

검색

## 千 Line 으로 비디오플레이어 만들기 1

멀티미디어 : 2009/02/14 22:39

### [영어 3개월이면 말할수있다](#)

아는 단어로 대충 끼워맞춰 영어회화 하십니까? 과연 맞는 방법일까요?

[www.nswenglish.com](http://www.nswenglish.com)

AdChoices

### [GoldWave Audio Software](#)

Record, Edit, Restore, Convert... Voice, Music, MP3. Free download!

[www.goldwave.com](http://www.goldwave.com)

AdChoices

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 01: Making Screenscaps

 tutorial01.c

첫번째 Tutorial에서 만들어 볼 것은 Video파일에서 처음 다섯 프레임을 읽어 ppm 이미지 파일로 저장 하는 것입니다.

이 예제를 실행하기 위해서는 libavformat 과 libavcodec 이 설치되어 있어야 합니다.

테스트 하실때에는 컴파일한 파일의 첫번째 아규먼트로 테스트할 비디오 파일명을 입력하면 됩니다.

그럼 시작해볼까요?

Movie 파일은 몇가지 기본적인 요소로 구성되어 있습니다.

첫째, Container 라고 불리는 파일 자체, 그리고 파일정보가 어디에 있는지를 결정하는 Container의 타입입니다. AVI나 Quicktime 같은 것을 Container라고 부르는 것이죠.

둘째로는 Stream의 다발 입니다. 예를 들면, 보통 오디오 스트림, 비디오 스트림 같은 것입니다. (Stream은 '시간에 따른 연속적 데이터 요소'를 흔히 일컫는 말입니다.)

Stream의 요소는 Frame 이라고 부르지요.

각 Stream은 각기 다른 Codec에 의해 인코딩되어 있습니다.

Codec은 실제 데이터를 어떻게 CODEd, DECODEd 할것인지를 정의해놓은 것이라서 CODEC이라는 이름이 붙게 되었습니다.


Codec의 예로는 DivX, MP3 같은것이 있어요

Packet 은 Stream을 읽은 것입니다.

Packet 은 RAW Frame으로 decode될 비트 데이터를 갖고 있는 데이터의 조각입니다.

바로 우리의 프로그램을 위해서 우리가 다루어야 할 것이기도 하지요.

우리의 목적은 완전한 Frame들을 갖고 있는 패킷이거나 오디오의 경우 multiple Frame 들 다루어야 하는 것입니다.

????? 

다음은 기초적인 오디오, 비디오를 다루는 내용입니다.

```
10 OPEN video_stream FROM video.avi
20 READ packet FROM video_stream INTO frame
30 IF frame NOT COMPLETE GOTO 20
40 DO SOMETHING WITH frame
50 GOTO 20
```

ffmpeg를 이용해 멀티미디어파일을 다루는 것은 이 프로그램에서 처럼 매우 간단합니다. 몇몇 프로그램에서는 월 하기가 매우 복잡하겠지만.... 그래서 이 Tutorial에서는 파일을 열고, 파일에 들어있는 비디오 스트림을 읽어서 뭔가를 해보도록 하겠습니다.(PPM 파일에 Frame을 쓰는것)

### Opening the File

첫번째 부분에서 파일을 어떻게 여는지 보도록 합시다.

ffmpeg에서 첫째로 할일을 라이브러리를 초기화 하는 것 입니다.

(몇몇 시스템에서는 <ffmpeg/avformat.h> , <ffmpeg/avcodec.h> 라고 써야될 수도 있습니다.)

```
view plain copy to clipboard print ?
01. #include <avcodec.h>
02. #include <avformat.h>
03. ...
04. int main(int argc, char *argv[]) {
05. av_register_all();
```

모든 파일 포맷과 코덱을 라이브러리와 함께 등록하고, 이것들은 대응되는 포맷이나 코덱이 열릴때 자동으로 사용됩니다.

av\_register\_all() 는 딱 한번만 호출해야 하기때문에 우리는 main() 함수 안에 쓰기로 합니다.

자~ 이제 우리는 실제 파일을 열어봅니다.

```
view plain copy to clipboard print ?
01. AVFormatContext *pFormatCtx;
02.
03.
04. if(av_open_input_file(&pFormatCtx, argv[1], NULL, 0, NULL)!=0)
05. return -1;
```

av\_open\_input\_file()

첫번째 아규먼트에서 파일명을 얻습니다. 이 함수는 파일헤더를 읽어오고 파일포맷에 대한 정보를 우리가 지정한 AVFormatContext structure에 저장합니다. 마지막 세개의 아규먼트들은 파일포맷, 버퍼크기, 포맷옵션을 지정하게 되어있지만 그냥 NULL이나 0을 세팅합니다. libavformat이 자동으로 찾거든요.

이 함수는 단지 헤더를 찾는 일만 합니다. 그래서 우리는 파일에서 스트림 정보를 찾아야 해요~

```
view plain copy to clipboard print ?
01.
02. if(av_find_stream_info(pFormatCtx)<0)
03.     return -1;
```

av\_find\_stream\_info()

이 함수는 pFormatCtx->streams 에 고유 정보를 넣어줍니다.

다음은 다루기 쉬운 디버깅함수를 소개합니다.

```
view plain copy to clipboard print ?
01.
02. dump_format(pFormatCtx, 0, argv[1], 0);
```

pFormatCtx->nb\_streams는 파일에 있는 스트림의 개수 이고

pFormatCtx->streams는 여기 저장된 각 스트림의 스트림 데이터가 들어있습니다.

우리는 이것을 통해서 video stream을 찾아야 해요.

```
view plain copy to clipboard print ?
01. int i;
02. AVCodecContext *pCodecCtx;
03.
04.
05. videoStream=-1;
06. for(i=0; i<pFormatCtx->nb_streams; i++)
07.     if(pFormatCtx->streams[i]->codec->codec_type==CODEC_TYPE_VIDEO) {
08.         videoStream=i;
09.         break;
10.     }
11. if(videoStream==-1)
12.     return -1;
13.
14.
15. pCodecCtx=pFormatCtx->streams[videoStream]->codec;
```

AVCodecContext

코덱에 관한 스트림 정보를 codec context라고 부릅니다. 이것은 코덱에 관한 모든 정보를 갖고 있습니다.

우리는 그 스트림을 사용하고 있고 그것을 가리키는 포인터도 갖고 있습니다.

하지만 여전히 우리는 실제 코덱을 찾고 그것을 열어야 합니다.

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

```

08.     }
09.
10.     if(avcodec_open(pCodecCtx, pCodec)<0)
11.         return -1;

```

avcodec\_find\_decoder(), avcodec\_open()

## Storing the Data

이젠 frame을 저장할 공간이 필요합니다.

```

view plain copy to clipboard print ?
01.     AVFrame *pFrame;
02.
03.
04.     pFrame=avcodec_alloc_frame();

```

AVFrame, avcodec\_alloc\_frame()

우리는 PPM 파일로 출력해야 하고, 24-bit RGB로 저장되어야 하기 때문에, native format 에서 RGB로 프레임을 변환해야 합니다. 이 작업은 ffmpeg가 해줄겁니다. ^\_^

대부분의 프로젝트에서는 초기 프레임을 특정 포맷으로 변환해야 할것입니다.

자~ 이제 변환된 프레임으로 프레임을 할당해 봅시다.

```

view plain copy to clipboard print ?
01.
02.     pFrameRGB=avcodec_alloc_frame();
03.     if(pFrameRGB==NULL)
04.         return -1;

```

프레임을 할당했더라도 우리는 변환시에 Raw Data를 저장할 장소가 필요합니다. 이때 필요한 싸이즈를 얻고 수동으로 장소를 할당하기 위해 avpicture\_get\_size 를 사용합니다.

```

view plain copy to clipboard print ?
01.     uint8_t *buffer;
02.     int numBytes;
03.
04.     numBytes=avpicture_get_size(PIX_FMT_RGB24, pCodecCtx->width,
05.                                pCodecCtx->height);
06.     buffer=(uint8_t *)av_malloc(numBytes*sizeof(uint8_t));

```

av\_malloc() 은 ffmpeg 에서 쓰는 malloc 인데 이건 malloc을 간단히 wrapper 한것이라 기능역시 유사합니다.

av\_malloc()은 메모리 릭 이나 double freeing, 또는 다른 malloc 문제 로부터 여러분을 보호해 줄수는 없습니다.

이제 avpicture\_fill()을 이용해서 Frame과 우리가 새로 할당한 버퍼를 연결합니다.

AVPicture cast : AVPicture struct 는 AVFrame struct의 작은 덩어리입니다. - AVFrame struct의 시작부분은 AVPicture struct 와 동일합니다.

```

view plain copy to clipboard print ?
01.
02.
03.
04.     avpicture_fill((AVPicture *)pFrameRGB, buffer, PIX_FMT_RGB24,
05.                   pCodecCtx->width, pCodecCtx->height);

```

avpicture\_fill(), AVFrame, AVPicture

드디어! stream을 읽을 준비가 다 되었군요~!

### Reading the Data

패킷에서 읽은것을 전체 비디오스트림에 걸쳐 읽어 이것을 디코딩해서 프레임에 넣으면 하나의 프레임이 완성되고 그것을 변환하고 저장할 것입니다.

```
view plain copy to clipboard print ?
01.  int frameFinished;
02.  AVPacket packet;
03.
04.  i=0;
05.  while(av_read_frame(pFormatCtx, &packet)>=0) {
06.
07.      if(packet.stream_index==videoStream) {
08.
09.          avcodec_decode_video(pCodecCtx, pFrame, &frameFinished,
10.                                packet.data, packet.size);
11.
12.
13.          if(frameFinished) {
14.
15.              img_convert((AVPicture *)pFrameRGB, PIX_FMT_RGB24,
16.                          (AVPicture*)pFrame, pCodecCtx->pix_fmt,
17.                          pCodecCtx->width, pCodecCtx->height);
18.
19.
20.              if(++i<=5)
21.                  SaveFrame(pFrameRGB, pCodecCtx->width,
22.                             pCodecCtx->height, i);
23.          }
24.      }
25.
26.
27.      av_free_packet(&packet);
28.  }
```

이 반복문은 간단합니다.

av\_read\_frame()는 패킷에서 읽어와서 AVPacket struct 에 저장합니다.

우리는 단지 packet structure만 할당하면 됩니다. 그러면 ffmpeg가 packet.data에 의해 지정된 내부데이터를 할당해 줍니다.

이것은 다음에 av\_free\_packet()함수로 메모리 해제를 할수 있습니다.

avcodec\_decode\_video()는 패킷을 프레임으로 변환합니다. 그러나 디코딩 된 프레임을 위한 모든 정보를 다 갖고 있지는 못합니다. 그래서 avcodec\_decode\_video() 함수는 다음 프레임으로 갈때 frameFinished 을 설정해줍니다.

드디어!! img\_convert() 함수를 사용하여 native format(pCodecCtx->pix\_fmt)를 RGB로 변환 합니다.

여러분은 AVFrame pointer를 AVPicture pointer로 할당할 수 있다는 것을 기억하세요.

우리는 이제 frame, width, height 정보를 SaveFrame 함수로 보냅니다.

SaveFrame 함수는 RGB 정보를 PPM 포맷의 파일에 기록합니다.

이제 개략적인 PPM 포맷을 설계해 봅시다.

```
view plain copy to clipboard print ?
01.  void SaveFrame(AVFrame *pFrame, int width, int height, int iFrame) {
02.      FILE *pFile;
03.      char szFilename[32];
04.      int y;
```

```

05.
06.
07.     sprintf(szFilename, "frame%d.ppm", iFrame);
08.     pFile=fopen(szFilename, "wb");
09.     if(pFile==NULL)
10.         return;
11.
12.
13.     fprintf(pFile, "P6\n%d %d\n255\n", width, height);
14.
15.
16.     for(y=0; y<height; y++)
17.         fwrite(pFrame->data[0]+y*pFrame->linesize[0], 1, width*3, pFile);
18.
19.
20.     fclose(pFile);
21. }

```

일반적 파일 열기처럼 파일을 열고 RGB 데이터를 기록합니다. 우리는 한번에 한줄씩 파일에 쓰기로 합니다.

PPM 파일은 RGB 정보를 긴 문자열로 늘어놓은 간단한 파일입니다. 만일 여러분이 HTML 색상코드를 아신다면, 이것은 끝에서 끝까지, 빨간 화면이라면 #ff0000#ff0000.... 이런식으로 각각의 픽셀의 컬러값을 늘어놓은 것과 비슷하다고 보시면 됩니다. (이 것들은 binary로 저장되고 연속되지 않습니다.) Header는 이미지의 넓이, 높이와 RGB값의 최대 싸이즈를 가리킵니다.

이제 main()함수로 돌아가볼까요?

일단 우리 비디오 스트림으로 부터 읽기를 완료했습니다.

그리고 메모리를 해제해 줍니다.

```

view plain copy to clipboard print ?
01.
02.     av_free(buffer);
03.     av_free(pFrameRGB);
04.
05.
06.     av_free(pFrame);
07.
08.
09.     avcodec_close(pCodecCtx);
10.
11.
12.     av_close_input_file(pFormatCtx);
13.
14.     return 0;

```

여러분은 avcode\_alloc\_frame과 av\_malloc으로 할당했었던 메모리를 해제하기 위해서 av\_free 를 써야한다고 말씀하시겠지요?

이게 바로 그 코드 입니다.

이제 Linux에서 다음을 실행시켜 봅시다.

```
gcc -o tutorial01 tutorial01.c -lavutil -lavformat -lavcodec -lz -lavutil -lm
```

ffmpeg가 구버전이라면 -lavutil 은 빼고

```
gcc -o tutorial01 tutorial01.c -lavformat -lavcodec -lz -lm
```

이렇게 실행시켜 보세요

대부분의 이미지 프로그램은 PPM 파일을 열수 있을 겁니다.

몇개의 테스트용 동영상 파일로 한번 테스트 해보시기 바랍니다.



千 Line 으로 비디오플레이어 만들기 1 조회수 1,202

9 아이유 열애설 상대 누구? '주변사람이 만류할 정...  
12 착한남자, 가장 잔인했던 1초 문채원의 키스가 두...

크리에이티브 커먼즈 라이선스



이 저작물은 크리에이티브 커먼즈 코리아 저작자표시-비영리-변경금지 2.0 대한민국 라이선스에 따라 이용하실 수 있습니다.

'멀티미디어' 카테고리의 다른 글

千 Line 으로 비디오플레이어 만들기 2 (0)	2009/02/14
<u>千 Line 으로 비디오플레이어 만들기 1 (0)</u>	2009/02/14
千 Line 으로 비디오플레이어 만들기 서문 (0)	2009/02/04
멀티처리 텀 프로젝트 (6)	2008/06/24

Posted by Real\_G

이전 1 ... 645 646 647 648 649 650 651 652 653 ... 1603 다음