Transcoding with FFmpeg

Coding Technology Document

- FFMPEG를 활용한 Visual Studio 내 Transcoding–

2017.01.10

작성자: 김문원, 황진하

목차

1. 개요
2. Windows compile
3. Header Include
4. Library 초기화
5. Media 파일 열기
6. Stream Demuxing
7. Decoding
8. Encoding
9. 결과
10. 시스템 환경
11. 부록
12. 개요

본 문서에서는 FFMPEG의 라이브러리를 이용해 Window 환경에서 Media file의 Decoding후, 다시 Encoding을 하는 Transcoding 과정을 다룬다. 과정은 소스코드를 중심으로 설명하며, 자세한 정보와 오류보고는 하단의 부록에 포함된다.

1. Windows Compile

- Windows로 포팅한 GNU 소프트웨어 도구 모음인 MinGW를 설치한다.

- yasm 어셈블러를 windows용으로 다운로드 후 MinGW설치 경로의 bin폴더에 “yasm.exe”로 이름 변경 후 저장

- MSYS (GNU 유틸리티) 다운로드 후 위와 같은 경로에 “pr.exe” 파일을 저장

- “pr.exe”파일이 없으면 configure 시에 오류가 발생한다.

- x264 소스코드를 다운로드한다. <http://www.videolan.org/developers/x264.html>

- ffmpeg 공식홈페이지로 가서 소스코드파일을 다운로드 한다.

- 다운받은 소스코드 폴더를 MinGW/msys/1.0 폴더에 저장한다.

- MinGW \ msys \ 1.0 \ msys.bat 실행

- 루트폴더가 MinGW \ msys \ 1.0 을 기준으로 생각한다.

- x264에 대한 configure를 실행 명령어 “./configure –enable-shared”

- make, make install 명령을 순서대로 수행하여 x264를 빌드한 후 설치한다

- 루트폴더에 local이라는 폴더가 생겼으면 편의를 위해 폴더이름을 x264로 변경한다

- “./configure –enable-shared –enable-static –target-os=mingw32 –enable-gpl –enable-libx264 –extra-cflags=”-I/x264/include” –extra-ldflags=”-L/x264/lib” –enable-gpl”

- 마지막에 경고가 뜨지만 pkg-config가 없다는 것이므로 최신 라이브러리에선 없는게 맞다.

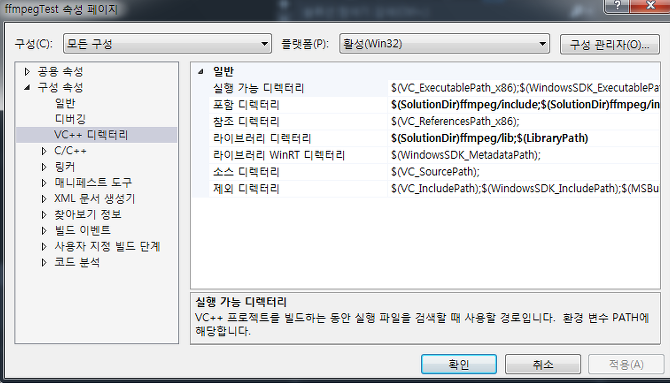
- MinGW\include\unistd.h 파일에 “\_cdecl”을 “\_\_cdecl”로 수정한다. 최신버전을 받았다면 수정하지 않아도 될 수 있다.  
- ffmpeg폴더 위치로 cd명령어를 통해 접근한 후 make와 make install을 해준다. 이 작업에서 꽤 많은 시간이 소요될 수 있다.

- 설치가 완료되면 MinGW\msys\1.0\local 폴더에 파일들이 정리되어 있다.

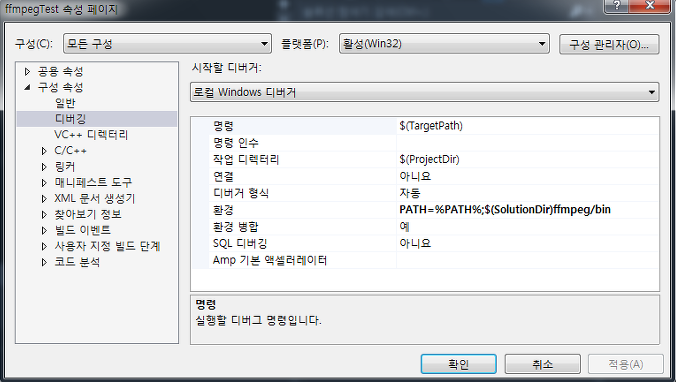
- 이후 외부 lib인 x264를 실행파일 폴더에 복사하여 사용하면 된다.

1. Visual Studio 환경 설정 방법

* 위 컴파일 과정에서 만들어진 bin,include,lib 폴더를 개발 솔류션 폴더 아래에 ffmpeg폴더로 생성하여 이동한다.
* 모든 구성 속성에 대하여 디렉터리 및 라이브러리 디렉터리에 각각 include폴더와 lib폴더를 입력한다.



* Ffmpeg 실행파일 경로 설정을 해준다.



* Wind32 응용프로그램을 만들어서 간단한 테스트를 진행해본다.

av\_log(NULL, AV\_LOG\_INFO,”TEST!!”);

1. Header Include

* FFMPEG를 사용하기 위해, 필요한 헤더 파일들을 포함한다.

#include <libavformat/avformat.h> // Basic Format manage header

#include <libavcodec/avcodec.h> // For using codec header

#include <libavutil/avutil.h> // AV Convenience header

#include <libavutil\imgutils.h> // Img Convenience header

#include <libswscale/swscale.h> // External API header

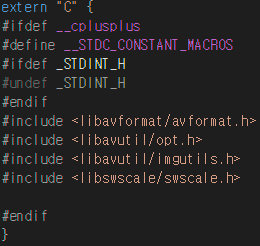
#include <libavformat/avio.h> // Input & output header

#include <libavutil/opt.h> // option header

#include <libavutil/mathematics.h> // math header

#include <libswresample/swresample.h> // Resample header

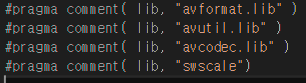
* Library 초기화 하기
* FFMPEG을 사용하기 전에, 지원하는 모든 Component를 사용 할 수 있게 먼저 등록한다.
* av\_register\_all();
* C형태로 라이브러리가 제작되었기 때문에 헤더들을 include할 때 다음과 같이 입력한다.



* 대부분의 예제코드들이 과거의 라이브러리로 만들어졌기 때문에 deprecated 오류가 생긴다. 원래 경고로 알려주지만 VS2013이후 버전에서는 오류로 취급한다. 다음과 같은 구문을 넣어 오류를 무시해준다.



* 라이브러리가 수많은 패치를 거쳐오면서 함수 이름이 변경되거나 헤더위치가 옮겨지거나해서 중복되서 정의되어있는 경우도 있다. 그래서 외부참조 오류가 날 경우 다음과같이 처리해준다. 직접 링크를 해주는 방법이다.



1. Media 파일 열기

[Input File]

* Media 파일을 다루기 위해 먼저, 파일을 Open 한다.
* avformat\_open\_input(AVFormatContext \*\*ps, const char \*filename, AVInputFormat \*fmt, AVdictionary \*\*options);
* AVFormatContext 파라미터는 해당 Media의 정보를 담은 구조체이다.
* Filename 파라미터는 Open할 파일의 이름이다.
* fmt 파라미터는 강제로 InputFormat 을 지정할 경우 사용한다. (보통 NULL)
* options 파라미터는 Demuxer의 추가 옵션을 설정할 때 사용한다. (보통 NULL)

[Output File]

* Media Data를 저장하기 위한, File과 OutputFormat을 설정한다.
* Transcoding 한 Media File을 저장하기 위한 파일 경로를 지정한다.

Ex) const char \*outputfile = "output.mkv";

* OutFile의 ContainerFormat을 정하기 위해, avformat\_alloc\_output\_context2 함수를 이용한다

Ex) avformat\_alloc\_output\_context2(&ofmt\_ctx, NULL, NULL, outputfile);

위 함수는 outputfile의 확장자를 기준으로 Format\_context를 Allocate 한다.

이 문서에서는 container format을 mp4로 설정한다.

avformat\_alloc\_output\_context2(&ofmt\_ctx, NULL, “mp4”, outputfile);

(OutputFormat을 직접 지정하거나, FormatName을 지정 할 수 있다.)

* AVOutputFormat \*ofmt을 allocate한 ofmt\_ctx의 OutputFormat으로 초기화 시킨다.

Ex) ofmt = ofmt\_ctx->oformat;

1. Stream Demuxing

* 대부분의 Media는 Video Stream 하나와 Audio Stream 하나로 구성 되어있으며, 보통 Video는 0번 Stream\_Index를 가지고, Audio는 1번 Stream\_Index를 가진다.

하지만, 복수의 Stream을 가지는 Media도 있을 수 있기에, 다음의 함수를 사용한다.

* nVSI = av\_find\_best\_stream(f\_ctx, AVMEDIA\_TYPE\_VIDEO, -1, -1, NULL, 0); 사용한다.

위 함수는 현재 FormatContext에서 다음 Stream의 Index를 반환하는 함수이다.

찾은 Stream Index를 이용해, 다음의 변수들을 초기화 한다.

AVCodecContext \*pVCtx = f\_ctx->streams[nVSI]->codec; // 비디오 코덱의 Context

AVCodecContext \*pACtx = f\_ctx->streams[nASI]->codec; // 오디오 코덱의 Context

AVFrame \*pVFrame = avcodec\_alloc\_frame(); // 비디오 Frame (Allocation)

AVFrame \*pAFrame = avcodec\_alloc\_frame(); // 오디오 Frame (Allocation)

int bGotPicture = 0; // Data 획득 구분

int bGotSound = 0;

Media file의 Video Stream 과 Audio Stream을 구분해서 사용할수 있는 것을 확인 할 수 있다.

[Stream Copy]

* Input\_FormatContext의 Stream의 CodecContext를 Output\_FormatContext의 Stream의 CodecContext에 Copy한다.
* Input 할 Stream을 초기화하고 필요한 Stream Data 가져온다

AVStream \*in\_stream = ifmt\_ctx->streams[i]; ( i=0 Video Stream, i=1 Audio Stream)

AVCodecContext\* in\_codec\_ctx = in\_stream->codec;

* Output 할 Stream을 선언하고, Output\_FormatContext에 스트림을 추가하기 위해, avformat\_new\_stream 함수를 이용한다.

Ex) AVStream \*out\_stream = avformat\_new\_stream(ofmt\_ctx, in\_stream->codec->codec);

* Codec\_Context를 복사하기 위해 avcodec\_copy\_context을 이용한다.

Ex) ret = avcodec\_copy\_context(out\_stream->codec, c);

* 저장하고자 하는 File 형식에 맞게 CodexContext를 설정하기 위해, codec\_tag를 0으로 초기화 한다.
* 구현된 코드는 다음과 같다.

for (int i = 0; i < ifmt\_ctx->nb\_streams; i++) {

/\*Video Context Copy\*/

AVStream \*in\_stream = ifmt\_ctx->streams[i];

AVCodecContext\* in\_codec\_ctx = in\_stream->codec;

AVStream \*out\_stream = avformat\_new\_stream(ofmt\_ctx, in\_stream->codec->codec);

if (ifmt\_ctx->streams[i]->codec->codec\_type == AVMEDIA\_TYPE\_VIDEO) {

ret = avcodec\_copy\_context(out\_stream->codec, c);

out\_stream->codec->codec\_tag = 0;

}

/\*Audio Context Copy\*/

else if (ifmt\_ctx->streams[i]->codec->codec\_type == AVMEDIA\_TYPE\_AUDIO) {

ret = avcodec\_copy\_context(out\_stream->codec, in\_stream->codec);

out\_stream->codec->codec\_tag = 0;

}

}

* 이후 OutputFile의 Header를 저장한다.

av\_dump\_format(ofmt\_ctx, 0, outputfile, 1); // Output\_Format의 Detail을 출력

* if (!(ofmt->flags & AVFMT\_NOFILE)) {
* ret = avio\_open(&ofmt\_ctx->pb, outputfile, AVIO\_FLAG\_WRITE); // File\_Open
* if (ret < 0) {
* fprintf(stderr, "Could not open output file '%s'", outputfile);
* }
* }
* ret = avformat\_write\_header(ofmt\_ctx, NULL); // Header 작성
* if (ret < 0) {
* fprintf(stderr, "Error occurred when opening output file\n");
* }

1. Decoding

Decoding 하기에 앞서 이에 사용 할 Decoder를 설정한다.

* 1. CODEC\_ID에 맞는 디코더를 찾기 위해, avcodec\_find\_decoder 함수를 사용한다.

이를 통해 Codec을 초기화 한다.

AVCodec \*pVideoCodec = avcodec\_find\_decoder(f\_ctx->streams[nVSI]->codec->codec\_id);

5.2 avcodec\_open2를 사용해 CodecContext를 Codec으로 초기화한다.

if (avcodec\_open2(f\_ctx->streams[nVSI]->codec, pVideoCodec, NULL) < 0)

5.3 Decoder 설정이 완료 되면, Decoding을 한다.

av\_read\_frame 함수를 이용해 AVFormatContext와 Packet을 전달, frame을 읽어온다.

이후 Packet의 Stream을 구분하여 Video, Audio를 나눈다.

각각의 Packet에 Decode 함수인 avcodec\_decode\_video2를 사용한다.

다음은 선언과 초기화를 제외한 Decoding 과정의 코드 예시이다.

while (av\_read\_frame(f\_ctx, &packet) >= 0) {

if (packet.stream\_index == nVSI) {

// Decode Video

avcodec\_decode\_video2(dec\_ctx, pVFrame, &bGotPicture, &packet);

if (bGotPicture) {

// Encode Video (Example)

encoding(pVFrame, packet);

}

}

}

1. Encoding
   1. Encoding을 하기에 앞서 필요한 변수들을 초기화 해준다.

AVCodec \*codec; // codec for encode

AVCodecContext\* c = NULL; // Codec ContextAVCodec \*codec;

* 1. Decoding과 마찬가지로

AVCodec 에 원하는 Encoder를 찾기 위해, avcodec\_find\_encoder 함수를 사용한다.

codec = avcodec\_find\_encoder(Codec\_id);

* 1. AVCodecContext 를 AVCodec 에 맞게 Allocation한다.

c = avcodec\_alloc\_context3(codec);

* 1. AVCodecContext c 에 원하는 인코딩 정보를 입력한다.

예시)

c->bit\_rate = 500000;

c->width = 1280;

c->height = 528;

c->time\_base = { 1,25 };

c->pix\_fmt = AV\_PIX\_FMT\_YUV420P;

* 1. Packet을 사용하기 전 초기화 해준다.

av\_init\_packet(&packet);

* 1. Encoding 함수인 avcodec\_encode\_video2 를 사용한다.

Source Data인 pVFrame을 입력해, 인코딩 후 packet으로 반환한다.

avcodec\_encode\_video2(c, &pkt, NULL, &got\_output);

* 1. Encoding 된 Packet을 본 문서에서는 Media File로 저장한다.

fwrite를 이용해 file로 Encoded Packet을 저장한다.

if (got\_output) {

printf("Write frame %3d (size=%5d)\n", i, packet.size);

fwrite(packet.data, 1, packet.size, f);

if (!ret && got\_output && packet.size) {

packet.stream\_index = 0;

/\* Write the compressed frame to the media file. \*/

//ret = av\_interleaved\_write\_frame(oc, &packet);

}

av\_free\_packet(&packet);

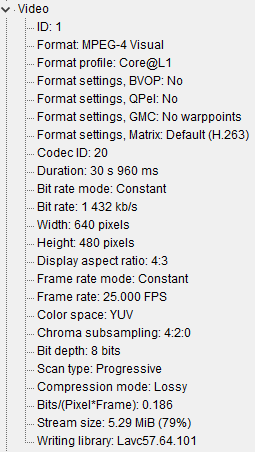
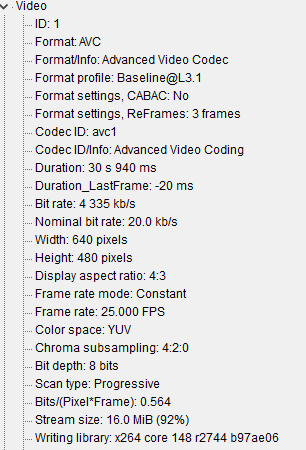
}

av\_interleaved\_write\_frame를 이용해 파라미터로 지정된 FormatContext에 맞게 파일을 쓸수 있다.

av\_free\_packet을 이용해, 사용 후, 남은 패킷은 해제한다.

1. 결과

변경 된 Media File의 정보를 확인 할 수 있다.

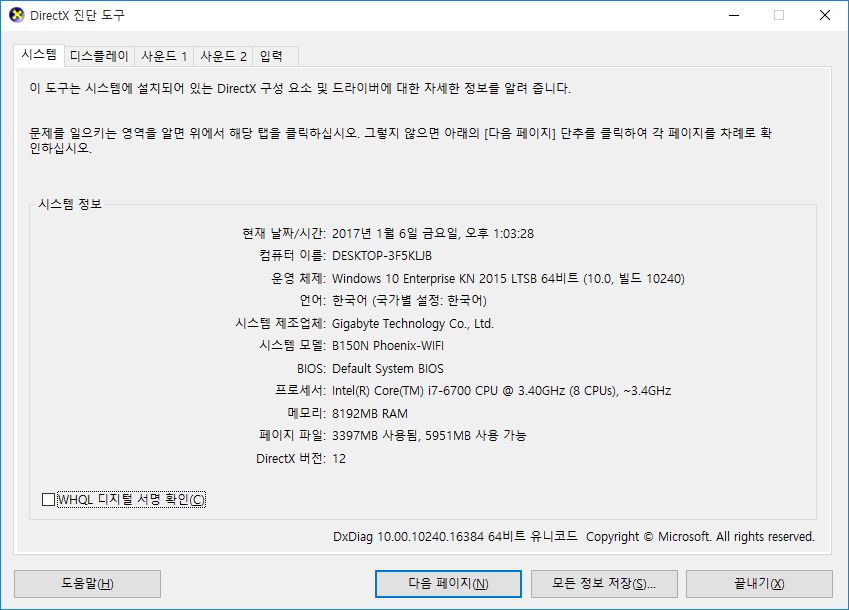
 

<인코딩 전 저장된 Video의 Info> <인코딩 후 저장된 Video의 Info>

1. 시스템 환경

Microsoft Visual Studio 2015 (v140)

FFMPEG 2.7.1



1. 참고문헌

FFMPEG 활용 코딩시에 참고했던 자료의 링크를 첨부한다.

1. <http://aslike.egloos.com/3081447>

기본적인 FFMPEG 코딩의 기초를 확인 할 수 있다.

1. <https://www.ffmpeg.org/doxygen/2.1/index.html>

FFMPEG 라이브러리의 Document를 확인 할 수 있다.

1. <http://dg087.tistory.com/61>

MinGW를 이용한, 윈도우용 라이브러리 제작을 확인 할수 있다.

1. <http://ffmpeg.org/doxygen/trunk/decoding__encoding_8c-source.html>

Decoding & Encoding의 예제 소스를 참고 할 수 있다.

본 Document의 Source code는

Github에서 다운로드 할수 있다.

https://github.com/Phigaro/FFMPEG