原 FFmpeg源代码简单分析：libavdevice的gdigrab

2015年03月25日 12:33:42　阅读数：10027

==========

FFmpeg的库函数源代码分析文章列表：

【架构图】

FFmpeg源代码结构图 - 解码

FFmpeg源代码结构图 - 编码

【通用】

FFmpeg 源代码简单分析：av_register_all()

FFmpeg 源代码简单分析：avcodec_register_all()

FFmpeg 源代码简单分析：内存的分配和释放（av_malloc()、av_free()等）

FFmpeg 源代码简单分析：常见结构体的初始化和销毁（AVFormatContext，AVFrame等）

FFmpeg 源代码简单分析：avio_open2()

FFmpeg 源代码简单分析：av_find_decoder()和av_find_encoder()

FFmpeg 源代码简单分析：avcodec_open2()

FFmpeg 源代码简单分析：avcodec_close()

【解码】

图解FFMPEG打开媒体的函数avformat_open_input

FFmpeg 源代码简单分析：avformat_open_input()

FFmpeg 源代码简单分析：avformat_find_stream_info()

FFmpeg 源代码简单分析：av_read_frame()

FFmpeg 源代码简单分析：avcodec_decode_video2()

FFmpeg 源代码简单分析：avformat_close_input()

【编码】

FFmpeg 源代码简单分析：avformat_alloc_output_context2()

FFmpeg 源代码简单分析：avformat_write_header()

FFmpeg 源代码简单分析：avcodec_encode_video()

FFmpeg 源代码简单分析：av_write_frame()

FFmpeg 源代码简单分析：av_write_trailer()

【其它】

FFmpeg源代码简单分析：日志输出系统（av_log()等）

FFmpeg源代码简单分析：结构体成员管理系统-AVClass

FFmpeg源代码简单分析：结构体成员管理系统-AVOption

FFmpeg源代码简单分析：libswscale的sws_getContext()

FFmpeg源代码简单分析：libswscale的sws_scale()

FFmpeg源代码简单分析：libavdevice的avdevice_register_all()

FFmpeg源代码简单分析：libavdevice的gdigrab

【脚本】

FFmpeg源代码简单分析：makefile

FFmpeg源代码简单分析：configure

【H.264】

FFmpeg的H.264解码器源代码简单分析：概述

==========

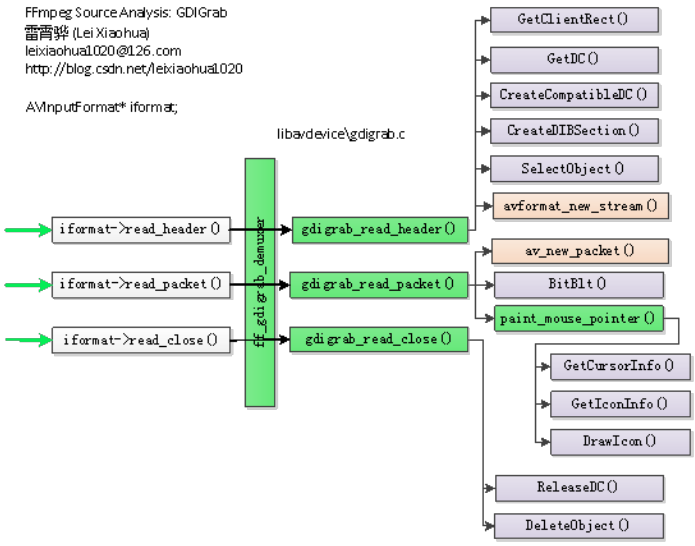
本文记录FFmpeg的libavdevice中GDIGrab组件的源代码。GDIGrab用于在Windows下屏幕录像（抓屏）。在ffmpeg.exe中使用可以参考文章：

FFmpeg获取DirectShow设备数据（摄像头，录屏）

编程使用可以参考文章：

最简单的基于FFmpeg的AVDevice例子（屏幕录制）

gdigrab的源代码位于libavdevice\gdigrab.c。关键函数的调用关系图如下图所示。图中绿色背景的函数代表源代码中自己声明的函数，紫色背景的函数代表Win32的API函数。



ff_gdigrab_demuxer

在FFmpeg中Device也被当做是一种Format，因为GDIGrab是一个输入设备，因此被当作一个AVInputFormat。GDIGrab对应的AVInputFormat结构体如下所示。

```cpp
AVInputFormat ff_gdigrab_demuxer = {
    .name           = "gdigrab",
    .long_name      = NULL_IF_CONFIG_SMALL("GDI API Windows frame grabber"),
    .priv_data_size = sizeof(struct gdigrab),
    .read_header    = gdigrab_read_header,
    .read_packet    = gdigrab_read_packet,
    .read_close     = gdigrab_read_close,
    .flags          = AVFMT_NOFILE,
    .priv_class     = &gdigrab_class,
};
```

从该结构体可以看出：

设备名称是“gdigrab”；

设备完整名称是“GDI API Windows frame grabber”；

初始化函数指针read_header()指向gdigrab_read_header()；

读取数据函数指针read_packet()指向gdigrab_read_packet()；

关闭函数指针read_close()指向gdigrab_read_close()；

Flags设置为AVFMT_NOFILE；
AVClass指定为gdigrab_class。
下面分析一下这些数据。

gdigrab_class

ff_gdigrab_demuxer指定它的AVClass为一个名称为“gdigrab_class”的静态变量。有关AVClass的概念之前已经记录过，在这里不再重复。gdigrab_class的定义如下。

```cpp
static const AVClass gdigrab_class = {
    .class_name = "GDIgrab indev",
    .item_name  = av_default_item_name,
    .option     = options,
    .version    = LIBAVUTIL_VERSION_INT,
};
```

从gdigrab_class的定义可以看出，它指定了一个名称为“options”的数组作为它的选项数组（赋值给AVClass的option变量）。

options

下面看一下这个options数组的定义，如下所示。

```cpp
#define OFFSET(x) offsetof(struct gdigrab, x)
#define DEC AV_OPT_FLAG_DECODING_PARAM
static const AVOption options[] = {
    { "draw_mouse", "draw the mouse pointer", OFFSET(draw_mouse), AV_OPT_TYPE_INT, {.i64 = 1}, 0, 1, DEC },
    { "show_region", "draw border around capture area", OFFSET(show_region), AV_OPT_TYPE_INT, {.i64 = 0}, 0, 1, DEC },
    { "framerate", "set video frame rate", OFFSET(framerate), AV_OPT_TYPE_VIDEO_RATE, {.str = "ntsc"}, 0, 0, DEC },
    { "video_size", "set video frame size", OFFSET(width), AV_OPT_TYPE_IMAGE_SIZE, {.str = NULL}, 0, 0, DEC },
    { "offset_x", "capture area x offset", OFFSET(offset_x), AV_OPT_TYPE_INT, {.i64 = 0}, INT_MIN, INT_MAX, DEC },
    { "offset_y", "capture area y offset", OFFSET(offset_y), AV_OPT_TYPE_INT, {.i64 = 0}, INT_MIN, INT_MAX, DEC },
    { NULL },
};
```

options数组中包含了该Device支持的选项。可以看出GDIGrab支持如下选项：
draw_mouse：画出鼠标指针。
show_region：划出抓屏区域的边界。
framerate：抓屏帧率。
video_size：抓屏的大小。
offset_x：抓屏起始点x轴坐标。
offset_y：抓屏起始点y轴坐标。
从宏定义“#define OFFSET(x) offsetof(struct gdigrab, x)”中可以看出，这些选项都存储在一个名称为“gdigrab”的结构体中。

Gdigrab 上下文结构体

Gdigrab上下文结构体中存储了GDIGrab设备用到的各种变量，定义如下所示。

```cpp
/**
 * GDI Device Demuxer context
 */
struct gdigrab {
    const AVClass *class;   /**< Class for private options */

    int        frame_size;  /**< Size in bytes of the frame pixel data */
    int        header_size; /**< Size in bytes of the DIB header */
    AVRational time_base;   /**< Time base */
    int64_t    time_frame;  /**< Current time */

    int        draw_mouse;  /**< Draw mouse cursor (private option) */
    int        show_region; /**< Draw border (private option) */
    AVRational framerate;   /**< Capture framerate (private option) */
    int        width;       /**< Width of the grab frame (private option) */
    int        height;      /**< Height of the grab frame (private option) */
    int        offset_x;    /**< Capture x offset (private option) */
    int        offset_y;    /**< Capture y offset (private option) */

    HWND       hwnd;        /**< Handle of the window for the grab */
    HDC        source_hdc;  /**< Source device context */
    HDC        dest_hdc;    /**< Destination, source-compatible DC */
    BITMAPINFO bmi;         /**< Information describing DIB format */
    HBITMAP    hbmp;        /**< Information on the bitmap captured */
    void      *buffer;      /**< The buffer containing the bitmap image data */
    RECT       clip_rect;   /**< The subarea of the screen or window to clip */

    HWND       region_hwnd; /**< Handle of the region border window */

    int cursor_error_printed;
};
```

gdigrab_read_header()

gdigrab_read_header()用于初始化gdigrab。函数的定义如下所示。

```cpp
/**
 * Initializes the gdi grab device demuxer (public device demuxer API).
 *
 * @param s1 Context from avformat core
 * @return AVERROR_IO error, 0 success
 */
static int
gdigrab_read_header(AVFormatContext *s1)
{
    struct gdigrab *gdigrab = s1->priv_data;
    //窗口句柄
    HWND hwnd;
    HDC source_hdc = NULL;
    HDC dest_hdc   = NULL;
    BITMAPINFO bmi;
    HBITMAP hbmp   = NULL;
    void *buffer   = NULL;

    const char *filename = s1->filename;
    const char *name     = NULL;
    AVStream   *st       = NULL;

    int bpp;
    RECT virtual_rect;
    //窗口的位置和大小
    RECT clip_rect;
    BITMAP bmp;
    int ret;
    //filename为窗口名称
    if (!strncmp(filename, "title=", 6)) {
        name = filename + 6;
        //查找窗口的句柄
        hwnd = FindWindow(NULL, name);
        if (!hwnd) {
            av_log(s1, AV_LOG_ERROR,
                   "Can't find window '%s', aborting.\n", name);
            ret = AVERROR(EIO);
            goto error;
        }
        if (gdigrab->show_region) {
            av_log(s1, AV_LOG_WARNING,
                    "Can't show region when grabbing a window.\n");
            gdigrab->show_region = 0;
        }
        //filename为desktop
    } else if (!strcmp(filename, "desktop")) {
        //窗口句柄为NULL
        hwnd = NULL;
    } else {
```

```
        av_log(s1, AV_LOG_ERROR,
               "Please use \"desktop\" or \"title=<windowname>\" to specify your target.\n");
        ret = AVERROR(EIO);
        goto error;
    }

    if (hwnd) {
        GetClientRect(hwnd, &virtual_rect);
    } else {
        //窗口句柄为NULL，代表是全屏
        virtual_rect.left = GetSystemMetrics(SM_XVIRTUALSCREEN);
        virtual_rect.top = GetSystemMetrics(SM_YVIRTUALSCREEN);
        virtual_rect.right = virtual_rect.left + GetSystemMetrics(SM_CXVIRTUALSCREEN);
        virtual_rect.bottom = virtual_rect.top + GetSystemMetrics(SM_CYVIRTUALSCREEN);
    }

    /* If no width or height set, use full screen/window area */
    if (!gdigrab->width || !gdigrab->height) {
        clip_rect.left = virtual_rect.left;
        clip_rect.top = virtual_rect.top;
        clip_rect.right = virtual_rect.right;
        clip_rect.bottom = virtual_rect.bottom;
    } else {
        clip_rect.left = gdigrab->offset_x;
        clip_rect.top = gdigrab->offset_y;
        clip_rect.right = gdigrab->width + gdigrab->offset_x;
        clip_rect.bottom = gdigrab->height + gdigrab->offset_y;
    }

    if (clip_rect.left < virtual_rect.left ||
            clip_rect.top < virtual_rect.top ||
            clip_rect.right > virtual_rect.right ||
            clip_rect.bottom > virtual_rect.bottom) {
            av_log(s1, AV_LOG_ERROR,
                    "Capture area (%li,%li),(%li,%li) extends outside window area (%li,%li),(%li,%li)",
                    clip_rect.left, clip_rect.top,
                    clip_rect.right, clip_rect.bottom,
                    virtual_rect.left, virtual_rect.top,
                    virtual_rect.right, virtual_rect.bottom);
            ret = AVERROR(EIO);
            goto error;
    }

    /* This will get the device context for the selected window, or if
     * none, the primary screen */
    //得到某个窗口句柄的DC
    source_hdc = GetDC(hwnd);
    if (!source_hdc) {
        WIN32_API_ERROR("Couldn't get window device context");
        ret = AVERROR(EIO);
        goto error;
    }
    bpp = GetDeviceCaps(source_hdc, BITSPIXEL);

    if (name) {
        av_log(s1, AV_LOG_INFO,
               "Found window %s, capturing %lix%lix%i at (%li,%li)\n",
               name,
               clip_rect.right - clip_rect.left,
               clip_rect.bottom - clip_rect.top,
               bpp, clip_rect.left, clip_rect.top);
    } else {
        av_log(s1, AV_LOG_INFO,
               "Capturing whole desktop as %lix%lix%i at (%li,%li)\n",
               clip_rect.right - clip_rect.left,
               clip_rect.bottom - clip_rect.top,
               bpp, clip_rect.left, clip_rect.top);
    }

    if (clip_rect.right - clip_rect.left <= 0 ||
            clip_rect.bottom - clip_rect.top <= 0 || bpp%8) {
        av_log(s1, AV_LOG_ERROR, "Invalid properties, aborting\n");
        ret = AVERROR(EIO);
        goto error;
    }
    //创建一个与指定设备兼容的HDC
    dest_hdc = CreateCompatibleDC(source_hdc);
    if (!dest_hdc) {
        WIN32_API_ERROR("Screen DC CreateCompatibleDC");
        ret = AVERROR(EIO);
        goto error;
    }

    /* Create a DIB and select it into the dest_hdc */
    //BMP
    bmi.bmiHeader.biSize          = sizeof(BITMAPINFOHEADER);
    bmi.bmiHeader.biWidth         = clip_rect.right - clip_rect.left;
    bmi.bmiHeader.biHeight        = -(clip_rect.bottom - clip_rect.top);
    bmi.bmiHeader.biPlanes        = 1;
    bmi.bmiHeader.biBitCount      = bpp;
    bmi.bmiHeader.biCompression   = BI_RGB;
```

```
    bmi.bmiHeader.biSizeImage     = 0;
    bmi.bmiHeader.biXPelsPerMeter = 0;
    bmi.bmiHeader.biYPelsPerMeter = 0;
    bmi.bmiHeader.biClrUsed       = 0;
    bmi.bmiHeader.biClrImportant  = 0;
    hbmp = CreateDIBSection(dest_hdc, &bmi, DIB_RGB_COLORS,
            &buffer, NULL, 0);
    if (!hbmp) {
        WIN32_API_ERROR("Creating DIB Section");
        ret = AVERROR(EIO);
        goto error;
    }

    if (!SelectObject(dest_hdc, hbmp)) {
        WIN32_API_ERROR("SelectObject");
        ret = AVERROR(EIO);
        goto error;
    }

    /* Get info from the bitmap */
    GetObject(hbmp, sizeof(BITMAP), &bmp);
    //创建AVStream
    st = avformat_new_stream(s1, NULL);
    if (!st) {
        ret = AVERROR(ENOMEM);
        goto error;
    }
    avpriv_set_pts_info(st, 64, 1, 1000000); /* 64 bits pts in us */
    //保存信息到GDIGrab上下文结构体
    gdigrab->frame_size  = bmp.bmWidthBytes * bmp.bmHeight * bmp.bmPlanes;
    gdigrab->header_size = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER) +
                           (bpp <= 8 ? (1 << bpp) : 0) * sizeof(RGBQUAD) /* palette size */;
    gdigrab->time_base   = av_inv_q(gdigrab->framerate);
    gdigrab->time_frame  = av_gettime() / av_q2d(gdigrab->time_base);

    gdigrab->hwnd       = hwnd;
    gdigrab->source_hdc = source_hdc;
    gdigrab->dest_hdc   = dest_hdc;
    gdigrab->hbmp       = hbmp;
    gdigrab->bmi        = bmi;
    gdigrab->buffer     = buffer;
    gdigrab->clip_rect  = clip_rect;

    gdigrab->cursor_error_printed = 0;

    if (gdigrab->show_region) {
        if (gdigrab_region_wnd_init(s1, gdigrab)) {
            ret = AVERROR(EIO);
            goto error;
        }
    }

    st->codec->codec_type = AVMEDIA_TYPE_VIDEO;
    st->codec->codec_id   = AV_CODEC_ID_BMP;
    st->codec->time_base  = gdigrab->time_base;
    st->codec->bit_rate   = (gdigrab->header_size + gdigrab->frame_size) * 1/av_q2d(gdigrab->time_base) * 8;

    return 0;

error:
    //如果出错了
    if (source_hdc)
        ReleaseDC(hwnd, source_hdc);
    if (dest_hdc)
        DeleteDC(dest_hdc);
    if (hbmp)
        DeleteObject(hbmp);
    if (source_hdc)
        DeleteDC(source_hdc);
    return ret;
}
```

从源代码可以看出，gdigrab_read_header()的流程大致如下所示：

（1）确定窗口的句柄hwnd。如果指定了“title=”的话，调用FindWindow()获取hwnd；如果指定了“desktop”，则设定hwnd为NULL。

（2）根据窗口的句柄hwnd确定抓屏的矩形区域。如果抓取指定窗口，则通过GetClientRect()函数；否则就抓取整个屏幕。

（3）调用GDI的API完成抓屏的一些初始化工作。包括：

a)通过GetDC()获得某个窗口句柄的HDC（在这里是source_hdc）。

b)通过CreateCompatibleDC()创建一个与指定设备兼容的HDC（在这里是dest_hdc）

c)通过CreateDIBSection()创建HBITMAP

d)通过SelectObject()绑定HBITMAP和HDC（指的是dest_hdc）

（4）通过avformat_new_stream()创建一个AVStream。

（5）将初始化时候的一些参数保存至GDIGrab的上下文结构体。

gdigrab_read_packet()

gdigrab_read_packet()用于读取一帧抓屏数据。该函数的定义如下所示。

```cpp
/**
 * Grabs a frame from gdi (public device demuxer API).
 *
 * @param s1 Context from avformat core
 * @param pkt Packet holding the grabbed frame
 * @return frame size in bytes
 */
static int gdigrab_read_packet(AVFormatContext *s1, AVPacket *pkt)
{
    struct gdigrab *gdigrab = s1->priv_data;
    //读取参数
    HDC        dest_hdc   = gdigrab->dest_hdc;
    HDC        source_hdc = gdigrab->source_hdc;
    RECT       clip_rect  = gdigrab->clip_rect;
    AVRational time_base  = gdigrab->time_base;
    int64_t    time_frame = gdigrab->time_frame;

    BITMAPFILEHEADER bfh;
    int file_size = gdigrab->header_size + gdigrab->frame_size;

    int64_t curtime, delay;

    /* Calculate the time of the next frame */
    time_frame += INT64_C(1000000);

    /* Run Window message processing queue */
    if (gdigrab->show_region)
        gdigrab_region_wnd_update(s1, gdigrab);

    /* wait based on the frame rate */
    //延时
    for (;;) {
        curtime = av_gettime();
        delay = time_frame * av_q2d(time_base) - curtime;
        if (delay <= 0) {
            if (delay < INT64_C(-1000000) * av_q2d(time_base)) {
                time_frame += INT64_C(1000000);
            }
            break;
        }
        if (s1->flags & AVFMT_FLAG_NONBLOCK) {
            return AVERROR(EAGAIN);
        } else {
            av_usleep(delay);
        }
    }
    //新建一个AVPacket
    if (av_new_packet(pkt, file_size) < 0)
        return AVERROR(ENOMEM);
    pkt->pts = curtime;

    /* Blit screen grab */
    //关键：BitBlt()完成抓屏功能
    if (!BitBlt(dest_hdc, 0, 0,
                clip_rect.right - clip_rect.left,
                clip_rect.bottom - clip_rect.top,
                source_hdc,
                clip_rect.left, clip_rect.top, SRCCOPY | CAPTUREBLT)) {
        WIN32_API_ERROR("Failed to capture image");
        return AVERROR(EIO);
    }
    //画鼠标指针？
    if (gdigrab->draw_mouse)
        paint_mouse_pointer(s1, gdigrab);

    /* Copy bits to packet data */
    //BMP文件头BITMAPFILEHEADER
    bfh.bfType = 0x4d42; /* "BM" in little-endian */
    bfh.bfSize = file_size;
    bfh.bfReserved1 = 0;
    bfh.bfReserved2 = 0;
    bfh.bfOffBits = gdigrab->header_size;
    //往AVPacket中拷贝数据
    //拷贝BITMAPFILEHEADER
    memcpy(pkt->data, &bfh, sizeof(bfh));
    //拷贝BITMAPINFOHEADER
    memcpy(pkt->data + sizeof(bfh), &gdigrab->bmi.bmiHeader, sizeof(gdigrab->bmi.bmiHeader));
    //不常见
    if (gdigrab->bmi.bmiHeader.biBitCount <= 8)
        GetDIBColorTable(dest_hdc, 0, 1 << gdigrab->bmi.bmiHeader.biBitCount,
                (RGBQUAD *) (pkt->data + sizeof(bfh) + sizeof(gdigrab->bmi.bmiHeader)));
    //拷贝像素数据
    memcpy(pkt->data + gdigrab->header_size, gdigrab->buffer, gdigrab->frame_size);

    gdigrab->time_frame = time_frame;

    return gdigrab->header_size + gdigrab->frame_size;
}
```

从源代码可以看出，gdigrab_read_packet()的流程大致如下所示：
（1）从GDIGrab上下文结构体读取初始化时候设定的参数。
（2）根据帧率参数进行延时。
（3）通过av_new_packet()新建一个AVPacket。
（4）通过BitBlt()完成抓屏功能。
（5）如果需要画鼠标指针的话，调用paint_mouse_pointer()，这里不做分析。
（6）按照顺序拷贝以下3项内容至AVPacket的data指向的内存：
a)BITMAPFILEHEADER
b)BITMAPINFOHEADER
c)抓屏的到的像素数据

gdigrab_read_close()

gdigrab_read_close()用于关闭gdigrab。该函数的定义如下所示。

```cpp
/**
 * Closes gdi frame grabber (public device demuxer API).
 *
 * @param s1 Context from avformat core
 * @return 0 success, !0 failure
 */
static int gdigrab_read_close(AVFormatContext *s1)
{
    struct gdigrab *s = s1->priv_data;

    if (s->show_region)
        gdigrab_region_wnd_destroy(s1, s);

    if (s->source_hdc)
        ReleaseDC(s->hwnd, s->source_hdc);
    if (s->dest_hdc)
        DeleteDC(s->dest_hdc);
    if (s->hbmp)
        DeleteObject(s->hbmp);
    if (s->source_hdc)
        DeleteDC(s->source_hdc);

    return 0;
}
```

从源代码可以看出，gdigrab_read_close ()完成了各种变量的清理工作。

雷霄骅
leixiaohua1020@126.com
http://blog.csdn.net/leixiaohua1020

文章标签： FFmpeg GDI 抓屏 屏幕录像 源代码
个人分类： FFMPEG
所属专栏： FFmpeg