千 Line 으로 비디오플레이어 만들기 4

멀티미디어 : 2009/02/17 03:56

ffmpeg 천줄로 비디오플레이어 만들기
http://www.dranger.com/ffmpeg/ffmpeg.html
http://hybridego.net/entry/천줄로-비디오플레이어-만들기
http://hybridego.net/entry/千-Line-으로-비디오플레이어-만들기-1
http://hybridego.net/entry/千-Line-으로-비디오플레이어-만들기-2
http://hybridego.net/entry/千-Line-으로-비디오플레이어-만들기-3
http://hybridego.net/entry/千-Line-으로-비디오플레이어-만들기-4
http://hybridego.net/entry/千-Line-으로-비디오플레이어-만들기-5
http://hybridego.net/entry/千-Line으로-비디오플레이어-만들기-6
http://hybridego.net/entry/千-Line으로-비디오플레이어-만들기-7
http://hybridego.net/entry/千-Line으로-비디오플레이어-만들기-8
http://hybridego.net/entry/千-Line으로-비디오플레이어-만들기-마무리
별로 중요하지 않은 부분중 생략된 부분이 있고 의역이 많이 있습니다.
군데군데 틀린 번역이 있을수 있습니다. (영어가 후달려서)

하지만 소스코드 설명 부분은 틀리지 않도록 노력했습니다.

위의 원문을 번역한 것입니다.
잘못된 부분이 있을지도 모르겠습니다.
혹시 잘못된 부분을 발견하시면 댓글로 알려주시면 감사하겠습니다.

이번 부분은 오역이 많은것 같습니다. ㅠ.ㅠ 소스코드도 이리뛰고 저리뛰고 그래서 정신이 없네요.

Tutorial 04: Spawning Threads

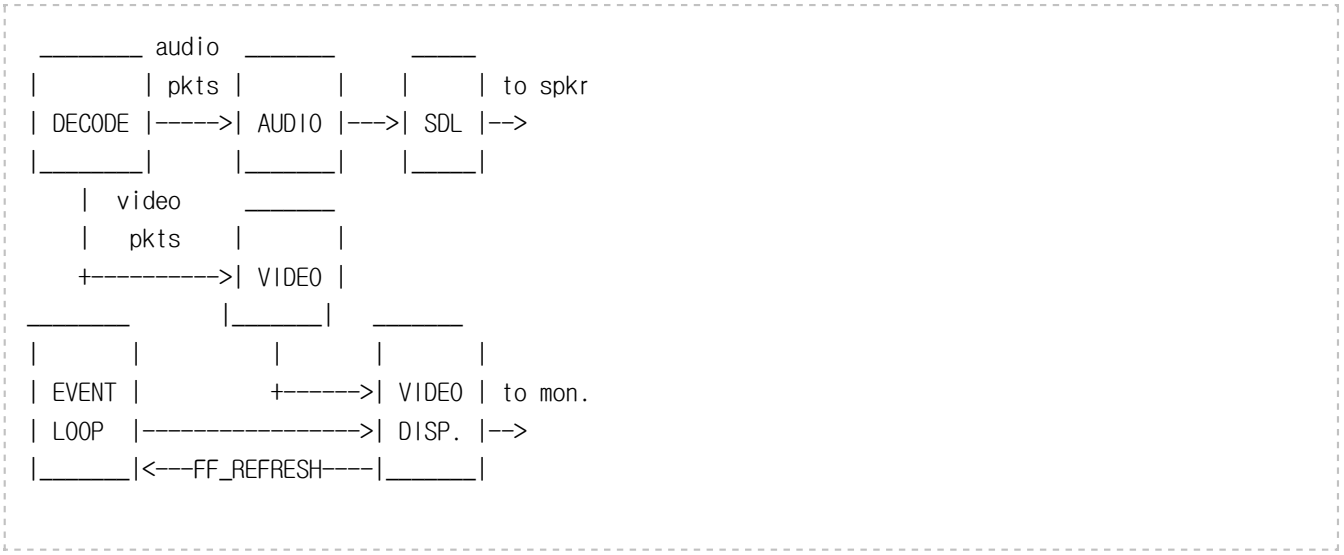
tutorial04.c

Overview

지난번에는 SDL의 오디오 기능을 이용해서 오디오 재생기능을 추가해봤었습니다.
SDL은 오디오가 필요할때 마다 콜백 할수를 정의한 쓰레드를 시작했었습니다. 이제 우리는 비디오 디스플레이 같은 종류의 작업을 할 것입니다. 지금 만들 코드는 Sync를 적용할 때 더 작동하기 쉽게 모듈화 하는 코드 입니다.
자~ 그럼 어디서부터 시작할까요?

먼저 우리의 main 함수를 보면 완전 개판입니다. ㅋ
지금은 이벤트 루프, 패킷에서 읽어오기, 비디오 디코딩이 전부 다 들어있습니다. @_@ 그래서 이제 이것들을 다 분리해낼 것입니다. 우리는 패킷을 디코딩하는것을 담당하는 쓰레드를 만들것 입니다. 이 패킷들은 큐에 추가되고 오디오, 비디오 쓰레드에 맞게 읽어질 것입니다. 오디오 쓰레드는 이미 우리가 원하는 대로 셋업되어 있고 비디오 쓰레드

는 비디오를 디스플레이 해야하기 때문에 조금더 복잡해질 것입니다. 이제 메인 루프에 실질적으로 비디오 디스플레이 하는 코드를 넣겠습니다. 그냥 루프돌면서 재생하는게 아니라 이벤트 루프에 따라서 비디오가 디스플레이 되도록 하겠습니다. 이 비디오 디코드 개념은 결과물 프레임을 다른 queue에 넣는 것입니다. 그러고 FF_REFRESH_EVENT라는 커스텀 이벤트를 만들어서 이벤트 시스템에 추가 합니다. 그리고 이벤트 루프에서 이벤트를 감지하면 큐에있는 다음 프레임을 디스플레이 합니다.
다음의 흐름도를 참고하세요



SDL_Delay 쓰레드를 이용하면 스크린에 다음 비디오 프레임을 보여줄 때를 정확히 컨트롤 할수 있습니다.
다음 튜토리얼에서는 비디오를 싱크 할때, 우리는 간단한 코드를 추가해서 다음 비디오로 refresh 할 것입니다. 그러면 화면이 제때 제때 보이게 될겠지요.

Simplifying Code

우리는 오디오와 비디오 코덱 정보 전부를 갖고 있습니다. 그리고 큐와 버퍼에 추가하는것들을 알고 있습니다. 이 모든것이 하나의 logical unit을 위한 것입니다. 바로 Movie 말이지요. 그래서 우리는 VideoState라는 모든 정보를 담을 수 있는 커다란 구조체를 정의합니다.

view plain copy to clipboard print ?

```
typedef struct VideoState {

  AVFormatContext *pFormatCtx;
  int             videoStream, audioStream;
  AVStream        *audio_st;
  PacketQueue     audioq;
  uint8_t         audio_buf[(AVCODEC_MAX_AUDIO_FRAME_SIZE * 3) / 2];
  unsigned int    audio_buf_size;
  unsigned int    audio_buf_index;
  AVPacket        audio_pkt;
  uint8_t         *audio_pkt_data;
  int             audio_pkt_size;
  AVStream        *video_st;
  PacketQueue     videoq;

  VideoPicture    pictq[VIDEO_PICTURE_QUEUE_SIZE];
  int             pictq_size, pictq_rindex, pictq_windex;
  SDL_mutex       *pictq_mutex;
  SDL_cond        *pictq_cond;

  SDL_Thread      *parse_tid;
  SDL_Thread      *video_tid;

  char            filename[1024];
  int             quit;
} VideoState;
```

AVFormatContext , AVStream , AVPacket , AVStream , SDL_mutex , SDL_cond , SDL_Thread

여기서 잠깐 우리가 뭘 하려고 하는지 알아볼까요?
우선 기본정보를 보겠습니다.
Format Context 와 오디오 비디오의 인덱스들 그리고 AVStream 오브젝트들을 보세요. 우리가 오디오 버퍼를 이 구조체에 넣었던 것을 알수가 있습니다.
audio_buf, audio_buf_size 같은 것들은 전부 오디오를 위한 정보입니다. 우리는 비디오를 위한 큐와 프레임 디코드를 위한 버퍼를 추가했었습니다. VideoPicture struct는 우리가 만든것이구요. 별도의 두 쓰레드를 지정하는 포인터와 quit 플래그, 무비파일의 이름을 지정할 char 배열이 있습니다.

이제 main 함수로 돌아가서 우리 프로그램이 어떻게 바뀌었는지 살펴보겠습니다.
일단 VideoState 구조체를 설정 합니다.

view plain copy to clipboard print ?

```
01.  int main(int argc, char *argv[]) {
02.
03.    SDL_Event       event;
04.
05.    VideoState      *is;
06.
07.    is = av_mallocz(sizeof(VideoState));
```

SDL_Event , av_mallocz()

av_mallocz() 함수는 메모리를 0으로 채워서 할당해줍니다.

그리고 이벤트 루프에서 display 함수를 호출하기 때문에 디스플레이 버퍼(pictq)를 위한 lock을 초기화 해줍니다.
다음에 우리는 qictq에서 pre-decoded 프레임을 가져올 것입니다. 동시에 비디오 디코더는 qictq에 정보를 넣을 것입니다. 우리는 어떤것이 먼저 실행될지 알수 없습니다. 이건 race condition 같은 것임을 알아보실 것입니다. 그래서 쓰레드가 시작하기 전에 이것을 할당하는 것입니다. 그리고 무비파일이름을 VideoState에 카피해 넣습니다.

view plain copy to clipboard print ?

```
01.  pstrcpy(is->filename, sizeof(is->filename), argv[1]);
02.
03.  is->pictq_mutex = SDL_CreateMutex();
04.  is->pictq_cond = SDL_CreateCond();
```

SDL_CreateMutex() , SDL_CreateCond()

pstrcpy 는 ffmpeg 가 제공하는 함수 인데 strncpy 보다 정확합니다.
(저는 psyrcpy 가 라이브러리에 정의되어 있지 않더군요. 그래서 인터넷에서 찾아다가 카피해 넣었습니다. by Real_G)

view plain copy to clipboard print ?

```
01.
02.
03.
04.
05.
06.
07.
08.
09.
```

```
10.
11. void pstrcpy(char *buf, int buf_size, const char *str)
12. {
13.     int c;
14.     char *q = buf;
15.
16.     if (buf_size <= 0)
17.         return;
18.
19.     for(;;) {
20.         c = *str++;
21.         if (c == 0 || q >= buf + buf_size - 1)
22.             break;
23.         *q++ = c;
24.     }
25.     *q = '\0';
26. }
```

Our First Thread

이제 쓰레드를 돌려보겠습니다.

```
view plain  copy to clipboard  print  ?
01. schedule_refresh(is, 40);
02.
03. is->parse_tid = SDL_CreateThread(decode_thread, is);
04. if(!is->parse_tid) {
05.   av_free(is);
06.   return -1;
07. }
```

SDL_CreateThread() , av_free()

schedule_refresh 함수는 다음에 정의하겠습니다.
이것은 기본적으로 지정된 millisecond 후에 FF_REFRESH_EVENT를 시스템에 발생시키는 일을 합니다. 이것은 이벤트 큐에 이것이 나타나면 비디오 refresh함수를 호출합니다.

SDL_CreateThread()함수는 새로운 쓰레드를 만듭니다. 이것은 부모 프로세스의 모든 메모리에 접근 가능하고 이 함수를 호출하면서 실행되기 시작합니다. 위 예제에서는 decode_thread 에 VideoState 구조체를 붙여서 호출했습니다. main 함수의 반은 바뀐것이 없습니다. 파일을 열고 오디오 비디오 스트림의 인덱스를 찾는 내용 그대로 입니다. 한가지 바뀐 것은 format context를 우리의 새로운 큰 구조체에 저장한다는것 뿐이군요.
스트림 인덱스들을 찾은 다음 Stream_component_open() 이라는 함수를 정의해서 호출합니다. 이 함수는 자연적인 방법으로 이것들을 분리합니다. 그리고 비디오와 오디오 코덱의 설정에 비슷한 부분이 많기 때문에 이 함수를 재활용할 것입니다.

stream_component_open() 함수는 코덱 디코더에서 찾을수 있는데 오디오 옵션을 설정하고 우리의 큰 구조체에 중요한 정보를 저장합니다. 그리고 우리의 오디오 비디오 쓰레드를 실행합니다. 이것 말고도 자동으로 코덱을 찾는 것 대신 강제로 설정하는 기능들 다른 여러 옵션을 추가할 수도 있습니다.

```
view plain  copy to clipboard  print  ?
01. int stream_component_open(VideoState *is, int stream_index) {
02.
03.   AVFormatContext *pFormatCtx = is->pFormatCtx;
04.   AVCodecContext *codecCtx;
05.   AVCodec *codec;
06.   SDL_AudioSpec wanted_spec, spec;
07.
```

```
08.   if(stream_index < 0 || stream_index >= pFormatCtx->nb_streams) {
09.     return -1;
10.   }
11.
12.
13.   codecCtx = pFormatCtx->streams[stream_index]->codec;
14.
15.   if(codecCtx->codec_type == CODEC_TYPE_AUDIO) {
16.
17.     wanted_spec.freq = codecCtx->sample_rate;
18.
19.     wanted_spec.callback = audio_callback;
20.     wanted_spec.userdata = is;
21.
22.     if(SDL_OpenAudio(&wanted_spec, &spec) < 0) {
23.       fprintf(stderr, "SDL_OpenAudio: %s\n", SDL_GetError());
24.       return -1;
25.     }
26.   }
27.   codec = avcodec_find_decoder(codecCtx->codec_id);
28.   if(!codec || (avcodec_open(codecCtx, codec) < 0)) {
29.     fprintf(stderr, "Unsupported codec!\n");
30.     return -1;
31.   }
32.
33.   switch(codecCtx->codec_type) {
34.   case CODEC_TYPE_AUDIO:
35.     is->audioStream = stream_index;
36.     is->audio_st = pFormatCtx->streams[stream_index];
37.     is->audio_buf_size = 0;
38.     is->audio_buf_index = 0;
39.     memset(&is->audio_pkt, 0, sizeof(is->audio_pkt));
40.     packet_queue_init(&is->audioq);
41.     SDL_PauseAudio(0);
42.     break;
43.   case CODEC_TYPE_VIDEO:
44.     is->videoStream = stream_index;
45.     is->video_st = pFormatCtx->streams[stream_index];
46.
47.     packet_queue_init(&is->videoq);
48.     is->video_tid = SDL_CreateThread(video_thread, is);
49.     break;
50.   default:
51.     break;
52.   }
53. }
```

AVFormatContext , AVCodecContext , SDL_AudioSpec , SDL_OpenAudio() , avcodec_find_decoder() , avcodec_open() , SDL_PauseAudio() , SDL_CreateThread()

이것은 오디오와 비디오를 처리하는 것만 빼고 전의 코드와 거의 같습니다. aCodecCtx 대신 우리의 큰 구조체의 오디오 콜백을 위한 userdata를 설정합니다. 우리는 audio_st와 video_st 스트림이 자동으로 저장되게 했습니다. 그리고 비디오 큐를 추가했고 오디오 큐도 마찬가지 방법으로 설정했습니다. 거의 모든 포인트는 비디오와 오디오 쓰레드를 실행합니다.

```
view plain  copy to clipboard  print  ?
01.     SDL_PauseAudio(0);
02.     break;
03.
04.
05.
06.     is->video_tid = SDL_CreateThread(video_thread, is);
```

SDL_PauseAudio() , SDL_CreateThread()

지난번에 SDL_PauseAudio() 과 SDL_CreateThread()를 같은 방법으로 사용했던것을 기억 하시나요?
video_thread()함수로 돌아가 봅시다.

아 그전에 decode_thread() 함수의 중간 부분으로 가봅시다. 이 부분은 패킷을 읽고 큐에 넣는 루프입니다.

view plain copy to clipboard print ?

```
 for(;;) {
   if(is->quit) {
     break;
   }

   if(is->audioq.size > MAX_AUDIOQ_SIZE ||
      is->videoq.size > MAX_VIDEOQ_SIZE) {
     SDL_Delay(10);
     continue;
   }
   if(av_read_frame(is->pFormatCtx, packet) < 0) {
     if(url_ferror(&pFormatCtx->pb) == 0) {
SDL_Delay(100);
continue;
     } else {
break;
     }
   }

   if(packet->stream_index == is->videoStream) {
     packet_queue_put(&is->videoq, packet);
   } else if(packet->stream_index == is->audioStream) {
     packet_queue_put(&is->audioq, packet);
   } else {
     av_free_packet(packet);
   }
 }
```

SDL_Delay() , av_read_frame() , url_ferror() , av_free_packet()

여기도 특별히 새로울것은 없습니다.
여기서는 오디오와 비디오 큐를 위해 MAX size를 정해주고 Read 에러를 체크하기 위한 함수가 추가되었습니다.
Format Context에는 pb.라고 부르는 ByteIOContext 구조체가 들어있습니다. ByteIOContext는 low-level 파일 정보를 모두 갖고 있도록 구성되어 있습니다. url_ferror 체크는 File 에서 읽는 도중에 어떤 에러가 발생하는지 볼수 있게 되어있습니다.

루프를 실행한 후, 우리 코드는 프로그램이 끝나거나 끝을 알려주기를 기다리고 있습니다.
다음 코드는 어떻게 이벤트들을 Push 하는지 알려줍니다.

view plain copy to clipboard print ?

```
while(!is->quit) {
  SDL_Delay(100);
}

ail:
if(1){
  SDL_Event event;
  event.type = FF_QUIT_EVENT;
  event.user.data1 = is;
  SDL_PushEvent(&event);
}
return 0;
```

SDL_Delay(), SDL_Event , SDL_PushEvent()

우리는 SDL 의 SDL_USEREVENT를 사용함으로써 User Event를 얻을수 있습니다. 첫번째 User Event는 SDL_USEREVENT값에 의해 할당될 것입니다. 다음은 SDL_USEREVENT + 1 이고 쭉~~~ 이렇게 됩니다. FF_QUIT_EVENT는 우리 프로그램에서 SDL_USEREVENT + 2 로 정의됩니다. 그리고 우리는 User Data를 통과할 수 있습니다. 그리고 여기서는 큰 구조체를 가리키는 포인터를 통과합니다.
드디어 우리는 SDL_PushEvent()를 호출했습니다. 이 이벤트 루프에서 우리는 SDL_QUIT_EVENT 섹센에 의해 단지 이것을 넣기만 할 뿐입니다. 우리는 이벤트 루프를 좀더 자세하게 볼것입니다. 우선은 언제 우리가 FF_QUIT_EVENT를 Push 하는지 확실히 알기만 하면 됩니다. 우리는 다음에 이것을 캐취하고 quit Flag를 발생시킬 것 입니다.

Getting the Frame : video_thread

우리 코덱이 준비된 다음에 비디오 쓰레드를 실행시킬 것입니다. 이 쓰레드는 비디오 큐의 패킷을 읽고 비디오를 프레임으로 디코드 하고 queue_picture 함수를 호출해서 처리된 프레임을 Picture 큐에 넣습니다.

view plain copy to clipboard print ?

```
01.  int video_thread(void *arg) {
02.    VideoState *is = (VideoState *)arg;
03.    AVPacket pkt1, *packet = &pkt1;
04.    int len1, frameFinished;
05.    AVFrame *pFrame;
06.
07.    pFrame = avcodec_alloc_frame();
08.
09.    for(;;) {
10.      if(packet_queue_get(&is->videoq, packet, 1) < 0) {
11.
12.        break;
13.      }
14.
15.      len1 = avcodec_decode_video(is->video_st->codec, pFrame, &frameFinished,
16.                  packet->data, packet->size);
17.
18.
19.      if(frameFinished) {
20.        if(queue_picture(is, pFrame) < 0) {
21.      break;
22.        }
23.      }
24.      av_free_packet(packet);
25.    }
26.    av_free(pFrame);
27.    return 0;
28.  }
```

AVPacket , AVFrame , avcodec_alloc_frame() , avcodec_decode_video() , av_free_packet() , av_free()

이 함수는 몇가지 빼고는 거의 유사합니다. 우리는 avcodec_decode_video 함수를 여기로 옮겼습니다. 그냥 아규먼트 몇개만 바꿨을 뿐입니다. AVStream 을 큰 구조체에 넣었고 거기에서 코덱을 얻습니다. 우리는 에러를 만나거나 종료가 될때 까지 비디오 큐에서 패킷을 쭉쭉 읽기만 하면 됩니다.

Queueing the Frame

디코드된 프레임이 저장된 함수를 보면 Picture 큐에 pFrame가 있습니다. Picture 큐는 SDL 오버레이 이기 때문에 우리 프레임을 변환해야 합니다. 우리가 만든 구조체의 Picture 큐에 그 데이터를 저장 합니다.

view plain copy to clipboard print ?

```
typedef struct VideoPicture {
  SDL_Overlay *bmp;
  int width, height;
  int allocated;
} VideoPicture;
```

우리 큰 구조체는 이것들의 버퍼를 갖고 있고 그곳에 저장할 수 있습니다. 하지만 SDL_Overlay 를 할당해야 합니다.

이 큐를 이용하기 위해 우리는 두개의 포인터를 갖고 있습니다. – 쓸 인덱스와 읽을 인덱스를 가리키는 것 두개요. 그리고 버퍼안에 얼마나 많은 picture가 있는지 계속 체크하고 있어야 합니다. 큐에 쓰기위해 우리는 버퍼가 클리어 되기를 기다려야 합니다. 그래서 VideoPicture를 저장할 공간을 갖고 있습니다. 쓸 인덱스에 이미 오버레이를 할당했다면 그것을 체크해봐야 합니다. 그렇지 않으면 공간을 할당해줘야 합니다. 또한 윈도우의 크기가 바뀌었다면 버퍼를 재할당 해줘야 합니다. 그러나 여기서 할당을 하는것 대신 locking issues를 피하게 하겠습니다.

view plain copy to clipboard print ?

```
int queue_picture(VideoState *is, AVFrame *pFrame) {

  VideoPicture *vp;
  int dst_pix_fmt;
  AVPicture pict;


  SDL_LockMutex(is->pictq_mutex);
  while(is->pictq_size >= VIDEO_PICTURE_QUEUE_SIZE &&
    !is->quit) {
    SDL_CondWait(is->pictq_cond, is->pictq_mutex);
  }
  SDL_UnlockMutex(is->pictq_mutex);

  if(is->quit)
    return -1;


  vp = &is->pictq[is->pictq_windex];


  if(!vp->bmp ||
     vp->width != is->video_st->codec->width ||
     vp->height != is->video_st->codec->height) {
    SDL_Event event;

    vp->allocated = 0;

    event.type = FF_ALLOC_EVENT;
    event.user.data1 = is;
    SDL_PushEvent(&event);


    SDL_LockMutex(is->pictq_mutex);
    while(!vp->allocated && !is->quit) {
      SDL_CondWait(is->pictq_cond, is->pictq_mutex);
    }
    SDL_UnlockMutex(is->pictq_mutex);
    if(is->quit) {
      return -1;
    }
  }
```

AVFrame , AVPicture , SDL_LockMutex() , SDL_CondWait() , SDL_UnlockMutex() , SDL_Event , SDL_PushEvent()

이 이벤트 메카니즘은 우리가 이미 봤던 종료시에 했던 그것과 같습니다. 우리는 FF_ALLOC_EVENT 를

SDL_USEREVENT 처럼 정의했습니다. 이벤트를 발생시키고 할당함수를 돌리는 할당변수를 기다립니다.

우리 이벤트 루프가 어떻게 바뀌었는지 보세요

```
view plain  copy to clipboard  print  ?
01.  for(;;) {
02.    SDL_WaitEvent(&event);
03.    switch(event.type) {
04.
05.    case FF_ALLOC_EVENT:
06.      alloc_picture(event.user.data1);
07.      break;
```

SDL_WaitEvent()

event.user.data1 이 우리 큰 구조체라는 것을 기억하세요.
다음 alloc_picture() 함수를 보시겠습니다.

```
view plain  copy to clipboard  print  ?
01.  void alloc_picture(void *userdata) {
02.
03.    VideoState *is = (VideoState *)userdata;
04.    VideoPicture *vp;
05.
06.    vp = &is->pictq[is->pictq_windex];
07.    if(vp->bmp) {
08.
09.      SDL_FreeYUVOverlay(vp->bmp);
10.    }
11.
12.    vp->bmp = SDL_CreateYUVOverlay(is->video_st->codec->width,
13.                   is->video_st->codec->height,
14.                   SDL_YV12_OVERLAY,
15.                   screen);
16.    vp->width = is->video_st->codec->width;
17.    vp->height = is->video_st->codec->height;
18.
19.    SDL_LockMutex(is->pictq_mutex);
20.    vp->allocated = 1;
21.    SDL_CondSignal(is->pictq_cond);
22.    SDL_UnlockMutex(is->pictq_mutex);
23.  }
```

SDL_FreeYUVOverlay() , SDL_CreateYUVOverlay() , SDL_LockMutex() , SDL_CondSignal() , SDL_UnlockMutex()

여러분은 우리 Main 루프에서 여기로 가져온 SDL_CreateYUVOverlay() 함수를 기억하셔야 합니다. 이 코드는 공정하게 self-explanitory 됩니다. 비디오 크기는 변하면 안되기 때문에 VideoPicture에 width, height 를 저장해야 하는 것을 기억해야 합니다.

자 모두 세팅 되었습니다. 우리는 YUV overlay를 할당했고 picture를 받을 준비가 되었습니다. queue_picture 로 돌아가서 오버레이로 frame을 복사하는 코드를 보겠습니다.

```
view plain  copy to clipboard  print  ?
01.  int queue_picture(VideoState *is, AVFrame *pFrame) {
02.
03.
04.
```

```
05.
06.
07.   if(vp->bmp) {
08.
09.     SDL_LockYUVOverlay(vp->bmp);
10.
11.     dst_pix_fmt = PIX_FMT_YUV420P;
12.
13.
14.     pict.data[0] = vp->bmp->pixels[0];
15.     pict.data[1] = vp->bmp->pixels[2];
16.     pict.data[2] = vp->bmp->pixels[1];
17.
18.     pict.linesize[0] = vp->bmp->pitches[0];
19.     pict.linesize[1] = vp->bmp->pitches[2];
20.     pict.linesize[2] = vp->bmp->pitches[1];
21.
22.
23.     img_convert(&pict, dst_pix_fmt,
24.         (AVPicture *)pFrame, is->video_st->codec->pix_fmt,
25.         is->video_st->codec->width, is->video_st->codec->height);
26.
27.     SDL_UnlockYUVOverlay(vp->bmp);
28.
29.     if(++is->pictq_windex == VIDEO_PICTURE_QUEUE_SIZE) {
30.       is->pictq_windex = 0;
31.     }
32.     SDL_LockMutex(is->pictq_mutex);
33.     is->pictq_size++;
34.     SDL_UnlockMutex(is->pictq_mutex);
35.   }
36.   return 0;
37. }
```

AVFrame , SDL_LockYUVOverlay() , img_convert() , AVPicture , SDL_UnlockYUVOverlay() , SDL_LockMutex() , SDL_UnlockMutex()

이 부분에 중요한 것은 우리가 전에 프레임으로 YUV overlay 를 채웠던 그 것 입니다. 마지막 비트는 간단하게 우리 큐에 값을 추가합니다. 큐는 꽉찰때 까지 계속 추가했다가 큐에 무엇인가 있는 동안 계속 읽어옵니다. 그래서 모든것은 is->pictq_size 에 관련되어 있고 lock이 필요합니다. 우리가 여기에 무엇을 하던간 write pointer 를 증가시켜야 하고 그래서 queue에 lock을 걸고 싸이즈를 늘려야 합니다. 그럼 Reader는 큐에 정보가 더 있는지 알수 있고 큐가 가득차면 Writer 가 인지할 것입니다.

Displaying the Video

이것은 비디오 쓰레드를 위한 것입니다. 우리가 전에 schedule_refresh() 함수를 호출했던 것을 기억하시나요? 이게 실제로 어떻게 되어있는지 보겠습니다.

```
view plain  copy to clipboard  print  ?
01.
02. static void schedule_refresh(VideoState *is, int delay) {
03.   SDL_AddTimer(delay, sdl_refresh_timer_cb, is);
04. }
```

SDL_AddTimer() 는 SDL에서 간한히 user-specfied 한 callback 함수를 만들수 있게 해줍니다. 정확한 millisecond 를 제공하고 추가적으로 user data를 전달할 수 있습니다. 우리는 이 함수를 이용해서 video update를 구현할 것입니다. 이 함수가 호출될 때 마다 타이머를 세팅할것이고 event를 발생시킬것 입니다. 이것을 main() 함수에 넣고 함수를 돌려서 Picture 에서 프레임을 땡겨올 것입니다.

일단 가장 우선으로 할 것은 이벤트를 발생시키는 것입니다.

view plain copy to clipboard print ?

```
static Uint32 sdl_refresh_timer_cb(Uint32 interval, void *opaque) {
  SDL_Event event;
  event.type = FF_REFRESH_EVENT;
  event.user.data1 = opaque;
  SDL_PushEvent(&event);
  return 0;
}
```

SDL_Event , SDL_PushEvent()
여기는 이벤트 Push 하는것과 유사합니다. FF_REFRESH_EVENT는 여기에서 SDL_USEREVENT + 1 로 정의됩니다. 한가지 알아야 할것은 return 0 을 하면 SDL이 타이머를 멈추고 callback 은 다시 만들어지지 않는다는 것입니다.

FF_REFRESH_EVENT 를 push 한것을 이벤트 루프에서 제어해야 합니다.

view plain copy to clipboard print ?

```
for(;;) {

  SDL_WaitEvent(&event);
  switch(event.type) {

  case FF_REFRESH_EVENT:
    video_refresh_timer(event.user.data1);
    break;
```

SDL_WaitEvent()

우리의 Picture 큐에서 데이타를 끌어올 것입니다.

view plain copy to clipboard print ?

```
void video_refresh_timer(void *userdata) {

  VideoState *is = (VideoState *)userdata;
  VideoPicture *vp;

  if(is->video_st) {
    if(is->pictq_size == 0) {
      schedule_refresh(is, 1);
    } else {
      vp = &is->pictq[is->pictq_rindex];


      schedule_refresh(is, 80);


      video_display(is);


      if(++is->pictq_rindex == VIDEO_PICTURE_QUEUE_SIZE) {
    is->pictq_rindex = 0;
      }
      SDL_LockMutex(is->pictq_mutex);
      is->pictq_size--;
      SDL_CondSignal(is->pictq_cond);
      SDL_UnlockMutex(is->pictq_mutex);
    }
  } else {
    schedule_refresh(is, 100);
  }
}
```

우리가 뭔가를 갖고있을때 큐에서 가져오는 매우 간단한 함수 입니다. 다음 비디오 프레임을 보여줘야 할때를 위해 Timer를 설정합니다. 스크린에 비디오를 보여주기 위해서 video_diplay를 호출합니다. 큐에 카운터를 증가시켜주고 사이즈는 줄여줍니다. 여러분은 혹시 이 함수에서 vp가 아무런 역할도 없다는 것을 눈치 채셨을지도 모릅니다. 우리는 다음에 비디오와 오디오를 싱크할때 이것을 이용해서 시간정보를 얻어낼 것입니다. "오옷! 타이밍 코드가 여기에?" 이 섹션에서는 다음 비디오를 얼마나 빨리 보여줄 것인지를 이해하시면 됩니다. 그리고 schedule_refresh() 함수에는 더미값으로 80을 넣겠습니다.
여러분은 이 값을 바꾸고 재 컴파일 해서 확인해 보세요. 어떤 변화가 있을까요?

자.... 이제 거의 다 되었습니다. 이제 마지막 한가지만 하면 되는데요. 비디오를 디스플레이 하는 것이죠
여기서 video_display() 함수가 등장합니다.

view plain copy to clipboard print ?

```
01. void video_display(VideoState *is) {
02.
03.   SDL_Rect rect;
04.   VideoPicture *vp;
05.   AVPicture pict;
06.   float aspect_ratio;
07.   int w, h, x, y;
08.   int i;
09.
10.   vp = &is->pictq[is->pictq_rindex];
11.   if(vp->bmp) {
12.     if(is->video_st->codec->sample_aspect_ratio.num == 0) {
13.       aspect_ratio = 0;
14.     } else {
15.       aspect_ratio = av_q2d(is->video_st->codec->sample_aspect_ratio) *
16.     is->video_st->codec->width / is->video_st->codec->height;
17.     }
18.     if(aspect_ratio <= 0.0) {
19.       aspect_ratio = (float)is->video_st->codec->width /
20.     (float)is->video_st->codec->height;
21.     }
22.     h = screen->h;
23.     w = ((int)rint(h * aspect_ratio)) & -3;
24.     if(w > screen->w) {
25.       w = screen->w;
26.       h = ((int)rint(w / aspect_ratio)) & -3;
27.     }
28.     x = (screen->w - w) / 2;
29.     y = (screen->h - h) / 2;
30.
31.     rect.x = x;
32.     rect.y = y;
33.     rect.w = w;
34.     rect.h = h;
35.     SDL_DisplayYUVOverlay(vp->bmp, &rect);
36.   }
37. }
```

SDL_Rect , AVPicture , av_q2d() , SDL_DisplayYUVOverlay()

스크린 크기가 변할수 있기 때문에 우리는 동적으로 무비의 크기를 측정해야 합니다. 그래서 우리는 일단 무비의 aspect ratio(width를 height로 나눈것) 를 측정해야 합니다. 몇몇 코덱은 홀수의 sample aspect ratio를 갖고 있습니다. 이것은 간단히 싱글 픽셀의 width/hight 비율입니다. Codec context 의 height 와 width 값은 픽셀로 측정되기 때문에 실제 aspect ratio는 sapect ratio 와 sample aspect ratio의 곱과 같습니다. 몇몇 코덱에서 aspect ratio 가 0인것을 볼수 있는데 이것은 간단히 size 가 1x1인것을 가리킵니다. 그러면 이제 우리는 영상을 우리 스크린에 맞출수 있습니다. & -3 bit-twiddling 을 하면 거의 4의 배수의 값이 나옵니다. 그러면 우리는 영상의 중심을 맞추고 SDL_DisplayYUVOverlay()를 호출합니다.

새로운 VideoStruct를 이용하려면 오디오 코드를 다시 써야 합니다. 약간만 바꾸면 됩니다. 다음의 샘플 코드를 보세요
마지막으로 할것은 ffmpeg의 내부 quit 콜백함수를 바꾸는 것입니다.

view plain copy to clipboard print ?

```
01. VideoState *global_video_state;
02.
03. int decode_interrupt_cb(void) {
04.   return (global_video_state && global_video_state->quit);
05. }
```

우리는 global_video_state를 main() 함수의 큰 구조체로 설정했습니다.

자 다되었군요! 컴파일 해봅니다.

```
gcc -o tutorial04 tutorial04.c -lavutil -lavformat -lavcodec -lz -lm ₩
`sdl-config --cflags --libs`
```

자~ 싱크가 안맞는 영상을 감상해 보세요 ~~~~

ㅋㅋㅋ

다음번에는 드디어 실질적인 비디오 플레이어를 만들것 입니다.

千 Line 으로 비디오플레이어 만들기 4 조회수 669

'멀티미디어' 카테고리의 다른 글

千 Line 으로 비디오플레이어 만들기 5 (0) 2009/02/27
千 Line 으로 비디오플레이어 만들기 4 (0) 2009/02/17
千 Line 으로 비디오플레이어 만들기 3 (0) 2009/02/14
千 Line 으로 비디오플레이어 만들기 2 (0) 2009/02/14

Posted by Real_G