

# 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 29 日

## 1 作業目標與章節摘要

FFMPEG 下載，並說明 output\_example。

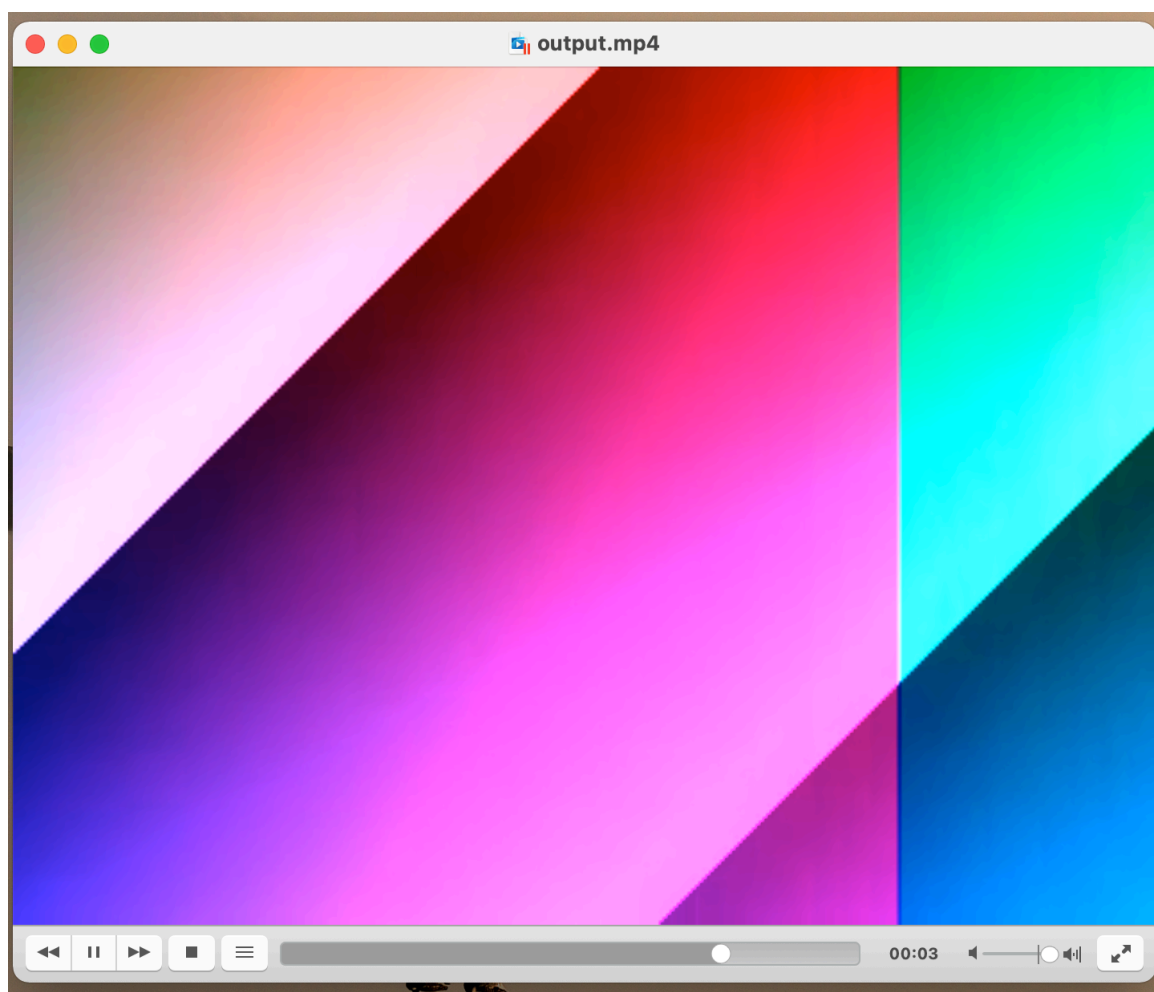


Fig. 1. 最後測試結果

## 2 文章與作業狀況

作業可以從 GitHub 下的 [kancheng/kan-cs-report-in-2022](https://github.com/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 原始碼與測試該過程，最後補充 /doc/example/muxing.c。

1. 前置準備與分析專案
2. output-example 原始碼與測試該過程
3. /doc/example/muxing.c。

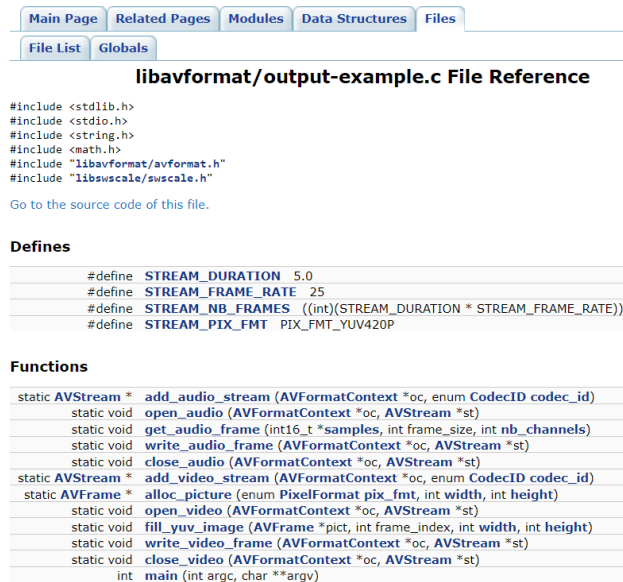


Fig. 2. 官方文件

### 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](https://ffmpeg.org/doxygen/0.6/output-example_8c.html)
4. [https://ffmpeg.org/doxygen/0.6/output-example\\_8c-source.html](https://ffmpeg.org/doxygen/0.6/output-example_8c-source.html)
5. [https://libav.org/documentation/doxygen/master/output\\_8c-example.html](https://libav.org/documentation/doxygen/master/output_8c-example.html)
6. [https://ffmpeg.org/doxygen/trunk/output-example\\_8c.html](https://ffmpeg.org/doxygen/trunk/output-example_8c.html)
7. [https://ffmpeg.org/doxygen/trunk/output-example\\_8c-source.html](https://ffmpeg.org/doxygen/trunk/output-example_8c-source.html)
8. [https://ffmpeg.org/doxygen/trunk/avformat\\_8h-source.html](https://ffmpeg.org/doxygen/trunk/avformat_8h-source.html)
9. [https://ffmpeg.org/doxygen/trunk/swscale\\_8h-source.html](https://ffmpeg.org/doxygen/trunk/swscale_8h-source.html)
10. [https://ffmpeg.org/doxygen/trunk/mathematics\\_8h\\_source.html](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

```
1 cloc .

1 (base) PS D:\FFmpeg-n3.0\ffmpeg-raw> cloc .
2 7698 text files .
3 4341 unique files .
```

```

4      3359 files ignored.
5
6 github.com/AIDanial/cloc v 1.92 T=92.26 s (47.1 files/s, 18264.0 lines/s)
7 -----
8 Language                files          blank          comment
9 -----
10 C                        2884          161056          107146
11      1068012
12 C/C++ Header            1034          17405           57010
13      121608
14 Assembly                281           11238           12309
15      100291
16 Bourne Shell            22            936             459
17      8150
18 make                    40            319             86
19      4370
20 C++                      3             343            143
21      2206
22 Objective-C              5             427            214
23      2092
24 CUDA                    5             259            121
25      1399
26 OpenCL                  13            273            359
27      1349
28 Perl                    7             256            349
29      1050
30 Markdown                7             204             0
31      868
32 Python                  6             119            97
33      577
34 XML                     9              4              0
35      432
36 XSD                      1             45              4
37      337
38 Windows Resource File   8             24            176
39      240
40 Metal                   1             34             42
41      202
42 CSS                      3             31             22
43      140
44 Verilog-SystemVerilog   8             0              0
45      56
46 awk                     1              6              5
47
48      53

```

```

29 Ruby                      1          9          0
      52
30 HTML                      1          5          4
      44
31 YAML                      1          0          0
      30
32 -----
33 SUM:                      4341       192993       178546
      1313558
34 -----
35 (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。

```

1 ./configure --disable-x86asm
2 ffmpeg

```

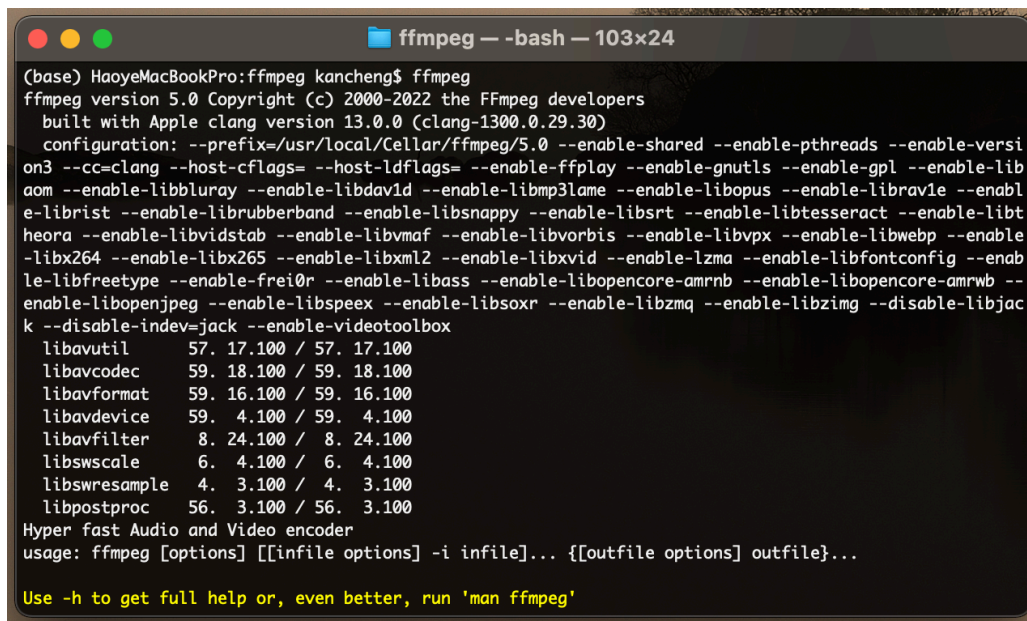
當然不用編譯，也可以使用 brew 等工具安裝現有編譯好的 FFMPEG。

```

(base) HaoyeMacBookPro:ffmpeg kancheng$ ./configure --disable-x86asm
install prefix             /usr/local
source path                .
C compiler                 gcc
C library                  libm
ARCH                      x86 (generic)
big-endian                 no
runtime cpu detection      yes
standalone assembly        no
x86 assembler              nasm
MMX enabled                yes
MMXEXT enabled             yes
3DNow! enabled             yes
3DNow! extended enabled    yes
SSE enabled                yes
SSSE3 enabled              yes
AESNI enabled              yes
AVX enabled                yes
AVX2 enabled               yes
AVX-512 enabled            yes

```

Fig. 3. 編譯



```

(base) HaoyeMacBookPro:ffmpeg kancheng$ ffmpeg
ffmpeg version 5.0 Copyright (c) 2000-2022 the FFmpeg developers
  built with Apple clang version 13.0.0 (clang-1300.0.29.30)
  configuration: --prefix=/usr/local/Cellar/ffmpeg/5.0 --enable-shared --enable-pthreads --enable-versi
on3 --cc=clang --host-cflags= --host-ldflags= --enable-ffplay --enable-gnutls --enable-gpl --enable-lib
aom --enable-libbluray --enable-libdav1d --enable-libmp3lame --enable-libopus --enable-librav1e --enabl
e-librist --enable-librubberband --enable-libsnpappy --enable-libsrt --enable-libtesseract --enable-libt
heora --enable-libvidstab --enable-libvmaf --enable-libvorbis --enable-libvpx --enable-libwebp --enable
-libx264 --enable-libx265 --enable-libxml2 --enable-libxvid --enable-lzma --enable-libfontconfig --enab
le-libfreetype --enable-frei0r --enable-libass --enable-libopencore-amrnb --enable-libopencore-amrwb --
enable-libopenjpeg --enable-libspeex --enable-libsoxr --enable-libzmq --enable-libzimg --disable-libjac
k --disable-indev=jack --enable-videotoolbox
  libavutil      57. 17.100 / 57. 17.100
  libavcodec     59. 18.100 / 59. 18.100
  libavformat    59. 16.100 / 59. 16.100
  libavdevice    59.  4.100 / 59.  4.100
  libavfilter     8. 24.100 /  8. 24.100
  libswscale      6.  4.100 /  6.  4.100
  libswresample  4.  3.100 /  4.  3.100
  libpostproc   56.  3.100 / 56.  3.100
Hyper fast Audio and Video encoder
usage: ffmpeg [options] [[infile options] -i infile]... {[outfile options] outfile}...

Use -h to get full help or, even better, run 'man ffmpeg'

```

Fig. 4. 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。

```
1 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。

```

1 commit fbe02459dc4f3c8f4d758c1a90ed8e35a800f3b9
2 Merge: 9a1963fbb8 b4675d0fbf
3 Author: Michael Niedermayer <michael@niedermayer.cc>
4 Date: Mon Jul 16 01:32:52 2012 +0200
5
6 Merge remote-tracking branch 'qatar/master'
7
8 * qatar/master:
9     configure: Check for CommandLineToArgvW
10    vc1dec: Do not use random pred_flag if motion vector data is skipped
11    vp8: Enclose pthread function calls in ifdefs
12    snow: refactor code to work around a compiler bug in MSVC.
13    vp8: Include the thread headers before using the pthread types
14    configure: Check for getaddrinfo in ws2tcpip.h, too
15    vp8: implement sliced threading
16    vp8: move data from VP8Context->VP8Macroblock
17    vp8: refactor decoding a single mb_row
18    doc: update api changes with the right commit hashes

```

```

19      mem: introduce av_malloc_array and av_mallocz_array
20
21      Conflicts:
22          configure
23          doc/APIchanges
24          libavcodec/vp8.c
25          libavutil/mem.h
26          libavutil/version.h
27
28      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

```
1 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。

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

其 Log 的顯示於此。

```

1 commit ab81f24ad43bddf77ddd25cba86780c1c884996c
2 Author: Diego Biurrun <diego@biurrun.de>
3 Date:   Sat Nov 2 17:05:28 2013 +0100
4
5     build: Integrate multilibrary examples into the build system
6
7     This includes moving libavformat/output-example to doc/examples/output.

```

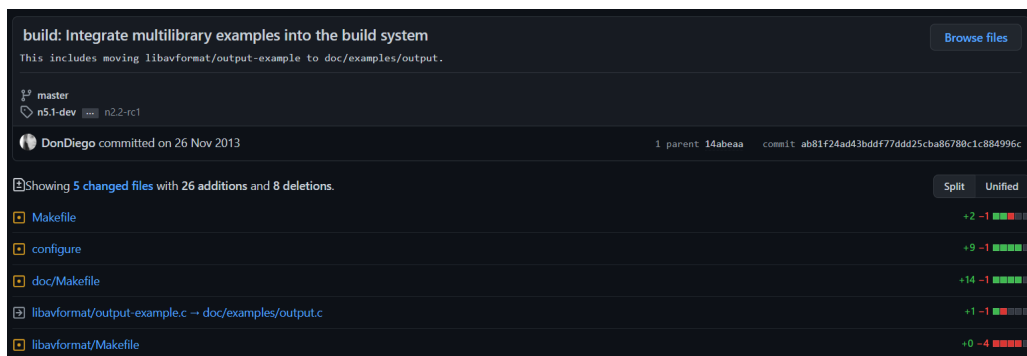


Fig. 5. GitHub 紀錄

Git Commit 滾動指令

```

1 git reset --hard HEAD^
2 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。

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

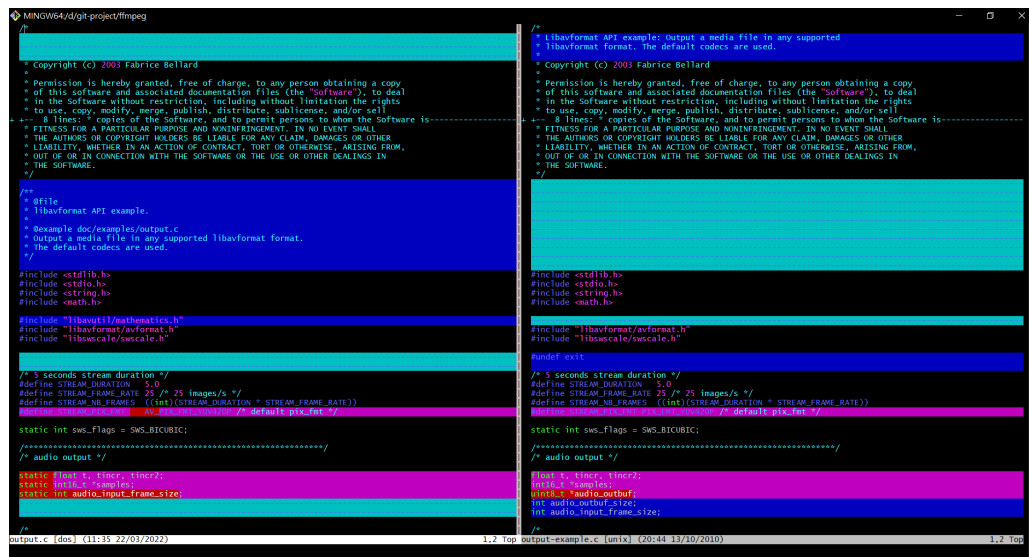


Fig. 6. Vim 行數對比

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 作業項目

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 源代碼說明

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。

```

1  /*
2  * Copyright (c) 2003 Fabrice Bellard
3  *
4  * Permission is hereby granted, free of charge, to any person obtaining a copy
5  * of this software and associated documentation files (the "Software"), to deal
6  * in the Software without restriction, including without limitation the rights
7  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
8  * copies of the Software, and to permit persons to whom the Software is
9  * furnished to do so, subject to the following conditions:
10 *
11 * The above copyright notice and this permission notice shall be included in
12 * all copies or substantial portions of the Software.
13 *
14 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
15 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
16 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
17 * THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

```

```

18  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
19  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
20  * THE SOFTWARE.
21  */
22
23 /**
24  * @file
25  * libavformat API example.
26  *
27  * @example doc/examples/output.c
28  * Output a media file in any supported libavformat format.
29  * The default codecs are used.
30  */

```

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

```

1  // C Library
2
3  #include <stdlib.h>
4  #include <stdio.h>
5  #include <string.h>
6  #include <math.h>
7
8  // FFMPEG Library
9  #include "libavutil/mathematics.h"
10 #include "libavformat/avformat.h"
11 #include "libswscale/swscale.h"
12
13 // 巨集定義
14 /* 5 seconds stream duration */
15 #define STREAM_DURATION 5.0
16 #define STREAM_FRAME_RATE 25 /* 25 images/s */
17 #define STREAM_NB_FRAMES ((int)(STREAM_DURATION * STREAM_FRAME_RATE))
18 #define STREAM_PIX_FMT AV_PIX_FMT_YUV420P /* default pix_fmt */
19
20 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。

```

1  // 音源輸出
2  static float t, tincr, tincr2;
3  static int16_t *samples;
4  static int audio_input_frame_size;
5
6  /* 加入音源輸出
7  * add an audio output stream

```



```

8  */
9  static AVStream *add_audio_stream(AVFormatContext *oc, enum AVCodecID codec_id)
10 {
11     AVCodecContext *c;
12     AVStream *st;
13     AVCodec *codec;
14 // 找到音頻編碼器
15     /* find the audio encoder */
16     codec = avcodec_find_encoder(codec_id);
17     if (!codec) {
18         fprintf(stderr, "codec not found\n");
19         exit(1);
20     }
21
22     st = avformat_new_stream(oc, codec);
23     if (!st) {
24         fprintf(stderr, "Could not alloc stream\n");
25         exit(1);
26     }
27
28     c = st->codec;
29
30 // 放樣本參數
31     /* put sample parameters */
32     c->sample_fmt = AV_SAMPLE_FMT_S16;
33     c->bit_rate = 64000;
34     c->sample_rate = 44100;
35     c->channels = 2;
36
37 // 某些格式希望 stream header 是分開的
38     /* some formats want stream headers to be separate
39     if (oc->oformat->flags & AVFMT_GLOBALHEADER)
40         c->flags |= CODEC_FLAG_GLOBAL_HEADER;
41
42     return st;
43 }

```

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。

```

1  static void open_audio(AVFormatContext *oc, AVStream *st)
2  {
3      AVCodecContext *c;
4
5      c = st->codec;
6
7      /* open it */
8      if (avcodec_open2(c, NULL, NULL) < 0) {

```

```

9         fprintf(stderr, "could not open codec\n");
10        exit(1);
11    }
12
13    /* init signal generator */
14    t      = 0;
15    tincr  = 2 * M_PI * 110.0 / c->sample_rate;
16    /* increment frequency by 110 Hz per second */
17    tincr2 = 2 * M_PI * 110.0 / c->sample_rate / c->sample_rate;
18
19    if (c->codec->capabilities & CODEC_CAP_VARIABLE_FRAME_SIZE)
20        audio_input_frame_size = 10000;
21    else
22        audio_input_frame_size = c->frame_size;
23    samples = av_malloc(audio_input_frame_size *
24                        av_get_bytes_per_sample(c->sample_fmt) *
25                        c->channels);
26 }

```

get\_audio\_frame 準備 “frame\_size” 樣本的 16 位虛擬音頻幀和 ‘nb\_channels’ 頻道。

```

1 static void get_audio_frame(int16_t *samples, int frame_size, int nb_channels)
2 {
3     int j, i, v;
4     int16_t *q;
5
6     q = samples;
7     for (j = 0; j < frame_size; j++) {
8         v = (int)(sin(t) * 10000);
9         for (i = 0; i < nb_channels; i++)
10             *q++ = v;
11         t      += tincr;
12         tincr += tincr2;
13     }
14 }

```

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。

```

1 static void write_audio_frame(AVFormatContext *oc, AVStream *st)
2 {
3     AVCodecContext *c;
4     AVPacket pkt = { 0 }; // data and size must be 0;
5     AVFrame *frame = av_frame_alloc();
6     int got_packet;
7
8     av_init_packet(&pkt);
9     c = st->codec;
10
11     get_audio_frame(samples, audio_input_frame_size, c->channels);

```

```

12     frame->nb_samples = audio_input_frame_size;
13     avcodec_fill_audio_frame(frame, c->channels, c->sample_fmt,
14                             (uint8_t *)samples,
15                             audio_input_frame_size *
16                             av_get_bytes_per_sample(c->sample_fmt) *
17                             c->channels, 1);
18
19     avcodec_encode_audio2(c, &pkt, frame, &got_packet);
20     if (!got_packet)
21         return;
22
23     pkt.stream_index = st->index;
24
25     /* Write the compressed frame to the media file. */
26     if (av_interleaved_write_frame(oc, &pkt) != 0) {
27         fprintf(stderr, "Error while writing audio frame\n");
28         exit(1);
29     }
30     avcodec_free_frame(&frame);
31 }

```

close\_audio 為關閉音源。

```

1 static void close_audio(AVFormatContext *oc, AVStream *st)
2 {
3     avcodec_close(st->codec);
4
5     av_free(samples);
6 }

```

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。

```

1 static AVFrame *picture, *tmp_picture;
2 static int frame_count;
3 static AVStream *add_video_stream(AVFormatContext *oc, enum AVCodecID codec_id)
4 {
5     AVCodecContext *c;
6     AVStream *st;
7     AVCodec *codec;
8     codec = avcodec_find_encoder(codec_id);
9     if (!codec) {
10         fprintf(stderr, "codec not found\n");
11         exit(1);
12     }
13     st = avformat_new_stream(oc, codec);
14     if (!st) {

```

```

15     fprintf(stderr, "Could not alloc stream\n");
16     exit(1);
17 }
18 c = st->codec;
19 c->bit_rate = 400000;
20 c->width = 352;
21 c->height = 288;
22 c->time_base.den = STREAM_FRAME_RATE;
23 c->time_base.num = 1;
24 c->gop_size = 12;
25 c->pix_fmt = STREAM_PIX_FMT;
26 if (c->codec_id == AV_CODEC_ID_MPEG2VIDEO) {
27     c->max_b_frames = 2;
28 }
29 if (c->codec_id == AV_CODEC_ID_MPEG1VIDEO) {
30     c->mb_decision = 2;
31 }
32 if (oc->oformat->flags & AVFMT_GLOBALHEADER)
33     c->flags |= CODEC_FLAG_GLOBAL_HEADER;
34 return st;
35 }

```

AVFrame 為處理 AVCodecContext 的重要組成部分。

```

1 static AVFrame *alloc_picture(enum AVPixelFormat pix_fmt, int width, int height)
2 {
3     AVFrame *picture;
4     uint8_t *picture_buf;
5     int size;
6
7     picture = av_frame_alloc();
8     if (!picture)
9         return NULL;
10    size = avpicture_get_size(pix_fmt, width, height);
11    picture_buf = av_malloc(size);
12    if (!picture_buf) {
13        av_free(picture);
14        return NULL;
15    }
16    avpicture_fill((AVPicture *)picture, picture_buf,
17                  pix_fmt, width, height);
18    return picture;
19 }

```

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。

```

1 static void open_video(AVFormatContext *oc, AVStream *st)
2 {

```

```

3     AVCodecContext *c;
4     c = st->codec;
5     if (avcodec_open2(c, NULL, NULL) < 0) {
6         fprintf(stderr, "could not open codec\n");
7         exit(1);
8     }
9     picture = alloc_picture(c->pix_fmt, c->width, c->height);
10    if (!picture) {
11        fprintf(stderr, "Could not allocate picture\n");
12        exit(1);
13    }
14    tmp_picture = NULL;
15    if (c->pix_fmt != AV_PIX_FMT_YUV420P) {
16        tmp_picture = alloc_picture(AV_PIX_FMT_YUV420P, c->width, c->height);
17        if (!tmp_picture) {
18            fprintf(stderr, "Could not allocate temporary picture\n");
19            exit(1);
20        }
21    }
22 }

```

fill\_yuv\_image 的函式為 dummy image 的處理，當中控制 Y、Cb、Cr。

```

1 static void fill_yuv_image(AVFrame *pict, int frame_index,
2                             int width, int height)
3 {
4     int x, y, i;
5     i = frame_index;
6     for (y = 0; y < height; y++)
7         for (x = 0; x < width; x++)
8             pict->data[0][y * pict->linesize[0] + x] = x + y + i * 3;
9     for (y = 0; y < height / 2; y++) {
10        for (x = 0; x < width / 2; x++) {
11            pict->data[1][y * pict->linesize[1] + x] = 128 + y + i * 2;
12            pict->data[2][y * pict->linesize[2] + x] = 64 + x + i * 5;
13        }
14    }
15 }

```

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。

```

1 static void write_video_frame(AVFormatContext *oc, AVStream *st)
2 {
3     int ret;
4     AVCodecContext *c;
5     static struct SwsContext *img_convert_ctx;
6

```

```

7      c = st->codec;
8
9      if (frame_count >= STREAM_NB_FRAMES) {
10     } else {
11         if (c->pix_fmt != AV_PIX_FMT_YUV420P) {
12             if (img_convert_ctx == NULL) {
13                 img_convert_ctx = sws_getContext(c->width, c->height,
14                                                  AV_PIX_FMT_YUV420P,
15                                                  c->width, c->height,
16                                                  c->pix_fmt,
17                                                  sws_flags, NULL, NULL, NULL);
18
19                 if (img_convert_ctx == NULL) {
20                     fprintf(stderr,
21                             "Cannot initialize the conversion context\n");
22                     exit(1);
23                 }
24             }
25             fill_yuv_image(tmp_picture, frame_count, c->width, c->height);
26             sws_scale(img_convert_ctx, tmp_picture->data, tmp_picture->linesize,
27                      0, c->height, picture->data, picture->linesize);
28         } else {
29             fill_yuv_image(picture, frame_count, c->width, c->height);
30         }
31     }
32
33     if (oc->oformat->flags & AVFMT_RAWPICTURE) {
34         AVPacket pkt;
35         av_init_packet(&pkt);
36         pkt.flags |= AV_PKT_FLAG_KEY;
37         pkt.stream_index = st->index;
38         pkt.data = (uint8_t *)picture;
39         pkt.size = sizeof(AVPicture);
40         ret = av_interleaved_write_frame(oc, &pkt);
41     } else {
42         AVPacket pkt = { 0 };
43         int got_packet;
44         av_init_packet(&pkt);
45         ret = avcodec_encode_video2(c, &pkt, picture, &got_packet);
46         if (!ret && got_packet && pkt.size) {
47             if (pkt.pts != AV_NOPTS_VALUE) {
48                 pkt.pts = av_rescale_q(pkt.pts,
49                                       c->time_base, st->time_base);
50             }
51             if (pkt.dts != AV_NOPTS_VALUE) {
52                 pkt.dts = av_rescale_q(pkt.dts,
53                                       c->time_base, st->time_base);
54             }
55         }
56     }
57 }

```

```

54         pkt.stream_index = st->index;
55         ret = av_interleaved_write_frame(oc, &pkt);
56     } else {
57         ret = 0;
58     }
59 }
60 if (ret != 0) {
61     fprintf(stderr, "Error while writing video frame\n");
62     exit(1);
63 }
64 frame_count++;
65 }

```

close\_video 關閉影像函式。

```

1 static void close_video(AVFormatContext *oc, AVStream *st)
2 {
3     avcodec_close(st->codec);
4     av_free(picture->data[0]);
5     av_free(picture);
6     if (tmp_picture) {
7         av_free(tmp_picture->data[0]);
8         av_free(tmp_picture);
9     }
10 }

```

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。

```

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

```

```

16         return 1;
17     }
18
19     filename = argv[1];
20     fmt = av_guess_format(NULL, filename, NULL);
21     if (!fmt) {
22         printf("Could not deduce output format from file extension: using MPEG.\n");
23         fmt = av_guess_format("mpeg", NULL, NULL);
24     }
25     if (!fmt) {
26         fprintf(stderr, "Could not find suitable output format\n");
27         return 1;
28     }
29     oc = avformat_alloc_context();
30     if (!oc) {
31         fprintf(stderr, "Memory error\n");
32         return 1;
33     }
34     oc->oformat = fmt;
35     snprintf(oc->filename, sizeof(oc->filename), "%s", filename);
36     video_st = NULL;
37     audio_st = NULL;
38     if (fmt->video_codec != AV_CODEC_ID_NONE) {
39         video_st = add_video_stream(oc, fmt->video_codec);
40     }
41     if (fmt->audio_codec != AV_CODEC_ID_NONE) {
42         audio_st = add_audio_stream(oc, fmt->audio_codec);
43     }
44     if (video_st)
45         open_video(oc, video_st);
46     if (audio_st)
47         open_audio(oc, audio_st);
48
49     av_dump_format(oc, 0, filename, 1);
50     if (!(fmt->flags & AVFMT_NOFILE)) {
51         if (avio_open(&oc->pb, filename, AVIO_FLAG_WRITE) < 0) {
52             fprintf(stderr, "Could not open '%s'\n", filename);
53             return 1;
54         }
55     }
56     avformat_write_header(oc, NULL);
57     for (;;) {
58         if (audio_st)
59             audio_pts = (double)audio_st->pts.val * audio_st->time_base.num /
60                 audio_st->time_base.den;
61         else

```



```

61         audio_pts = 0.0;
62
63         if (video_st)
64             video_pts = (double)video_st->pts.val * video_st->time_base.num /
65                         video_st->time_base.den;
66         else
67             video_pts = 0.0;
68         if ((!audio_st || audio_pts >= STREAM_DURATION) &&
69             (!video_st || video_pts >= STREAM_DURATION))
70             break;
71         if (!video_st || (video_st && audio_st && audio_pts < video_pts)) {
72             write_audio_frame(oc, audio_st);
73         } else {
74             write_video_frame(oc, video_st);
75         }
76     }
77     av_write_trailer(oc);
78     if (video_st)
79         close_video(oc, video_st);
80     if (audio_st)
81         close_audio(oc, audio_st);
82     for (i = 0; i < oc->nb_streams; i++) {
83         av_freep(&oc->streams[i]->codec);
84         av_freep(&oc->streams[i]);
85     }
86     if (!(fmt->flags & AVFMT_NOFILE))
87         avio_close(oc->pb);
88     av_free(oc);
89     return 0;
90 }

```

### 5.3 output-example 測試工作

在此使用 GCC 進行編譯，並針對 ffmpeg 0.5.13 的版本進行測試。架設過去的 Linux 跟 GCC 按照當時的技術文件進行還原。

```

1  # 下載早期版本
2  # wget http://www.ffmpeg.org/releases/ffmpeg-0.5.13.tar.bz2
3
4  # 解壓縮
5  # tar -jxvf ffmpeg-0.5.13.tar.bz2
6
7  # vim ffmpeg_configure.sh
8  ./configure \
9  --prefix=/YOUR_INSTLL_DIRECTORY \
10 --enable-gpl --enable-nonfree --enable-version3 \
11 --enable-swscale --enable-avfilter \

```

```
12 --enable-pthreads
13 【保存并退出】
```

進入目錄用 vim 建立 ffmpeg\_configure.sh ，完成後退出。

```
1 # vim ffmpeg_configure.sh
```

```
1 ./configure \
2 --prefix=/YOUR_INSTLL_DIRECTORY \
3 --enable-gpl --enable-nonfree --enable-version3 \
4 --enable-swscale --enable-avfilter \
5 --enable-pthreads
```

改變權限後用 make

```
1 # chmod +x ffmpeg_configure.sh
2 # ./ffmpeg_configure.sh
3 # make
4 # make install
```

將安裝目錄 “” 下的

YOUR\_INSTLL\_DIRECTORY/include YOUR\_INSTLL\_DIRECTORY/lib

打包，就可以生成一個開發用的 SDK

```
1 # tar -czvf YOUR_INSTLL_DIRECTORY/ ffmpeg_sdk.tar.gz
```

引入後編譯，解決：

```
1 # gcc output_example.c -o output_example
2 -I/opt/ffmpeg/sourcecode/ffmpeg-0.5.13.install/include
3 -L/opt/ffmpeg/sourcecode/ffmpeg-0.5.13.install/lib
4 -lavformat -lavdevice -lavcodec -lavutil -lavfilter
5 -pthread -ldl -lswscale -lbz2 -lasound -lz -lm
```

運行

```
1 # ./output_example xxx.flv
```

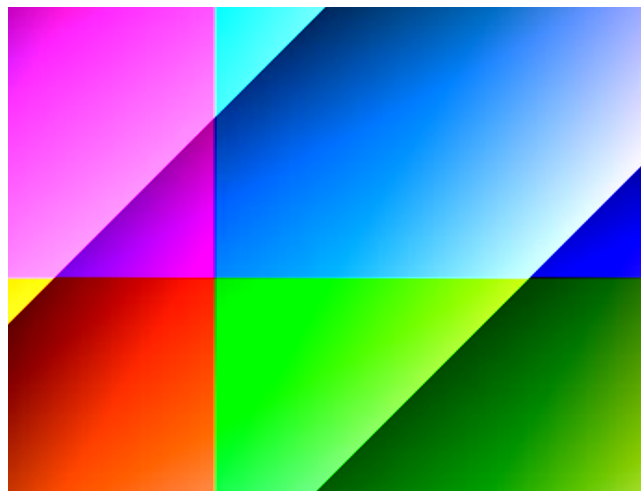


Fig. 7. output-example 測試結果

## 6 muxing.c 測試工作

後來經過調研才發現類似的 Code 於此，其測試內容於之前雷同，但是這回使用 Mac。

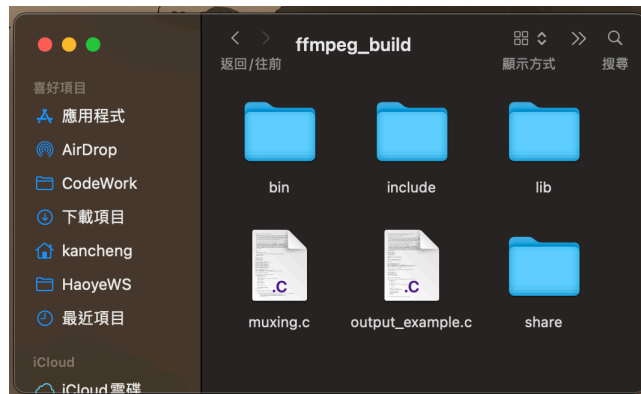


Fig. 8. muxing.c 目錄

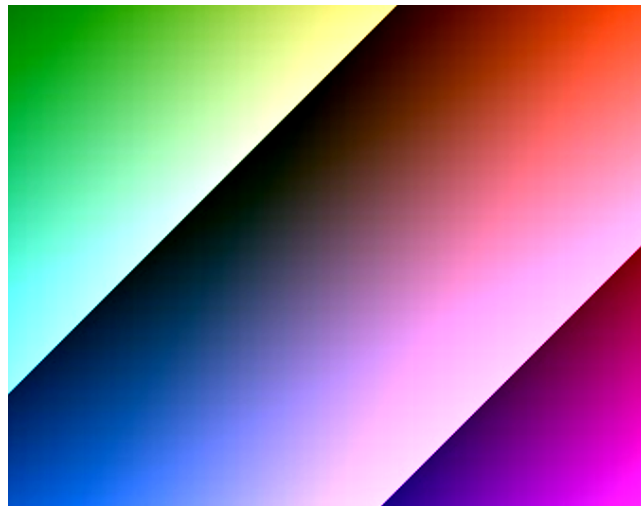


Fig. 9. muxing.c 測試結果