Blending video files

Coding Technology Document

- FFMPEG를 활용한 Video Blending–

2017.02.22

작성자 : 김문원

목차

[1. Summary 2](#_Toc473100050)

[2. System Environment 2](#_Toc473100051)

[3. Environmet Setting 2](#_Toc473100052)

[4. Coding Guide 3](#_Toc473100053)

[4.1 RGB class structure 3](#_Toc473100054)

[4.2 Media class structure 4](#_Toc473100055)

[4.3 Moving Image class structure 5](#_Toc473100056)

[4.4 Load video media files 5](#_Toc473100057)

[4.5 Load Image media files 8](#_Toc473100058)

[4.6 Main Thread 9](#_Toc473100059)

[4.7 MFC tool 11](#_Toc473100060)

[5. Result 12](#_Toc473100061)

[5.1 output 12](#_Toc473100062)

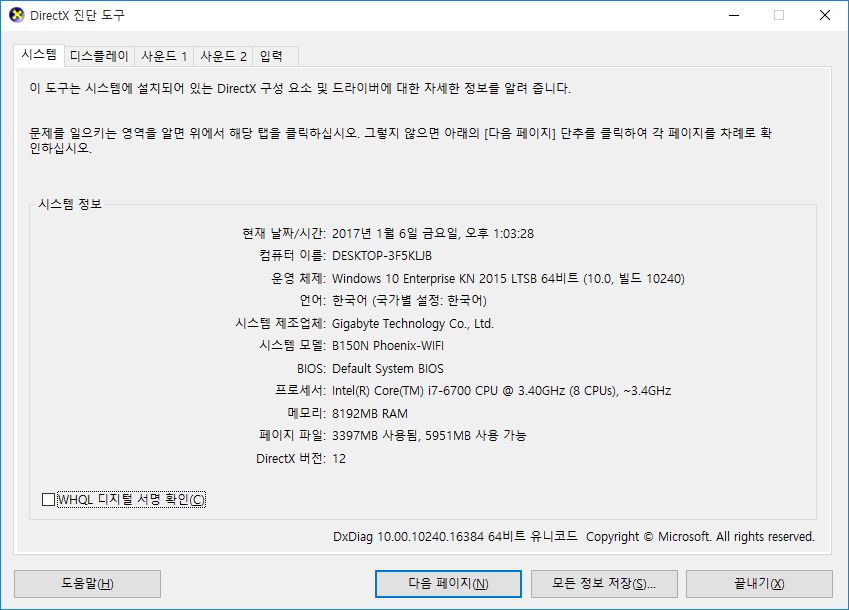
# Summary

본 문서는 FFmpeg transcoding 을 활용해 서로 다른 영상을 합성하고 연결하여 새로운 영상을 인코딩하는 과정을 담은 문서다. 이 문서에서 다루는 내용은 완성본이 아니라 지금까지 진행된 상황을 정리한 문서다.

# System Environment

Microsoft Visual Studio 2015 (v140)

FFMPEG 3.1.2



# Environmet Setting

* MFC 기반 프로그램으로 코딩하였지만 DLL로 묶기 쉽게 모두 클래스화 시켰다.
* X64 디버그 모드로 컴파일을 진행했으며 릴리즈 모드로 변경시 필요한 lib나 dll파일을 release 폴더로 옮겨주어야한다. (.exe 실행파일을 찾으면 쉽다)

# Coding Guide[[1]](#footnote-1)

## RGBImage 클래스 구조

* + - 스트림정보로부터 읽어온 frame을 RGB형태로 변환하여 저장하기위한 이미지클래스이다.

typedef uint8\_t u\_char;

struct data8 {

u\_char\* Rdata;

u\_char\* Gdata;

u\_char\* Bdata;

u\_char\* Ydata;

};

class RGBImage {

private:

int width;

int height;

data8 buffer;

public:

RGBImage();

RGBImage(int width, int height);

~RGBImage();

bool isGood();

void copyFrame(AVFrame\* frame);

u\_char getYdata(int index);

bool checkPixelWhite(int index);

void getPixelColor(u\_char& R, u\_char& G, u\_char& B, int index);

void setPixelColor(u\_char R, u\_char G, u\_char B, int index);

void RGB2YUV(AVFrame& frame);

};

* + - 주요 함수는 copyFrame, RGB2YUV 이다.

각각 Frame(YUV format) 을 RGB형태로 저장하고, 반대로 후자의 경우 RGB형태로 저장된 버퍼들을 이용해 다시 frame에 YUV 데이터를 입혀주는 함수이다.

#본 문서에서 시도한 블랜딩 방법은 합성할 영상의 흑백정도( Y값 기준)를 섞는 비율로 적용해 검은색이면 섞이지않고 흰색일수록 많이 섞이도록 하는 방법을 사용하였다. 그래서 data8 structure를 보면 Ydata 버퍼를 가지고있다.

## Media class 구조

* + - 입력받은 영상들의 정보를 저장할 클래스

class Media {

private:

AVInputFormat \*ifmt = NULL;

AVOutputFormat \*ofmt = NULL;

AVFrame\* filtered\_frame;

AVPacket read\_pkt;

AVPacket out\_pkt;

AVFrame\* pVFrame;

AVFrame\* pAFrame;

int nVSI = -1;

int nASI = -1;

public:

AVCodecContext\* enc\_codec\_ctx = NULL;

AVCodecContext\* dec\_ctx = NULL;

AVFormatContext \*ifmt\_ctx = NULL;

void init\_input\_format(const char\* input\_file\_path);

void init\_decoder(int flag);

void init\_encoder( int flag, AVCodecID v\_codec\_id,AVCodecContext\* temp\_ctx);

double calc\_time\_base();

};

* + - Public 으로 선언한 ifmt\_ctx는 파일을 읽어오고 나면 스트림 정보가 저장되어있다. Readpacket 함수를 통해 ifmt\_ctx에서 패킷을 한 개씩 받아오게 되는데 구조는 패킷을 빼내 갈수록 버퍼가 비어진다. 그래서 이미 합성한 부분을 다시 반복해서 합성하려면 새로 파일을 불러와야하는 과정이 필요하다.
    - Public에 선언된 세가지 함수는 파일을 불러오는데 중요한 역할을 한다.
    - Init\_input\_format 함수는 파일의 스트림 정보를 불러오고 직접 파일을 여는 함수이다.
    - Init\_decoder 는 스트림정보에 맞는 코덱을 찾아 코덱 context를 불러오는 역할을 한다.
    - Init\_encoder 는 이 영상을 어떤 식으로 인코딩할 것인가에 대한 정보를 담는것인데 이부분은 매우 중요하다. 아래에 인코딩 절차에서 설명한다.
      * 다음과 같은 함수들은 MFC 다이얼로그 부분에도 존재하는데 기존에 있던 소스를 참고용으로 남겨둔것이므로 실제로 사용되지는 않는다.

## Moving Image class 구조

* + - 합성되는 이미지의 위치를 정의해주기 위해 사용하는 클래스

class MovingImage {

private:

CPoint to;

CPoint from;

CPoint current;

int image\_index;

int count;

bool left;

int speed;

int range;

int is\_curved;

int was\_curved;

int sum;

int frame\_num;

double up\_speed;

public:

MovingImage();

MovingImage(CPoint from, CPoint to);

~MovingImage();

void SetToPos(CPoint data);

void SetFromPos(CPoint data);

void SetImage(int index);

int GetImageIndex();

CPoint GetCurrentPosition();

void calcNextPosition();

void goup();

bool ok2go();

};

* + - 원영상에 합성할 이미지나 영상을 원영상보다 작은 크기로 지정한다고 가정할 때 합성되는 이미지나 영상이 어떻게 움직이고 효과가 생길지를 Moving Image클래스를 통해 구현할수있다.
    - From -> To 는 어디서 어디로 가야하는지, Current는 현재 좌표를 나타낸다.

## Load Video Media Files

* + - 미리 설정해둔 파일경로로 파일들을 불러온다.
    - 파일은 원영상, 합성용 영상 으로 나눠서 입력받는다.
    - OnBnClickedButtonLoad() MFC 다이얼로그에 Load버튼을 누를시 동작하는 함수

void CProjectEveDlg::OnBnClickedButtonLoad()

{

// TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.

TCHAR szFilters[] = \_T("Video Files (\*.mp4)|\*.mp4|All Files (\*.\*)|\*.\*||");

CFileDialog fileDlg(TRUE, \_T("mp4"), NULL,OFN\_FILEMUSTEXIST | OFN\_HIDEREADONLY, szFilters);

// Display the file dialog. When user clicks OK,

CString fileName = NULL;

CString pathName = NULL;

wchar\_t\* wchar\_str;

int char\_str\_len; //char\* 형 변수의 길이를 위한 변수

//fileDlg.DoModal();

…중략

new\_media = new Media[3];//원본영상

test\_media = new Media[3];//합성영상

ifmt\_ctx = init\_input\_format(szFilePath);

new\_media[0].init\_input\_format(szFilePath);

new\_media[1].init\_input\_format("sample.mp4");

new\_media[2].init\_input\_format("sample2.mp4");

test\_media[0].init\_input\_format("massive.mp4");

test\_media[1].init\_input\_format("bbong.mp4");

test\_media[2].init\_input\_format("glare.mp4");

// Data member in your dialog class

// Set its text (string loaded from string table)

CString text = \_T("Load Complete");

m\_status\_text.SetWindowText(text);

//스트림정보 추가

setStreamInfo();

//CString message;

//message.Format(\_T("%d"), ifmt\_ctx->streams[0]->codec->width);

//MessageBox(\_T("width : ")+message, fileName);

}

* + - 결과적으로 new\_media라는 원본영상 3가지, test\_media라는 합성용 영상에 3가지 영상이 로드된다.
      * Char\* 형에서 CString으로의 변형과 그 반대 작업은 Modal 내부 코드를 보면 확인 가능하다.

## Image Media Load

* + - 영상과는 다른 합성용 이미지를 불러오는 부분이다.

int CProjectEveDlg::OpenImage(const char\* imageFileName)

{

…중략

while (av\_read\_frame(picFormatCtx, &packet) >= 0)

{

if (packet.stream\_index != 0)

continue;

int ret = avcodec\_decode\_video2(pCodecCtx, pictureFrame, &frameFinished, &packet);

if (ret > 0)

{

FilterContext\* imagefilter\_ctx = &vfilter\_ctx;

if (av\_buffersrc\_add\_frame(imagefilter\_ctx->src\_ctx, pictureFrame) < 0)

{

printf("Error occurred when putting frame into filter context\n");

}

while (1)

{

// Get frame from filter, if it returns < 0 then filter is currently empty.

if (av\_buffersink\_get\_frame(imagefilter\_ctx->sink\_ctx, m\_picture\_frame) < 0)

{

break;

}

} // while

printf("Frame is decoded, size %d\n", ret);

return 1;

}

else

printf("Error [%d] while decoding frame: %s\n", ret, strerror(AVERROR(ret)));

}

}

* + - 중간에 필터를 적용한 부분은 합성을 하기위해 원하는 크기로 이미지를 리사이징하는 필터다.
    - 현재 적용한 코드는 아니지만 이미지를 불러올 때 쓰기위해 남겨둔 코드다.

## Main Thread

* + - ThreadFirstData 쓰레드를 실행시키면 영상 합성이 진행된다..

UINT CProjectEveDlg::ThreadFirstData(LPVOID \_mothod)

* + - 다음 코드로 인코딩 되길 원하는 속성들을 직접 설정할 수 있다. .

AVCodecContext\* temp\_ctx = NULL;

temp\_ctx = pDlg->init\_encoder(pDlg->new\_media[0].ifmt\_ctx, Video\_flag);;

temp\_ctx->width = pDlg->m\_scale\_width;

temp\_ctx->height = pDlg->m\_scale\_height;

temp\_ctx->pix\_fmt = pDlg->new\_media[0].ifmt\_ctx->streams[pDlg->nVSI]->codec->pix\_fmt;

temp\_ctx->time\_base = pDlg->new\_media[0].ifmt\_ctx->streams[pDlg->nVSI]->time\_base;

temp\_ctx->bit\_rate = 50000 \* 1000;

for (int i = 0; i < 3; i++) {

pDlg->test\_media[i].init\_encoder(Video\_flag, AV\_CODEC\_ID\_H264, temp\_ctx);

}

* + - now\_pts 는 현재 인코딩하는 패킷의 pts값과 dts값을 정해주기위해 정의
    - N 값은 frame\_rate\*second 로 원하는 시간까지의 인코딩을 하기위한 변수
    - select, select\_origin은 원본영상과 합성영상을 선택하기위한 변수다.

long long now\_pts = 0;

int N = 0;

int select = 0;

int select\_origin = 0;

while (N < calc\_time\_base \*30)

* + - 다음의 반복문으로 인코딩이 시작되는데 calc\_time\_base는 인코딩하길 원하는 frame\_rate, 뒤에 곱하는 수는 인코딩을 원하는 영상의 길이다.

if (N < 5\*calc\_time\_base) {

select = 0;

}

else if (N < 10 \* calc\_time\_base) {

sw = true;

}

else if (N < 15 \* calc\_time\_base) {

back = 0;

sw = false;

select = 1;

}

else if (N < 20 \* calc\_time\_base) {

sw = true;

select\_origin = 2;

}

else {

sw = false;

select = 2;

}

* + - 조건문으로 구현해놓았지만 추후 알고리즘이 확정되면 변경해야할 가장 중요한 부분이다. 5초단위로 새로운 영상으로 변경되고 기존영상과 다음영상이 겹치는 부분에선 sw값을 true로 바꿔 영상 전환 효과를 적용하는 시점을 알 수 있게 해준다.

now\_pts += 30\*100;

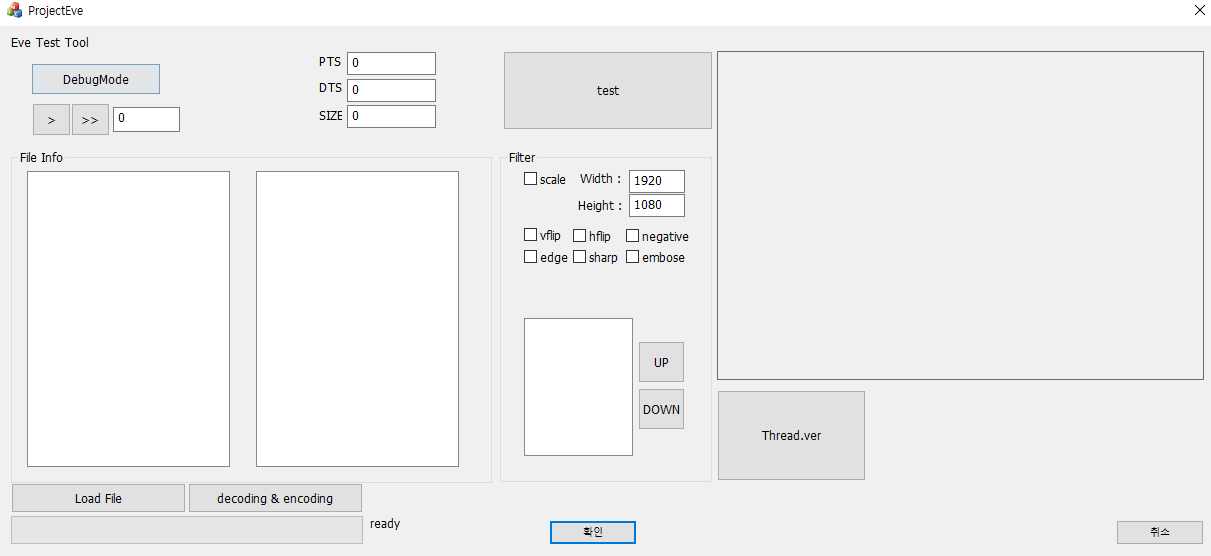
out\_pkt.pts = now\_pts;

out\_pkt.dts = now\_pts;

* + - 매우 중요한 부분인 pts,dts 설정 부분이다.
    - Pts는 패킷이 보여지는 presentation time stamp 이다.
    - Dts는 패킷이 디코딩 되어지는 decoding time stamp이다.
    - Pts와 dts는 동일하게 설정하여도 무관하며 30frame 기준으로 3000 씩 차이가 나게 패킷을 입력해주면 된다. (15frame -> 6000)
    - 반복문 안에 조건문이 크게 3가지 존재한다. 첫번째 조건문에서는 원본영상의 첫번째 영상 프레임이 찾아질 때 까지 반복한다. 만약 영상 프레임이 찾아졋다면 그 프레임을 기억해둔다.
    - 두번째 반복문은 현재 영상 전환효과가 필요한지를 확인한다. 만약 전환효과가 필요한 부분이라면 다음영상의 프레임을 얻어와서 첫번째 조건문에서 기억해뒀던 프레임과 원하는 영상처리를 한 후에 새로 덮어씌운다.

* + - 세번째 반복문에서는 합성을 할 영상의 프레임을 얻어와 기억해뒀던 프레임과 합성하는 작업을 한 후에 pts와 dts를 조절해서 인코딩 한다.

## MFC 툴 사용방법



* + - 단순한 테스트는 scale필터만 체크후 원하는 해상도를 입력하면 된다.
    - 현재 실시간 디버깅 모드는 비활성화 해두었다.
    - Load File로 원하는 원영상을 선택한 후 Thread.ver 버튼을 누르면 인코딩이 진행된다.

# Result

## 화면 전환 효과(밀어내기)



5.2 다양한 효과+비디오 적용 결과

- 프로젝트 폴더 내부에 따로 저장되어있다.

#추가 2월 28일 오후10시 까지

<https://github.com/gomyk/ProjectEve_MFC> 에 수정&정리된 코드 업로드 예정

1. [↑](#footnote-ref-1)