frame->mb_type || !frame->motion_val[0]) {
        return false;
    }

#define IS_8X8(a)  ((a)&MB_TYPE_8x8)
#define IS_16X8(a) ((a)&MB_TYPE_16x8)
#define IS_8X16(a) ((a)&MB_TYPE_8x16)
#define IS_INTERLACED(a) ((a)&MB_TYPE_INTERLACED)
#define USES_LIST(a, list) ((a) & ((MB_TYPE_P0L0|MB_TYPE_P1L0)<<(2*(list))))

    const int shift = 1 + ((frame->play_flags & CODEC_FLAG_QPEL) ? 1 : 0);
    const int mv_sample_log2 = 4 - frame->motion_subsample_log2;
    const int mv_stride = (frame->mb_width << mv_sample_log2) + (avctx->codec_id == AV_CODEC_ID_H264 ? 0 : 1);
    int direction = 0;

    int mulx = (dstdx << 12) / avctx->width;
    int muly = (dstdy << 12) / avctx->height;
    //提取两个方向上的运动矢量信息（根据不同的宏块划分，可以分成几种情况）
    //在AVCodecContext的motion_val中
    for (int mb_y = 0; mb_y < frame->mb_height; mb_y++)
        for (int mb_x = 0; mb_x < frame->mb_width; mb_x++) {
            const int mb_index = mb_x + mb_y * frame->mb_stride;
            if (!USES_LIST(frame->mb_type[mb_index], direction)) {
                continue;
            }
            …此处代码太长，略
        }
#undef IS_8X8
#undef IS_16X8
#undef IS_8X16
#undef IS_INTERLACED
#undef USES_LIST
    return true;
}
```

下面来看几个很重要的函数，这几个函数继承自TvideoCodecDec类。

beginDecompress()用于解码器的初始化 。注：这个函数的代码太长了，因此只选择一点关键的代码。

```cpp
//---------------------------- decompression ---------------------------
bool TvideoCodecLibavcodec::beginDecompress(TffPictBase &pict, FOURCC fcc, const CMediaType &mt, int sourceFlags)
{
    palette_size = 0;
    prior_out_rtStart = REFTIME_INVALID;
    prior_out_rtStop = 0;
    rtStart = rtStop = REFTIME_INVALID;
    prior_in_rtStart = prior_in_rtStop = REFTIME_INVALID;
    mpeg2_in_doubt = codecId == AV_CODEC_ID_MPEG2VIDEO;

    int using_dxva = 0;

    int numthreads = deci->getParam2(IDFF_numLAVCdecThreads);
    int thread_type = 0;
    if (numthreads > 1 && sup_threads_dec_frame(codecId)) {
        thread_type = FF_THREAD_FRAME;
    } else if (numthreads > 1 && sup_threads_dec_slice(codecId)) {
        thread_type = FF_THREAD_SLICE;
    }

    if (numthreads > 1 && thread_type != 0) {
        threadcount = numthreads;
    } else {
        threadcount = 1;
    }

    if (codecId == CODEC_ID_H264_DXVA) {
        codecId = AV_CODEC_ID_H264;
        using_dxva = 1;
    } else if (codecId == CODEC_ID_VC1_DXVA) {
        codecId = AV_CODEC_ID_VC1;
        using_dxva = 1;
    }

    avcodec = libavcodec->avcodec_find_decoder(codecId);
    if (!avcodec) {
        return false;
    }
    avctx = libavcodec->avcodec_alloc_context(avcodec, this);
    avctx->thread_type = thread_type;
    avctx->thread_count = threadcount;
    avctx->h264_using_dxva = using_dxva;
    if (codecId == AV_CODEC_ID_H264) {
        // If we do not set this, first B-frames before the IDR pictures are dropped.
        avctx->has_b_frames = 1;
    }

    frame = libavcodec->avcodec_alloc_frame();
    avctx->width = pict.rectFull.dx;
    avctx->height = pict.rectFull.dy;
    intra_matrix = avctx->intra_matrix = (uint16_t*)calloc(sizeof(uint16_t), 64);
    inter_matrix = avctx->inter_matrix = (uint16_t*)calloc(sizeof(uint16_t), 64);
    ownmatrices = true;


    // Fix for new Haali custom media type and fourcc. ffmpeg does not understand it, we have to change it to FOURCC_AVC1
    if (fcc == FOURCC_CCV1) {
        fcc = FOURCC_AVC1;
    }

    avctx->codec_tag = fcc;
    avctx->workaround_bugs = deci->getParam2(IDFF_workaroundBugs);
#if 0
    avctx->error_concealment = FF_EC_GUESS_MVS | FF_EC_DEBLOCK;
    avctx->err_recognition   = AV_EF_CRCCHECK | AV_EF_BITSTREAM | AV_EF_BUFFER | AV_EF_COMPLIANT | AV_EF_AGGRESSIVE;
#endif
    if (codecId == AV_CODEC_ID_MJPEG) {
        avctx->flags |= CODEC_FLAG_TRUNCATED;
    }
    if (mpeg12_codec(codecId) && deci->getParam2(IDFF_fastMpeg2)) {
        avctx->flags2 = CODEC_FLAG2_FAST;
    }
    if (codecId == AV_CODEC_ID_H264)
        if (int skip = deci->getParam2(IDFF_fastH264)) {
            avctx->skip_loop_filter = skip & 2 ? AVDISCARD_ALL : AVDISCARD_NONREF;
        }
    initialSkipLoopFilter = avctx->skip_loop_filter;

    avctx->debug_mv = !using_dxva; //(deci->getParam2(IDFF_isVis) & deci->getParam2(IDFF_visMV));

    avctx->idct_algo = limit(deci->getParam2(IDFF_idct), 0, 6);
    if (extradata) {
        delete extradata;
    }
extradata = new Textradata(mt, FF_INPUT_BUFFER_PADDING_SIZE);
此处代码太长，略…
}
```

从代码中可以看出这个函数的流程是：

1.avcodec_find_decoder();
2.avcodec_alloc_context();
3.avcodec_alloc_frame();
4.avcodec_open();

主要做了libavcodec初始化工作。

begin decompress()用于解码器的初始化。 注：这个函数的代码太长了，因此只选择一点关键的代码。

```cpp
HRESULT TvideoCodecLibavcodec::decompress(const unsigned char *src, size_t srcLen0, IMediaSample *pIn)
{
    代码太长，略…
    AVPacket avpkt;
    libavcodec->av_init_packet(&avpkt);
    if (palette_size) {
        uint32_t *pal = (uint32_t *)libavcodec->av_packet_new_side_data(&avpkt, AV_PKT_DATA_PALETTE, AVPALETTE_SIZE);
        for (int i = 0; i < palette_size / 4; i++) {
            pal[i] = 0xFF << 24 | AV_RL32(palette + i);
        }
    }

    while (!src || size > 0) {
        int used_bytes;

        avctx->reordered_opaque = rtStart;
        avctx->reordered_opaque2 = rtStop;
        avctx->reordered_opaque3 = size;

        if (sendextradata && extradata->data && extradata->size > 0) {
            avpkt.data = (uint8_t *)extradata->data;
            avpkt.size = (int)extradata->size;
            used_bytes = libavcodec->avcodec_decode_video2(avctx, frame, &got_picture, &avpkt);
            sendextradata = false;
            if (used_bytes > 0) {
                used_bytes = 0;
            }
            if (mpeg12_codec(codecId)) {
                avctx->extradata = NULL;
                avctx->extradata_size = 0;
            }
        } else {
            unsigned int neededsize = size + FF_INPUT_BUFFER_PADDING_SIZE;

            if (ffbuflen < neededsize) {
                ffbuf = (unsigned char*)realloc(ffbuf, ffbuflen = neededsize);
            }

            if (src) {
                memcpy(ffbuf, src, size);
                memset(ffbuf + size, 0, FF_INPUT_BUFFER_PADDING_SIZE);
            }
            if (parser) {
                uint8_t *outBuf = NULL;
                int out_size = 0;
                used_bytes = libavcodec-
>av_parser_parse2(parser, avctx, &outBuf, &out_size, src ? ffbuf : NULL, size, AV_NOPTS_VALUE, AV_NOPTS_VALUE, 0);
                if (prior_in_rtStart == REFTIME_INVALID) {
                    prior_in_rtStart = rtStart;
                    prior_in_rtStop = rtStop;
                }
                if (out_size > 0 || !src) {
                    mpeg2_in_doubt = false;
                    avpkt.data = out_size > 0 ? outBuf : NULL;
                    avpkt.size = out_size;
                    if (out_size > used_bytes) {
                        avctx->reordered_opaque = prior_in_rtStart;
                        avctx->reordered_opaque2 = prior_in_rtStop;
                    } else {
                        avctx->reordered_opaque = rtStart;
                        avctx->reordered_opaque2 = rtStop;
                    }
                    prior_in_rtStart = rtStart;
                    prior_in_rtStop = rtStop;
                    avctx->reordered_opaque3 = out_size;
                    if (h264RandomAccess.search(avpkt.data, avpkt.size)) {
                        libavcodec->avcodec_decode_video2(avctx, frame, &got_picture, &avpkt);
                        h264RandomAccess.judgeUsability(&got_picture);
                    } else {
                        got_picture = 0;
                    }
                } else {
                    got_picture = 0;
                }
            } else {
                avpkt.data = src ? ffbuf : NULL;
```

```
                avpkt.size = size;
                if (codecId == AV_CODEC_ID_H264) {
                    if (h264RandomAccess.search(avpkt.data, avpkt.size)) {
                        used_bytes = libavcodec->avcodec_decode_video2(avctx, frame, &got_picture, &avpkt);
                        if (used_bytes < 0) {
                            return S_OK;
                        }
                        h264RandomAccess.judgeUsability(&got_picture);
                    } else {
                        got_picture = 0;
                        return S_OK;
                    }
                } else {
                    used_bytes = libavcodec->avcodec_decode_video2(avctx, frame, &got_picture, &avpkt);
                }
            }
        }
    代码太长，略…
}
```

从代码中可以看出这个函数的流程是：

1.AVPacket avpkt;
2.av_init_packet();
3.avcodec_decode_video2();

和ffmpeg的解码流程相差不大。

文章标签： ffdshow libavcodec 解码器 源代码 ffmpeg

个人分类： ffdshow

所属专栏： 开源多媒体项目源代码分析