千 Line 으로 비디오플레이어 만들기 3

멀티미디어 : 2009/02/14 23:42

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.

Tutorial 03 : Play Sound

tutorial03.c

Audio

이제 우리는 Sound 재생을 해야할 차례 입니다.
SDL은 사운드 출력 메소드도 제공해줍니다. SDL_OpenAudio()함수로 사운드 장치를 이용할 수 있습니다. 이 함수는 SDL_AudioSpec struct 를 아큐먼트로 받는데 여기에는 우리가 출력할 모든 오디오 정보를 담고 있습니다.

이것을 어떻게 쓰는지 보기전에 컴퓨터가 오디오를 어떻게 제어하는지를 알아볼까요?
디지탈 오디오는 샘플들의 긴 스트림으로 이루어져 있습니다. 각각의 샘플은 audio waveform 의 값을 나타냅니다.
Sound는 sample rate들의 기록입니다. 간단히 말하면 각각의 샘플을 얼마나 빨리 재생할지, 그리고 초당 샘플수가 측정된 것 입니다. 예를들면 sample rates 가 22,050 그리고 44,100 samples per second 같은것들은 라디오나 CD에서 쓰이는 rate들 입니다. 추가적으로 대부분의 오디오들은 surround나 stereo를 위해 한개 이상의 채널을 갖을수 있습니다. stereo sample 같은 경우에는 한번에 2개의 sample이 재생될것입니다. 우리가 비디오파일에서 데이타를 받을때, 얼마나 많은 샘플을 얻게될지 모릅니다. 하지만 ffmpeg는 부분적인 샘플을 주지 않을것입니다. - 이것은 stereo sample을 쪼갤수 없다는 것을 의미하기도 합니다.

SDL의 오디오 재생 메소드 : 여러분은 여러분의 오디오 옵션을 지정해야 합니다. : sample rate(SDL structure에서 frequenct를 freq 라고 부릅니다.), 채널 수, 콜백함수, 사용자데이타 들을 세팅합니다.
오디오를 플레이하기 시작할때 SDL은 콜백 함수를 연속적으로 호출하고 오디오 버퍼와 바이트 수를 채울것을 요청할것입니다. 우리는 SDL_AudioSpec struct 의 정보를 입력한 다음에 SDL_OpenAudio()함수를 호출합니다. 그러면 오디오 디바이스를 열고 AudioSpec struct 을 돌려줄것입니다. 여기서 나온 spec들이 실제 사용하게 될 것입니다.

Setting Up the Audio

우리는 지금 정신이 몽롱 합니다.
우리는 아직 오디오 스트림에 대한 아무런 정보도 없거든요.
자~ 소스코드로 가서 우리가 비디오 스트림을 찾았던 부분에서 오디오 스트림도 찾아봅시다.

view plain copy to clipboard print ?

```
01.
02.  videoStream=-1;
03.  audioStream=-1;
04.  for(i=0; i < pFormatCtx->nb_streams; i++) {
05.    if(pFormatCtx->streams[i]->codec->codec_type==CODEC_TYPE_VIDEO
06.       &&
07.          videoStream < 0) {
08.      videoStream=i;
09.    }
10.    if(pFormatCtx->streams[i]->codec->codec_type==CODEC_TYPE_AUDIO &&
11.       audioStream < 0) {
12.      audioStream=i;
13.    }
14.  }
15.  if(videoStream==-1)
16.    return -1;
17.  if(audioStream==-1)
18.    return -1;
```

stream의 AVCodecContext 에서 우리가 원하는 모든 정보를 얻을 수 있습니다.

그냥 비디오 스트림에서 우리가 했던대로 하면 됩니다. 참~ 쉽죠잉~?

```
view plain  copy to clipboard  print  ?
01.  AVCodecContext *aCodecCtx;
02.
03.  aCodecCtx=pFormatCtx->streams[audioStream]->codec;
```

AVCodecContext
다음 codec context에 우리가 오디오 set up에 필요한 모든 정보가 다 들어있습니다.

```
view plain  copy to clipboard  print  ?
01.  wanted_spec.freq = aCodecCtx->sample_rate;
02.  wanted_spec.format = AUDIO_S16SYS;
03.  wanted_spec.channels = aCodecCtx->channels;
04.  wanted_spec.silence = 0;
05.  wanted_spec.samples = SDL_AUDIO_BUFFER_SIZE;
06.  wanted_spec.callback = audio_callback;
07.  wanted_spec.userdata = aCodecCtx;
08.
09.  if(SDL_OpenAudio(&wanted_spec, &spec) < 0) {
10.    fprintf(stderr, "SDL_OpenAudio: %s\n", SDL_GetError());
11.    return -1;
12.  }
```

SDL_OpenAudio()
다음을 볼까요??

- freq : 앞에서 설명한바와 같이 sample rate 입니다.
- format : SDL에 주어야할 포맷입니다. S16SYS 에서 S는 "signed"를 의미하고 16은 각각의 샘플이 16bit 이라는것을 의미합니다. 그리고 SYS는 시스템의 endian-order를 나타냅니다. avcodec_decode_audio2 를 사용하면 오디오 in 의 포맷을 우리에게 알려줍니다.

- channels : 오디오 체널의 수.
- silence : silence를 지정한 값. 오디오가 signed 이기 때문에 0이 주로 쓰인다.
- samples : SDL이 추가 오디오 데이타를 요청할 때를 대비한 오디오 버퍼의 사이즈. 512~8192의 값을 사용하는것이 좋고 ffplay에서는 1024를 사용합니다.
- callback : 여기서 callback 함수는 건너뜁니다. 다음에 callback함수에 대해서 알아보도록 합니다.
- userdata : SDL은 callback에게 어떤 user data를 가리키고 있는 void pointer 를 돌려줍니다.

SDL_OpenAudio 으로 오디오를 열어봅시다.

여러분은 여러분의 오디오 데이타를 혼합하고 그것을 오디오 스트림으로 보내는 콜백 함수를 만들어야 합니다.. 그 다음엔, 원하는 오디오 포맷과 속도를 선택하고, 오디오 장치를 엽니다. SDL_PauseAudio(0) 를 호출할때까지 오디오는 실제로 재생되지 않습니다. 이 함수는 여러분의 콜백 함수가 실행되기 전에, (필요하다면) 다른 오디오 초기화작업을 수행할 수 있도록 합니다. 사운드 출력을 마친 다음, SDL_CloseAudio() 함수를 사용해 오디오 출력을 닫아야 합니다.

팁:
만약 여러분의 애플리케이션이 다른 오디오 포맷을 처리할 수 있다면, 두번째 SDL_AudioSpec 포인터를 SDL_OpenAudio() 에 보내 실제 하드웨어 오디오 포맷을 취하도록 해야 한다. 만약 두번째 포인터를 NULL로 남겨놓는다면, 오디오 데이타는 실행시에 하드웨어 오디오 포맷으로 변환될 것이다.

전에 했던 tutorial을 기억하신다면 오디오 코덱을 스스로 열어야할 필요가 있다는 것을 아실것입니다.
다음처럼 하면 간단합니다.

```
view plain  copy to clipboard  print  ?
01.  AVCodec          *aCodec;
02.
03.  aCodec = avcodec_find_decoder(aCodecCtx->codec_id);
04.  if(!aCodec) {
05.    fprintf(stderr, "Unsupported codec!\n");
06.    return -1;
07.  }
08.  avcodec_open(aCodecCtx, aCodec);
```

avcodec_find_decoder(), avcodec_open()

Queues

이제 스트림에서 오디오 정보를 얻을 준비가 다 되었습니다. 하지만 이 정보로 뭘해야 할까요? 우리는 무비파일에서 계속 패킷을 얻을 것인데 같은 시간 SDL은 callback 함수를 호출할 것입니다.
이 방법은 어떤 종류의 전역 구조체를 생성하는 방법입니다. 우리는 여기에 오디오로부터 얻은 오디오 패킷을 채울수 있습니다. 그래서 우리는 패킷의 queue를 만들것인데 ffmpeg는 이것을 위해 AVPacketList 를 지원해 줍니다. 이것은 패킷을 위한 linked list 입니다. queue의 구조는 다음과 같습니다.

```
view plain  copy to clipboard  print  ?
01.  typedef struct PacketQueue {
02.    AVPacketList *first_pkt, *last_pkt;
03.    int nb_packets;
04.    int size;
05.    SDL_mutex *mutex;
06.    SDL_cond *cond;
07.  } PacketQueue;
```

AVPacketList , SDL_mutex , SDL_cond

우선 nb_packets를 지정해야 하는데 이것은 size 같은것이 아닙니다. size는 packet->size로 부터 얻은 1byte size를 참조합니다. 여러분은 여기서 mutex와 condtion variable를 선언해줘야 합니다. SDL은 audio process를 분리된 Thread처럼 실행하고 있기 때문이죠. 만약 여러분이 queue에 락을 정확히 걸지 않으면 여러분의 데이타는 엉망진창이 될것입니다. 이젠 queue 의 실행을 보겠습니다.

우선 queue를 초기화하는 함수를 만들어 보겠습니다.

```
view plain  copy to clipboard  print  ?
01.  void packet_queue_init(PacketQueue *q) {
02.    memset(q, 0, sizeof(PacketQueue));
03.    q->mutex = SDL_CreateMutex();
04.    q->cond = SDL_CreateCond();
05.  }
```

SDL_CreateMutex(), SDL_CreateCond()

이번에는 만들어진 queue에 데이터를 채우는 함수를 만들겠습니다.

```
view plain  copy to clipboard  print  ?
01. int packet_queue_put(PacketQueue *q, AVPacket *pkt) {
02.
03.   AVPacketList *pkt1;
04.   if(av_dup_packet(pkt) < 0) {
05.     return -1;
06.   }
07.   pkt1 = av_malloc(sizeof(AVPacketList));
08.   if (!pkt1)
09.     return -1;
10.   pkt1->pkt = *pkt;
11.   pkt1->next = NULL;
12.
13.
14.   SDL_LockMutex(q->mutex);
15.
16.   if (!q->last_pkt)
17.     q->first_pkt = pkt1;
18.   else
19.     q->last_pkt->next = pkt1;
20.   q->last_pkt = pkt1;
21.   q->nb_packets++;
22.   q->size += pkt1->pkt.size;
23.   SDL_CondSignal(q->cond);
24.
25.   SDL_UnlockMutex(q->mutex);
26.   return 0;
27. }
```

AVPacket , AVPacketList , av_dup_packet() , av_malloc() , AVPacketList , SDL_LockMutex() , SDL_CondSignal() , SDL_UnlockMutex()

SDL_LockMutex()를 이용해서 queue에 mutex를 잠그고 데이타를 추가합니다. 그리고 SDL_CondSignal()로 get 함수에 signal을 보냅니다. 그런 다음 mutex를 unlock 합니다.

다음이 get 함수에 대응하는 내용입니다.
어떻게 SDL_CondWait() 함수가 Block을 걸게 되는지 생각해보세요.

```
view plain  copy to clipboard  print  ?
01. int quit = 0;
02.
03. static int
04. packet_queue_get(PacketQueue * q, AVPacket * pkt, int block)
05. {
06.     AVPacketList   *pkt1;
07.
08.     int             ret;
09.
10.     SDL_LockMutex(q->mutex);
11.
12.     for (;;)
13.     {
14.
15.         if (quit)
16.         {
17.             ret = -1;
18.             break;
19.         }
20.
21.         pkt1 = q->first_pkt;
22.         if (pkt1)
23.         {
24.             q->first_pkt = pkt1->next;
25.             if (!q->first_pkt)
26.                 q->last_pkt = NULL;
27.             q->nb_packets--;
28.             q->size -= pkt1->pkt.size;
29.             *pkt = pkt1->pkt;
30.             av_free(pkt1);
31.             ret = 1;
32.             break;
33.         }
34.         else if (!block)
35.         {
36.             ret = 0;
37.             break;
38.         }
39.         else
40.         {
41.             SDL_CondWait(q->cond, q->mutex);
42.         }
43.     }
44.     SDL_UnlockMutex(q->mutex);
45.     return ret;
46. }
```

AVPacket , AVPacketList , SDL_LockMutex() , av_free() , SDL_CondWait() , SDL_UnlockMutex()

보시다시피 무한루프로 함수를 둘러쌌습니다. 그래서 우리가 블록을 얻길 원한다면 데이터를 확실히 얻을수 있을 것입니다. 무한루프가 일어나는 것을 막으려면 SDL의 SDL_CondWait() 함수를 사용하게합니다. 기본적으로 SDL_CondWait 는 제공된 mutex를 unlock하고 다른 쓰레드에서 SDL_CondSignal() 또는 SDL_CondBroadcast()를 호출한 signal을 기다렸다가 다시 lock을 걸고 진행합니다. 그러나 이것은 mutex로 잡혀있습니다. 만약 lock이 걸려있다면 queue에 어떠한 것도 넣을수 없습니다.

In Case of Fire
우리는 quit 전역변수를 갖고 있는데 이것은 프로그램 종료 signal이 발생하지 않았다는 것을 확실하게 체크해 줍니다. (SDL은 자동으로 TERM signal 같은것을 제어합니다.)
만약 그렇지 않으면 무한루프에 빠져서 우리가 kill -9로 프로그램을 죽여야 할것입니다. 만약 몇몇 blocking 함수를 종료할 필요성이 있다면 ffmpeg는 callback을 체크하고 볼수 있는 함수를 제공합니다.: url_set_interrupt_cb

```
view plain  copy to clipboard  print  ?
01. int decode_interrupt_cb(void) {
02.   return quit;
03. }
04. ...
05. main() {
06. ...
07.   url_set_interrupt_cb(decode_interrupt_cb);
08. ...
09.   SDL_PollEvent(&event);
10.   switch(event.type) {
11.   case SDL_QUIT:
12.     quit = 1;
13. ...
```

이것은 ffmpeg 함수만을 지원합니다. 당연히 SDL의 함수는 안됩니다.
우리는 quit flag를 반드시 1로 세팅해야 합니다.

Feeding Packets

queue 셋업이 하나 남았군요

```
view plain  copy to clipboard  print  ?
01. PacketQueue audioq;
02. main() {
03. ...
04.   avcodec_open(aCodecCtx, aCodec);
05.
06.   packet_queue_init(&audioq);
07.   SDL_PauseAudio(0);
```

avcodec_open() , SDL_PauseAudio()

SDL_PauseAudio() 로 오디오 디바이스를 구동합니다. 데이터를 얻지 못하면 아무소리도 안납니다.

우리는 queue 셋업을 해놨기 때문에 이제 packet을 feeding 할 준비가 다 되었습니다.
이제 패킷을 읽어오는 루프를 실행 합니다.

```
view plain  copy to clipboard  print  ?
01. while(av_read_frame(pFormatCtx, &packet)>=0) {
02.
03.   if(packet.stream_index==videoStream) {
04.
05.     ....
06.     }
07.   } else if(packet.stream_index==audioStream) {
```

```
    packet_queue_put(&audioq, &packet);
  } else {
    av_free_packet(&packet);
  }
```

av_read_frame(), av_free_packet()
패킷을 큐에 넣은 다음에 메모리 해제를 하지 않습니다. 다음에 디코드 할때 해제할 것입니다.

Fetching Packets

이제queue의 패킷에 붙일 audio_callback 함수를 만들어 봅시다.
callback 함수는 void callback(void *userdata, Uint8 *stream, int len)의 형태여야 합니다. , userdata는 물론 우리가 SDL에 준 포인터 입니다. 스트림은 우리가 쓰고있는 오디오 데이타의 버퍼 이고, len은 버퍼의 싸이즈 입니다.
다음 코드를 보시죠

view plain copy to clipboard print ?

```
void audio_callback(void *userdata, Uint8 *stream, int len) {

  AVCodecContext *aCodecCtx = (AVCodecContext *)userdata;
  int len1, audio_size;

  static uint8_t audio_buf[(AVCODEC_MAX_AUDIO_FRAME_SIZE * 3) / 2];
  static unsigned int audio_buf_size = 0;
  static unsigned int audio_buf_index = 0;

  while(len > 0) {
    if(audio_buf_index >= audio_buf_size) {

      audio_size = audio_decode_frame(aCodecCtx, audio_buf,
                                      sizeof(audio_buf));
      if(audio_size < 0) {

    audio_buf_size = 1024;
    memset(audio_buf, 0, audio_buf_size);
      } else {
    audio_buf_size = audio_size;
      }
      audio_buf_index = 0;
    }
    len1 = audio_buf_size - audio_buf_index;
    if(len1 > len)
      len1 = len;
    memcpy(stream, (uint8_t *)audio_buf + audio_buf_index, len1);
    len -= len1;
    stream += len1;
    audio_buf_index += len1;
  }
}
```

AVCodecContext

이것은 기본적인 다른 함수로부터 데이터를 끌어오는 간단한 루프입니다. 우리는 쓰고, audio_decode_frame()를 실행하고 결과를 하나의 중계 버퍼에 저장합니다. 그리고 len만큼의 byte를 stream에 쓰도록 시도합니다. 그리고 아직 충분하지 못하면 데이터를 더 가져오고 데이터가 남았다면 기다렸다가 저장합니다. audio_buf의 사이즈는 가장 큰 audio frame의 크기에 1.5배로 설정합니다. ffmpeg는 우리에게 괜찬은 쿠션을 제공해 줍니다.

Finally Decoding the Audio
다음은 디코더 소스 입니다.

view plain copy to clipboard print ?

```
int audio_decode_frame(AVCodecContext *aCodecCtx, uint8_t *audio_buf,
                       int buf_size) {

  static AVPacket pkt;
  static uint8_t *audio_pkt_data = NULL;
  static int audio_pkt_size = 0;

  int len1, data_size;

  for(;;) {
```

```
    while(audio_pkt_size > 0) {
      data_size = buf_size;
      len1 = avcodec_decode_audio2(aCodecCtx, (int16_t *)audio_buf, &data_size,
                  audio_pkt_data, audio_pkt_size);
      if(len1 < 0) {

    audio_pkt_size = 0;
    break;
      }
      audio_pkt_data += len1;
      audio_pkt_size -= len1;
      if(data_size <= 0) {

    continue;
      }

      return data_size;
    }
    if(pkt.data)
      av_free_packet(&pkt);

    if(quit) {
      return -1;
    }

    if(packet_queue_get(&audioq, &pkt, 1) < 0) {
      return -1;
    }
    audio_pkt_data = pkt.data;
    audio_pkt_size = pkt.size;
  }
}
```

AVCodecContext , AVPacket , avcodec_decode_audio2() , av_free_packet()

모든 처리는 함수의 끝 부분에서 시작합니다. call packet_queue_get()를 호출한 부분. queue에서 패킷을 집어와서 그 정보를 저장합니다. 그리고 한 패킷으로 avcodec_decode_audio2()를 호출합니다. 한 패킷은 한 프레임 이상을 갖고 있을 수도 있습니다. 그래서 여러분은 아마 패킷으로부터 모든 데이타를 얻기 위해 이것을 여러번 호출해야 할지도 모릅니다.
또한 여러분은 SDL이 8bit int buffer를 주기 때문에 audio_buf를 casting 해야 하는 것을 기억하셔야 합니다. 그리고 ffmpeg는 우리에게 16bit int buffer를 줍니다.
또한 len1과 data_size 가 다르다는 것도 알고 있어야 합니다. len1 은 우리가 패킷을 얼마나 사용했는지를 나타내는 것이고 data_size는 raw data가 반환된 양을 나타냅니다.

우리가 데이터를 받았을 때, 우리가 여전히 queue로 부터 데이터를 더 얻어야 한다거나 작업이 완료 되었거나 즉시 리턴합니다. 우리가 처리할 패킷을 가지고 있다면 이것을 다음에 저장합니다. 만약 패킷 처리가 끝났다면 paket을 해제 합니다.
이제 queue로 루프를 돌면서 읽은 것을 모두 옮겼습니다. audio_callback 함수에 의해 읽어진 데이타를 SDL이 여러분의 사운드 카드로 쏠 것입니다.
이제 컴파일을 해볼까요?

```
gcc -o tutorial03 tutorial03.c -lavutil -lavformat -lavcodec -lz -lm ₩`sdl-config --cfl
```

얼라리~ 여전히 비디오는 후딱 지나가는군요. 하지만 오디오는 제대로 재생됩니다.
왜그럴까요?
그것은 오디오 정보는 Sample rate를 갖고 있기 때문입니다. - 우리는 가능한 빨리 오디오 정보를 뽑아내지만 오디오는 sample rate에 따라 천천히 스트림으로 부터 재생 됩니다.

우리는 이제 비디오와 오디오의 싱크를 맞추면 됩니다. 그전에 우리는 프로그램을 약간 손봐야 할 필요가 있습니다.

오디오를 큐에 넣는 메소드와 플레이 하는 것을 다른 쓰레드에서 하도록 분리해야 합니다. 이렇게 해야 유지보수 하기 쉽고, 더 모듈화 되기 때문입니다.
비디오와 오디오를 싱크하기 전에 우리의 코드를 더 다루기 쉽게 만들어야 합니다.
다음 Tutorial 에서는 쓰레드를 만들어 보겠습니다.!

千 Line 으로 비디오플레이어 만들기 3 조회수 771

'멀티미디어' 카테고리의 다른 글

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
千 Line 으로 비디오플레이어 만들기 1 (0) 2009/02/14

Posted by Real_G