mplayer源代码解析 （资料汇总）——vedio

 (2011-12-28 17:16:20)

[[http://simg.sinajs.cn/blog7style/images/common/sg_trans.gif](javascript:;)转载▼](javascript:;)

|  |  |
| --- | --- |
| 标签：  [linux\_mplayer](http://search.sina.com.cn/?c=blog&q=linux_mplayer&by=tag)    [杂谈](http://search.sina.com.cn/?c=blog&q=%D4%D3%CC%B8&by=tag) | 分类： [linux\_mplayer](http://blog.sina.com.cn/s/articlelist_1896501717_1_1.html" \t "_blank) |

原文地址： <http://qzone.qq.com/blog/81182980-1228458373>   
从Mplayer.c的main开始   
处理参数   
mconfig = m\_config\_new();   
m\_config\_register\_options(mconfig,mplayer\_opts);   
// TODO : add something to let modules register their options   
mp\_input\_register\_options(mconfig);   
parse\_cfgfiles(mconfig);   
初始化mpctx结构体，mpctx应该是mplayer context的意思，顾名思义是一个统筹全局的变量。   
static MPContext \*mpctx = &mpctx\_s;   
// Not all functions in mplayer.c take the context as an argument yet   
static MPContext mpctx\_s = {   
.osd\_function = OSD\_PLAY,   
.begin\_skip = MP\_NOPTS\_VALUE,   
.play\_tree\_step = 1,   
.global\_sub\_pos = -1,   
.set\_of\_sub\_pos = -1,   
.file\_format = DEMUXER\_TYPE\_UNKNOWN,   
.loop\_times = -1,   
#ifdef HAS\_DVBIN\_SUPPORT   
.last\_dvb\_step = 1,   
#endif   
};   
原型   
typedef struct MPContext {   
int osd\_show\_percentage;   
int osd\_function;   
ao\_functions\_t \*audio\_out;   
play\_tree\_t \*playtree;   
play\_tree\_iter\_t \*playtree\_iter;   
int eof;   
int play\_tree\_step;   
int loop\_times;   
stream\_t \*stream;   
demuxer\_t \*demuxer;   
sh\_audio\_t \*sh\_audio;   
sh\_video\_t \*sh\_video;   
demux\_stream\_t \*d\_audio;   
demux\_stream\_t \*d\_video;   
demux\_stream\_t \*d\_sub;   
mixer\_t mixer;   
vo\_functions\_t \*video\_out;   
// Frames buffered in the vo ready to flip. Currently always 0 or 1.   
// This is really a vo variable but currently there’s no suitable vo   
// struct.   
int num\_buffered\_frames;   
// AV sync: the next frame should be shown when the audio out has this   
// much (in seconds) buffered data left. Increased when more data is   
// written to the ao, decreased when moving to the next frame.   
// In the audio-only case used as a timer since the last seek   
// by the audio CPU usage meter.   
double delay;   
float begin\_skip; ///< start time of the current skip while on edlout mode   
// audio is muted if either EDL or user activates mute   
short edl\_muted; ///< Stores whether EDL is currently in muted mode.   
short user\_muted; ///< Stores whether user wanted muted mode.   
int global\_sub\_size; // this encompasses all subtitle sources   
int global\_sub\_pos; // this encompasses all subtitle sources   
int set\_of\_sub\_pos;   
int set\_of\_sub\_size;   
int global\_sub\_indices[SUB\_SOURCES];   
一些GUI相关的操作   
打开字幕流   
打开音视频流   
mpctx->stream=open\_stream(filename,0,&mpctx->file\_format);   
fileformat文件还是TV流DEMUXER\_TYPE\_PLAYLIST或DEMUXER\_TYPE\_UNKNOWN DEMUXER\_TYPE\_TV   
current\_module记录状态vobsub  open\_stream handle\_playlist dumpstream   
stream\_reset(mpctx->stream);   
stream\_seek(mpctx->stream,mpctx->stream->start\_pos);   
f=fopen(stream\_dump\_name,”wb”); dump文件流   
stream->type==STREAMTYPE\_DVD   
//============ Open DEMUXERS — DETECT file type ======================   
Demux。分离视频流和音频流   
mpctx->demuxer=demux\_open(mpctx->stream,mpctx->file\_format,audio\_id,video\_id,dvdsub\_id,filename);   
Demux过程   
demux\_open   
get\_demuxer\_type\_from\_name   
……   
mpctx->d\_audio=mpctx->demuxer->audio;   
mpctx->d\_video=mpctx->demuxer->video;   
mpctx->d\_sub=mpctx->demuxer->sub;   
mpctx->sh\_audio=mpctx->d\_audio->sh;   
mpctx->sh\_video=mpctx->d\_video->sh;   
分离了之后就开始分别Play audio和video   
这里只关心play video   
   
vo\_pts=mpctx->sh\_video->timer\*90000.0;   
vo\_fps=mpctx->sh\_video->fps;   
if (!mpctx->num\_buffered\_frames) {   
double frame\_time = update\_video(&blit\_frame);   
mp\_dbg(MSGT\_AVSYNC,MSGL\_DBG2,”\*\*\* ftime=%5.3f \*\*\*\n”,frame\_time);   
if (mpctx->sh\_video->vf\_inited < 0) {   
mp\_msg(MSGT\_CPLAYER,MSGL\_FATAL, MSGTR\_NotInitializeVOPorVO);   
mpctx->eof = 1; goto goto\_next\_file;   
}   
if (frame\_time < 0)   
mpctx->eof = 1;   
else {   
// might return with !eof && !blit\_frame if !correct\_pts   
mpctx->num\_buffered\_frames += blit\_frame;   
time\_frame += frame\_time / playback\_speed;  // for nosound   
}   
}   
关键的函数是update\_video   
根据pts是否正确调整一下同步并在必要的时候丢帧处理。   
最终调用decode\_video开始解码（包括generate\_video\_frame里）。   
mpi = mpvdec->decode(sh\_video, start, in\_size, drop\_frame);   
mpvdec是在main里通过reinit\_video\_chain的一系列调用动态选定的解码程序。   
其实就一结构体。它的原型是   
typedef struct vd\_functions\_s   
{   
vd\_info\_t \*info;   
int (\*init)(sh\_video\_t \*sh);   
void (\*uninit)(sh\_video\_t \*sh);   
int (\*control)(sh\_video\_t \*sh,int cmd,void\* arg, …);   
mp\_image\_t\* (\*decode)(sh\_video\_t \*sh,void\* data,int len,int flags);   
} vd\_functions\_t;   
这是所有解码器必须实现的接口。   
int (\*init)(sh\_video\_t \*sh);是一个名为init的指针，指向一个接受sh\_video\_t \*类型参数，并返回int类型值的函数地址。   
那些vd\_开头的文件都是解码相关的。随便打开一个vd文件以上几个函数和info变量肯定都包含了。   
mpi被mplayer用来存储解码后的图像。在mp\_image.h里定义。   
typedef struct mp\_image\_s {   
unsigned short flags;   
unsigned char type;   
unsigned char bpp;  // bits/pixel. NOT depth! for RGB it will be n\*8   
unsigned int imgfmt;   
int width,height;  // stored dimensions   
int x,y,w,h;  // visible dimensions   
unsigned char\* planes[MP\_MAX\_PLANES];   
int stride[MP\_MAX\_PLANES];   
char \* qscale;   
int qstride;   
int pict\_type; // 0->unknown, 1->I, 2->P, 3->B   
int fields;   
int qscale\_type; // 0->mpeg1/4/h263, 1->mpeg2   
int num\_planes;   
   
int chroma\_width;   
int chroma\_height;   
int chroma\_x\_shift; // horizontal   
int chroma\_y\_shift; // vertical   
   
void\* priv;   
} mp\_image\_t;   
图 像在解码以后会输出到显示器，mplayer本来就是一个视频播放器么。但也有可能作为输入提供给编码器进行二次编码，MP附带的 mencoder.exe就是专门用来编码的。在这之前可以定义filter对图像进行处理，以实现各种效果。所有以vf\_开头的文件，都是这样的 filter。   
图像的显示是通过vo，即video out来实现的。解码器只负责把解码完成的帧传给vo，怎样显示就不用管了。这也是平台相关性最大的部分，单独分出来的好处是不言而喻的，像在 Windows下有通过direcx实现的vo，Linux下有输出到X的vo。vo\_\*文件是各种不同的vo实现，只是他们不都是以显示为目的，像 vo\_md5sum.c只是计算一下图像的md5值。   
在解码完成以后，即得到mpi以后，filter\_video被调用，其结果是整个filter链上的所有filter都被调用了一遍，包括最后的VO，在vo的put\_image里把图像输出到显示器。这个时候需要考虑的是图像存储的方法即用哪种色彩空间。

[MPlayer core]   
| (1)   
\_\_\_\_\_V\_\_\_\_\_\_   (2)   /~~~~~~~~~~/     (3,4)   |~~~~~~|   
|           | —–> | vd\_XXX.c |   ——-> | vd.c |   
| decvideo |         /\_\_\_\_\_\_\_\_\_\_/   <-(3a)– |\_\_\_\_\_\_|   
|           | —–,   ,………….(3a,4a)…..:   
~~~~~~~~~~~~   (6) V   V   
/~~~~~~~~/     /~~~~~~~~/   (8)   
| vf\_X.c | –> | vf\_Y.c | —->   vf\_vo.c / ve\_XXX.c   
/\_\_\_\_\_\_\_\_/     /\_\_\_\_\_\_\_\_/   
|               ^   
(7) |   |~~~~~~|   : (7a)   
`-> | vf.c |…:   
|\_\_\_\_\_\_|   
感觉Mplayer的开发人员们都是无比的牛，硬是用原始的C实现了很多OO语言才支持的特性，带来不好的结果是代码看起来比较费劲.  
   
原文地址： <http://browser.bokee.com/2171300.html>

  注：本文只关注FBDEV相关的内容，其他的或不提或略过。[由于时间比较紧,许多地方肯定还有不足的地方,欢迎指教]  
首先从Mplayer.c中的main函数开始，  
注意到  
//================== Init VIDEO (codec & libvo) ==========================  
current\_module=”preinit\_libvo”;  
if(!(video\_out=init\_best\_video\_out(video\_driver\_list)))

这将跳进Video\_out.c  
for(i=0;video\_out\_drivers[i];i++){  
if(!video\_driver->preinit(vo\_subdevice))  
如果你的FBDEV打开的话，即若有  
#ifdef HAVE\_FBDEV  
&video\_out\_fbdev,  
&video\_out\_fbdev2,  
那么，video\_driver->preinit(vo\_subdevice)这句话将跳进Vo\_fbdev.c中执行preinit操作。  
Preinit中进行了  
fb\_dev\_fd = open(fb\_dev\_name, O\_RDWR)  
和一些ioctl对framebuffer设备的信息进行读取：  
ioctl(fb\_dev\_fd, FBIOGET\_VSCREENINFO, &fb\_vinfo)  
至此，FBDEV相关的pre初始化操作基本结束。

。。。省略其他无关的信息  
进入  
  
uint32\_t (\*draw\_slice)(uint8\_t \*src[], int stride[], int w,int h, int x,int y);  
也就是此时将调用Vo\_fbdev.c中的  
static uint32\_t draw\_slice(uint8\_t \*src[], int stride[], int w, int h, int x, int y)  
{  
uint8\_t \*d;  
uint8\_t \*s;

d = center + (fb\_line\_len \* y + x) \* fb\_pixel\_size;

s = src[0];  
while (h) {  
memcpy(d, s, w \* fb\_pixel\_size);  
d += fb\_line\_len;  
s += stride[0];  
h–;  
}

return 0;  
}  
从而完成从mpi到framebuffer的内存拷贝！