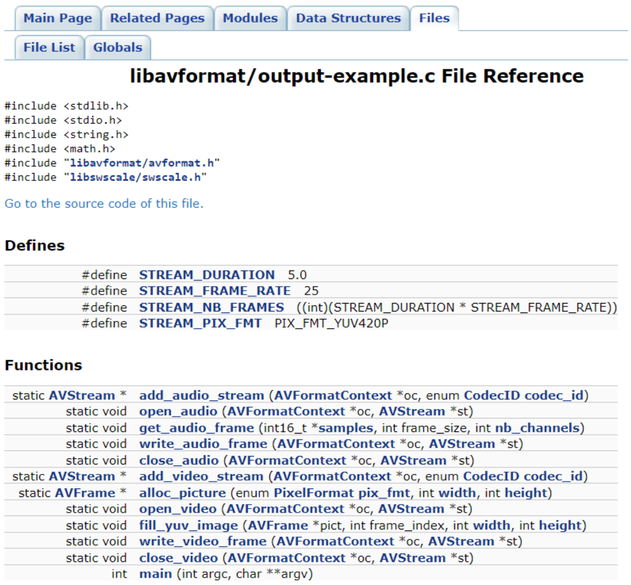
數字媒體軟件與系統開發

干皓丞，2101212850, 信息工程學院

2022 年 3 月 22 日

1 作業目標與章節摘要

FFMGEG 下載，並說明 output_example 。



Main Page | Related Pages | Modules | Data Structures | Files

File List | Globals

libavformat/output-example.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "libavformat/avformat.h"
#include "libswscale/swscale.h"
```

Go to the source code of this file.

Defines

#define	STREAM_DURATION 5.0
#define	STREAM_FRAME_RATE 25
#define	STREAM_NB_FRAMES ((int)(STREAM_DURATION * STREAM_FRAME_RATE))
#define	STREAM_PIX_FMT PIX_FMT_YUV420P

Functions

static AVStream *	add_audio_stream (AVFormatContext *oc, enum CodecID codec_id)
static void	open_audio (AVFormatContext *oc, AVStream *st)
static void	get_audio_frame (int16_t *samples, int frame_size, int nb_channels)
static void	write_audio_frame (AVFormatContext *oc, AVStream *st)
static void	close_audio (AVFormatContext *oc, AVStream *st)
static AVStream *	add_video_stream (AVFormatContext *oc, enum CodecID codec_id)
static AVFrame *	alloc_picture (enum PixelFormat pix_fmt, int width, int height)
static void	open_video (AVFormatContext *oc, AVStream *st)
static void	fill_yuv_image (AVFrame *pict, int frame_index, int width, int height)
static void	write_video_frame (AVFormatContext *oc, AVStream *st)
static void	close_video (AVFormatContext *oc, AVStream *st)
int	main (int argc, char **argv)

Fig. 1. 官方文件

2 文章與作業狀況

作業可以從 GitHub 下的 kancheng/kan-cs-report-in-2022 專案找到，作業程式碼與文件目錄為 kan-cs-report-in-2022/DMSASD/ffmpeg。實際執行的環境與實驗設備為 Google 的 Colab 、MacBook Pro (Retina, 15-inch, Mid 2014) 、Acer Aspire R7 與 HP Victus (Nvidia GeForce RTX 3060) 。

https://github.com/kancheng/kan-cs-report-in-2022/tree/main/DMSASD/ffmpeg

3 作業內容概述

此作業分為二大部分，第一部分說明前置準備與分析專案，第二部分則描述 output-example 該過程。

1. 前置準備與分析專案
2. output-example

4 前置準備與分析專案

其官方文件、原始碼檔案與相關連結如下:

1. https://github.com/FFmpeg/FFmpeg
2. https://www.ffmpeg.org/download.html
3. https://ffmpeg.org/doxygen/0.6/output-example_8c.html
4. https://ffmpeg.org/doxygen/0.6/output-example_8c-source.html
5. https://libav.org/documentation/doxygen/master/output_8c-example.html
6. https://ffmpeg.org/doxygen/trunk/output-example_8c.html
7. https://ffmpeg.org/doxygen/trunk/output-example_8c-source.html
8. https://ffmpeg.org/doxygen/trunk/avformat_8h-source.html
9. https://ffmpeg.org/doxygen/trunk/swscale_8h-source.html
10. https://ffmpeg.org/doxygen/trunk/mathematics_8h_source.html

4.1 Cloc 分析 FFMPEG

將下載來的原始碼進行 cloc 指令與結果分析

```
cloc .
```

```
(base) PS D:\FFmpeg-n3.0\ffmpeg-raw> cloc .
    7698 text files.
    4341 unique files.
    3359 files ignored.

github.com/AlDanial/cloc v 1.92  T=92.26 s (47.1 files/s, 18264.0 lines/s)
-------------------------------------------------------------------------------------

Language                          files          blank        comment
    code
-------------------------------------------------------------------------------------

C                                  2884         161056         107146
    1068012
C/C++ Header                       1034          17405          57010
    121608
Assembly                            281          11238          12309
    100291
Bourne Shell                         22            936            459
    8150
make                                 40            319             86
    4370
C++                                   3            343            143
    2206
Objective-C                           5            427            214
    2092
CUDA                                  5            259            121
    1399
OpenCL                               13            273            359
    1349
```

```
Perl                             7          256          349
    1050
Markdown                         7          204            0
    868
Python                           6          119           97
    577
XML                              9            4            0
    432
XSD                              1           45            4
    337
Windows Resource File            8           24          176
    240
Metal                            1           34           42
    202
CSS                              3           31           22
    140
Verilog-SystemVerilog            8            0            0
              56
awk                              1            6            5
              53
Ruby                             1            9            0
              52
HTML                             1            5            4
              44
YAML                             1            0            0
              30
-------------------------------------------------------------------------------

SUM:                          4341       192993       178546
    1313558
-------------------------------------------------------------------------------

(base) PS D:\FFmpeg-n3.0\ffmpeg-raw>
```

4.2 Mac 編譯 FFMPEG

嘗試用內部的 configure 進行編譯。

```
./configure --disable-x86asm
ffmpeg
```

當然不用編譯，也可以使用 brew 等工具安裝現有編譯好的 FFMPEG。

Fig. 2. 編譯

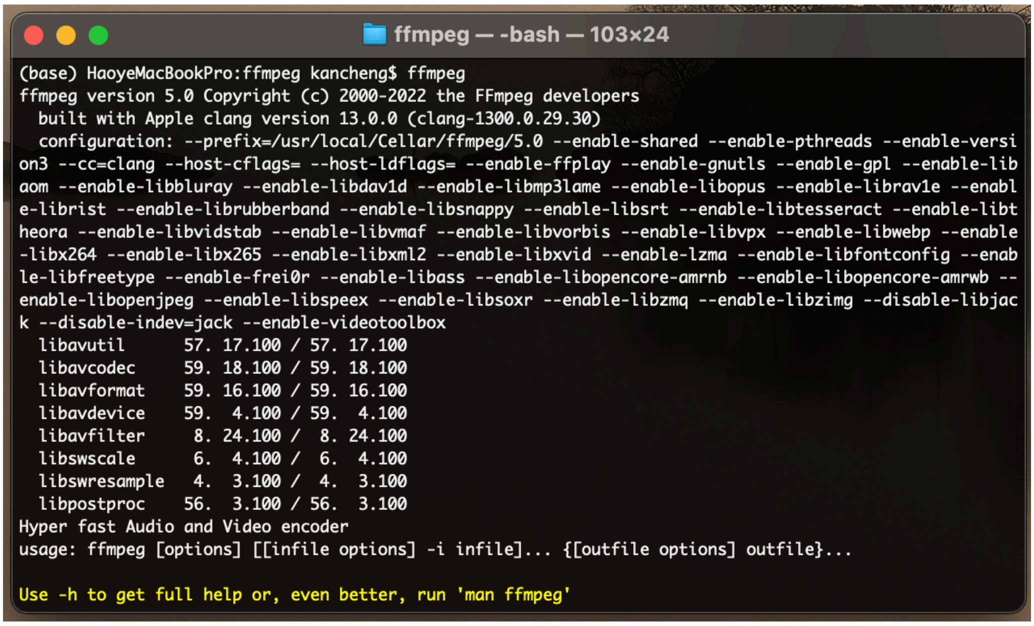


Fig. 3. 編譯成功

4.3 output-example.c 版本

已知社群中文素材所找的內容，其實時間大多為 2010 左右，而該範例原始碼於 0.6 還可以找到，但在 0.7 版此檔案就已經被拔掉，其大的版本可以在 v0.6.1 中下載。

從 GitHub 的版本號中可以看到是由 Michael Niedermayer 所提交的合併更動時消失。在此可以用指令將下載來的 FFMPEG 匯出 Git Commit 紀錄，來追專案的變化

將所有 Log 紀錄用指令輸出至一個 txt 檔案中。

```
git log > log.txt
```

接下來發現 Michael Niedermayer ，FFMPEG 的專案開發者的相關批改，也就是在這個合併後，該檔案就沒再出現了。

```
commit fbe02459dc4f3c8f4d758c1a90ed8e35a800f3b9
Merge: 9a1963fbb8 b4675d0fbf
Author: Michael Niedermayer <michael@niedermayer.cc>
Date:   Mon Jul 16 01:32:52 2012 +0200

    Merge remote-tracking branch 'qatar/master'

    * qatar/master:
      configure: Check for CommandLineToArgvW
      vc1dec: Do not use random pred_flag if motion vector data is skipped
      vp8: Enclose pthread function calls in ifdefs
      snow: refactor code to work around a compiler bug in MSVC.
      vp8: Include the thread headers before using the pthread types
      configure: Check for getaddrinfo in ws2tcpip.h, too
      vp8: implement sliced threading
      vp8: move data from VP8Context->VP8Macroblock
      vp8: refactor decoding a single mb_row
      doc: update api changes with the right commit hashes
      mem: introduce av_malloc_array and av_mallocz_array

    Conflicts:
            configure
            doc/APIchanges
            libavcodec/vp8.c
            libavutil/mem.h
            libavutil/version.h

    Merged-by: Michael Niedermayer <michaelni@gmx.at>
```

為了確定該檔案是否有可能只是改名，過者遷移路徑，在此用另外一個指令繼續。

Git 追檔案更動

```
git log --full-history -- libavformat/output-example.c
```

最後發現被搬移至此 doc/examples/output.c ，更後面就沒有該範例的存在。

```
libavformat/output-example.c → doc/examples/output.c
```

其 Log 的顯示於此。

```
commit ab81f24ad43bddf77ddd25cba86780c1c884996c
Author: Diego Biurrun <diego@biurrun.de>
Date:   Sat Nov 2 17:05:28 2013 +0100

    build: Integrate multilibrary examples into the build system

    This includes moving libavformat/output-example to doc/examples/output.
```

Fig. 4. GitHub 紀錄

Git Commit 滾動指令

```
git reset --hard HEAD^
git reset --hard [COMMIT]
```

綜上所述，目前遇到有兩個版本，一個是 doc/examples/output.c 最後版本，一個是 libavformat/output-example.c 在最後 v0.6 的版本。在此用 vim 進行對比。

```
vim -d output.c output-example.c
```

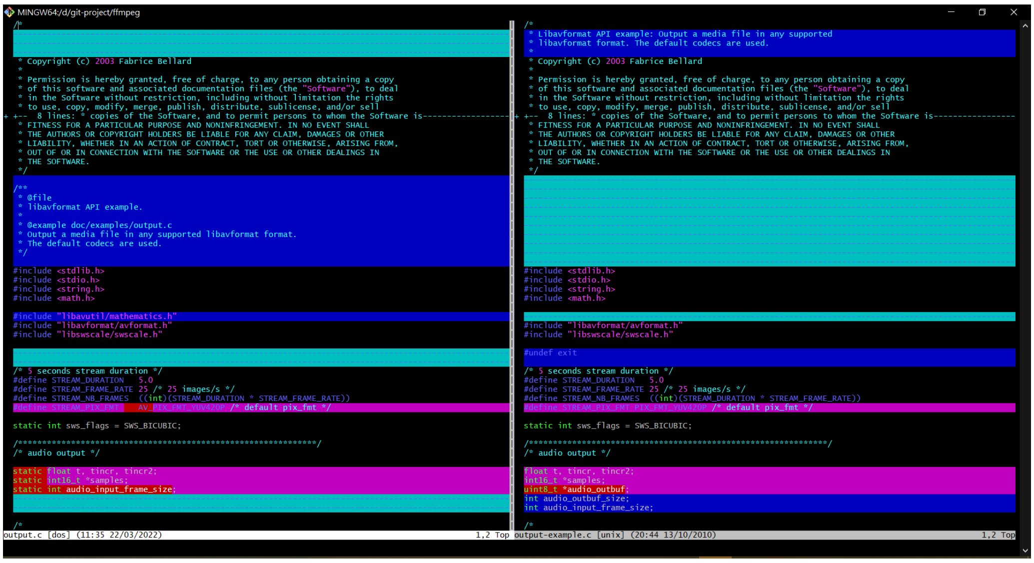


Fig. 5. Vim 行數對比

從上面可以看到 doc/examples/output.c 相對 libavformat/output-example.c 多了不少更進，在此，本作業用 doc/examples/output.c 版本進行分析。

5 output-example

5.1 output-example

在此本作業在此針對 doc/examples/output.c 整理成一份檔名為 output-zh-read.c 的中文註解說明版本如下。而後分析都根據官方文件與專案程式碼。

https://github.com/kancheng/kan-cs-report-in-2022/blob/main/DMSASD/ffmpeg/output-zh-read.c

5.2 output-example

該程式碼開頭為給使用者版權宣告註解與先前從 libavformat API example 遷移過來的說明。

```
/*
 * Copyright (c) 2003 Fabrice Bellard
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
 * THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

/**
 * @file
 * libavformat API example.
 *
 * @example doc/examples/output.c
 * Output a media file in any supported libavformat format.
 * The default codecs are used.
 */
```

再來就是該程式碼的 C 語言套件庫，部分包含常用的 stdlib.h 、stdio.h、string.h、math.h。FFMPEG 自身的 Library，當中包含了 mathematics.h 數學處理包，與 AVFormat 有關的 avformat.h。跟比如將 YUV420P 轉換成 YUYV422 會用到的換圖大小 swscale.h。最後則是該程式會需要的巨集定義與相關的變數宣告

```
// C Library

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

// FFMPEG Library
#include "libavutil/mathematics.h"
#include "libavformat/avformat.h"
```

```
#include "libswscale/swscale.h"

// 巨集定義
/* 5 seconds stream duration */
#define STREAM_DURATION   5.0
#define STREAM_FRAME_RATE 25 /* 25 images/s */
#define STREAM_NB_FRAMES  ((int)(STREAM_DURATION * STREAM_FRAME_RATE))
#define STREAM_PIX_FMT    AV_PIX_FMT_YUV420P /* default pix_fmt */

static int sws_flags = SWS_BICUBIC;
```

AVStream 為負責音源輸出與加入加入音源輸出的函式。當中有個尋找音頻編碼器會判斷，然後放入樣本參數，另外在一些 stream header 進行額外處理。

```
// 音源輸出
static float t, tincr, tincr2;
static int16_t *samples;
static int audio_input_frame_size;

/* 加入音源輸出
 * add an audio output stream
 */
static AVStream *add_audio_stream(AVFormatContext *oc, enum AVCodecID codec_id)
{
    AVCodecContext *c;
    AVStream *st;
    AVCodec *codec;
// 找到音頻編碼器
    /* find the audio encoder */
    codec = avcodec_find_encoder(codec_id);
    if (!codec) {
        fprintf(stderr, "codec not found\n");
        exit(1);
    }

    st = avformat_new_stream(oc, codec);
    if (!st) {
        fprintf(stderr, "Could not alloc stream\n");
        exit(1);
    }

    c = st->codec;

// 放樣本參數
    /* put sample parameters */
    c->sample_fmt  = AV_SAMPLE_FMT_S16;
    c->bit_rate    = 64000;
    c->sample_rate = 44100;
```

```
    c->channels     = 2;

// 某些格式希望 stream header 是分開的
    // some formats want stream headers to be separate
    if (oc->oformat->flags & AVFMT_GLOBALHEADER)
        c->flags |= CODEC_FLAG_GLOBAL_HEADER;

    return st;
}
```

開啟音源函式部分，該函數會進行初始化。當中會有以每秒 110 Hz 的速度遞增頻率，M_PI 則是 FFMPEG 的 mathematics.h。

```
static void open_audio(AVFormatContext *oc, AVStream *st)
{
    AVCodecContext *c;

    c = st->codec;

    /* open it */
    if (avcodec_open2(c, NULL, NULL) < 0) {
        fprintf(stderr, "could not open codec\n");
        exit(1);
    }

    /* init signal generator */
    t     = 0;
    tincr = 2 * M_PI * 110.0 / c->sample_rate;
    /* increment frequency by 110 Hz per second */
    tincr2 = 2 * M_PI * 110.0 / c->sample_rate / c->sample_rate;

    if (c->codec->capabilities & CODEC_CAP_VARIABLE_FRAME_SIZE)
        audio_input_frame_size = 10000;
    else
        audio_input_frame_size = c->frame_size;
    samples = av_malloc(audio_input_frame_size *
                        av_get_bytes_per_sample(c->sample_fmt) *
                        c->channels);
}
```

get_audio_frame 準備“frame_size”樣本的 16 位虛擬音頻幀和’nb_channels’ 頻道。

```
static void get_audio_frame(int16_t *samples, int frame_size, int nb_channels)
{
    int j, i, v;
    int16_t *q;

    q = samples;
    for (j = 0; j < frame_size; j++) {
```

```
        v = (int)(sin(t) * 10000);
        for (i = 0; i < nb_channels; i++)
            *q++ = v;
        t     += tincr;
        tincr += tincr2;
    }
}
```

write_audio_frame 則是進行編碼工作，pkt 的數據和大小初始必須為 0，其後將壓縮幀寫入媒體文件。

```
static void write_audio_frame(AVFormatContext *oc, AVStream *st)
{
    AVCodecContext *c;
    AVPacket pkt = { 0 }; // data and size must be 0;
    AVFrame *frame = av_frame_alloc();
    int got_packet;

    av_init_packet(&pkt);
    c = st->codec;

    get_audio_frame(samples, audio_input_frame_size, c->channels);
    frame->nb_samples = audio_input_frame_size;
    avcodec_fill_audio_frame(frame, c->channels, c->sample_fmt,
                             (uint8_t *)samples,
                             audio_input_frame_size *
                             av_get_bytes_per_sample(c->sample_fmt) *
                             c->channels, 1);

    avcodec_encode_audio2(c, &pkt, frame, &got_packet);
    if (!got_packet)
        return;

    pkt.stream_index = st->index;

    /* Write the compressed frame to the media file. */
    if (av_interleaved_write_frame(oc, &pkt) != 0) {
        fprintf(stderr, "Error while writing audio frame\n");
        exit(1);
    }
    avcodec_free_frame(&frame);
}
```

close_audio 為關閉音源。

```
static void close_audio(AVFormatContext *oc, AVStream *st)
{
    avcodec_close(st->codec);

    av_free(samples);
```

```
}
```

AVStream 部分為影像輸出部分，過程中會先找到影像的 encoder，而後放其參數，且設分辨率必須是二的倍數。而 timebase 這是表示幀時間戳的基本時間單位（以秒為單位）。對於固定 fps 內容，時基應為 1/幀速率，時間戳增量應等於 1。最後最多每十二幀發射一幀。同時為了需要避免使用某些係數溢出的宏塊。這不會發生在普通視頻中，它只是在這裡發生，因為色度平面的運動與亮度平面不匹配。另外為了測試添加了 B 幀。

```
static AVFrame *picture, *tmp_picture;
static int frame_count;
static AVStream *add_video_stream(AVFormatContext *oc, enum AVCodecID codec_id)
{
    AVCodecContext *c;
    AVStream *st;
    AVCodec *codec;
    codec = avcodec_find_encoder(codec_id);
    if (!codec) {
        fprintf(stderr, "codec not found\n");
        exit(1);
    }
    st = avformat_new_stream(oc, codec);
    if (!st) {
        fprintf(stderr, "Could not alloc stream\n");
        exit(1);
    }
    c = st->codec;
    c->bit_rate = 400000;
    c->width    = 352;
    c->height   = 288;
    c->time_base.den = STREAM_FRAME_RATE;
    c->time_base.num = 1;
    c->gop_size      = 12;
    c->pix_fmt       = STREAM_PIX_FMT;
    if (c->codec_id == AV_CODEC_ID_MPEG2VIDEO) {
        c->max_b_frames = 2;
    }
    if (c->codec_id == AV_CODEC_ID_MPEG1VIDEO) {
        c->mb_decision = 2;
    }
    if (oc->oformat->flags & AVFMT_GLOBALHEADER)
        c->flags |= CODEC_FLAG_GLOBAL_HEADER;
    return st;
}
```

AVFrame 為處理 AVCodecContext 的重要組成部分。

```
static AVFrame *alloc_picture(enum AVPixelFormat pix_fmt, int width, int height)
{
    AVFrame *picture;
    uint8_t *picture_buf;
```

```
    int size;

    picture = av_frame_alloc();
    if (!picture)
        return NULL;
    size        = avpicture_get_size(pix_fmt, width, height);
    picture_buf = av_malloc(size);
    if (!picture_buf) {
        av_free(picture);
        return NULL;
    }
    avpicture_fill((AVPicture *)picture, picture_buf,
                   pix_fmt, width, height);
    return picture;
}
```

open_video 為開啟影像函式，當中有一個 codec，同時分配編碼的原始圖片，同時如果輸出格式不是 YUV420P，那麼也需要一張臨時的 YUV420P 圖片。然後將其轉換為所需的輸出格式。

```
static void open_video(AVFormatContext *oc, AVStream *st)
{
    AVCodecContext *c;
    c = st->codec;
    if (avcodec_open2(c, NULL, NULL) < 0) {
        fprintf(stderr, "could not open codec\n");
        exit(1);
    }
    picture = alloc_picture(c->pix_fmt, c->width, c->height);
    if (!picture) {
        fprintf(stderr, "Could not allocate picture\n");
        exit(1);
    }
    tmp_picture = NULL;
    if (c->pix_fmt != AV_PIX_FMT_YUV420P) {
        tmp_picture = alloc_picture(AV_PIX_FMT_YUV420P, c->width, c->height);
        if (!tmp_picture) {
            fprintf(stderr, "Could not allocate temporary picture\n");
            exit(1);
        }
    }
}
```

fill_yuv_image 的函式為 dummy image 的處理，當中控制 Y、Cb、Cr。

```
static void fill_yuv_image(AVFrame *pict, int frame_index,
                           int width, int height)
{
    int x, y, i;
    i = frame_index;
```

```
    for (y = 0; y < height; y++)
        for (x = 0; x < width; x++)
            pict->data[0][y * pict->linesize[0] + x] = x + y + i * 3;
    for (y = 0; y < height / 2; y++) {
        for (x = 0; x < width / 2; x++) {
            pict->data[1][y * pict->linesize[1] + x] = 128 + y + i * 2;
            pict->data[2][y * pict->linesize[2] + x] = 64 + x + i * 5;
        }
    }
}
```

write_video_frame 在此處理影像，不再需要壓縮幀。如果使用 B 幀，編解碼器有幾幀的延遲，所以我們通過再次傳遞相同的圖片來獲得最後一幀，由於該函式只生成一張 YUV420P 圖片，如果需要會將其轉換為編解碼器像素格式。而後面則是 encode 影像的處理。最後將壓縮幀寫入媒體文件。

```
static void write_video_frame(AVFormatContext *oc, AVStream *st)
{
    int ret;
    AVCodecContext *c;
    static struct SwsContext *img_convert_ctx;

    c = st->codec;

    if (frame_count >= STREAM_NB_FRAMES) {
    } else {
        if (c->pix_fmt != AV_PIX_FMT_YUV420P) {
            if (img_convert_ctx == NULL) {
                img_convert_ctx = sws_getContext(c->width, c->height,
                                                 AV_PIX_FMT_YUV420P,
                                                 c->width, c->height,
                                                 c->pix_fmt,
                                                 sws_flags, NULL, NULL, NULL);
                if (img_convert_ctx == NULL) {
                    fprintf(stderr,
                            "Cannot initialize the conversion context\n");
                    exit(1);
                }
            }
            fill_yuv_image(tmp_picture, frame_count, c->width, c->height);
            sws_scale(img_convert_ctx, tmp_picture->data, tmp_picture->linesize,
                      0, c->height, picture->data, picture->linesize);
        } else {
            fill_yuv_image(picture, frame_count, c->width, c->height);
        }
    }

    if (oc->oformat->flags & AVFMT_RAWPICTURE) {
        AVPacket pkt;
```

```
        av_init_packet(&pkt);
        pkt.flags        |= AV_PKT_FLAG_KEY;
        pkt.stream_index  = st->index;
        pkt.data          = (uint8_t *)picture;
        pkt.size          = sizeof(AVPicture);
        ret = av_interleaved_write_frame(oc, &pkt);
    } else {
        AVPacket pkt = { 0 };
        int got_packet;
        av_init_packet(&pkt);
        ret = avcodec_encode_video2(c, &pkt, picture, &got_packet);
        if (!ret && got_packet && pkt.size) {
            if (pkt.pts != AV_NOPTS_VALUE) {
                pkt.pts = av_rescale_q(pkt.pts,
                                       c->time_base, st->time_base);
            }
            if (pkt.dts != AV_NOPTS_VALUE) {
                pkt.dts = av_rescale_q(pkt.dts,
                                       c->time_base, st->time_base);
            }
            pkt.stream_index = st->index;
            ret = av_interleaved_write_frame(oc, &pkt);
        } else {
            ret = 0;
        }
    }
    if (ret != 0) {
        fprintf(stderr, "Error while writing video frame\n");
        exit(1);
    }
    frame_count++;
}
```

close_video 關閉影像函式。

```
static void close_video(AVFormatContext *oc, AVStream *st)
{
    avcodec_close(st->codec);
    av_free(picture->data[0]);
    av_free(picture);
    if (tmp_picture) {
        av_free(tmp_picture->data[0]);
        av_free(tmp_picture);
    }
}
```

最後則是 main 函式，已開始會初始化 libavcodec，並註冊所有編解碼器和格式，並且從名稱中自動檢測輸出格式。默認為 MPEG，而後分配輸出媒體內文，最後使用默認格式編解碼器添加音頻和視頻流並初始化編解

碼器。

當現在所有參數都設置好後，則可以打開音頻和視頻編解碼器並分配必要的編碼緩衝區。另外過程中有需要，可打開輸出文件，並寫入 stream header，計算計算當前的音頻和視頻時間，最後寫入交錯的音頻和視頻幀與關閉每一個 codec 跟輸出檔案。

```
int main(int argc, char **argv)
{
    const char *filename;
    AVOutputFormat *fmt;
    AVFormatContext *oc;
    AVStream *audio_st, *video_st;
    double audio_pts, video_pts;
    int i;
    av_register_all();
    if (argc != 2) {
        printf("usage: %s output_file\n"
               "API example program to output a media file with libavformat.\n"
               "The output format is automatically guessed according to the file
                   extension.\n"
               "Raw images can also be output by using '%%d' in the filename\n"
               "\n", argv[0]);
        return 1;
    }

    filename = argv[1];
    fmt = av_guess_format(NULL, filename, NULL);
    if (!fmt) {
        printf("Could not deduce output format from file extension: using MPEG.\
           n");
        fmt = av_guess_format("mpeg", NULL, NULL);
    }
    if (!fmt) {
        fprintf(stderr, "Could not find suitable output format\n");
        return 1;
    }
    oc = avformat_alloc_context();
    if (!oc) {
        fprintf(stderr, "Memory error\n");
        return 1;
    }
    oc->oformat = fmt;
    snprintf(oc->filename, sizeof(oc->filename), "%s", filename);
    video_st = NULL;
    audio_st = NULL;
    if (fmt->video_codec != AV_CODEC_ID_NONE) {
        video_st = add_video_stream(oc, fmt->video_codec);
    }
```

```
    if (fmt->audio_codec != AV_CODEC_ID_NONE) {
        audio_st = add_audio_stream(oc, fmt->audio_codec);
    }
    if (video_st)
        open_video(oc, video_st);
    if (audio_st)
        open_audio(oc, audio_st);

    av_dump_format(oc, 0, filename, 1);
    if (!(fmt->flags & AVFMT_NOFILE)) {
        if (avio_open(&oc->pb, filename, AVIO_FLAG_WRITE) < 0) {
            fprintf(stderr, "Could not open '%s'\n", filename);
            return 1;
        }
    }
    avformat_write_header(oc, NULL);
    for (;;) {
        if (audio_st)
            audio_pts = (double)audio_st->pts.val * audio_st->time_base.num /
                audio_st->time_base.den;
        else
            audio_pts = 0.0;

        if (video_st)
            video_pts = (double)video_st->pts.val * video_st->time_base.num /
                        video_st->time_base.den;
        else
            video_pts = 0.0;
        if ((!audio_st || audio_pts >= STREAM_DURATION) &&
            (!video_st || video_pts >= STREAM_DURATION))
            break;
        if (!video_st || (video_st && audio_st && audio_pts < video_pts)) {
            write_audio_frame(oc, audio_st);
        } else {
            write_video_frame(oc, video_st);
        }
    }
    av_write_trailer(oc);
    if (video_st)
        close_video(oc, video_st);
    if (audio_st)
        close_audio(oc, audio_st);
    for (i = 0; i < oc->nb_streams; i++) {
        av_freep(&oc->streams[i]->codec);
        av_freep(&oc->streams[i]);
    }
    if (!(fmt->flags & AVFMT_NOFILE))
```

```
        avio_close(oc->pb);
    av_free(oc);
    return 0;
}
```