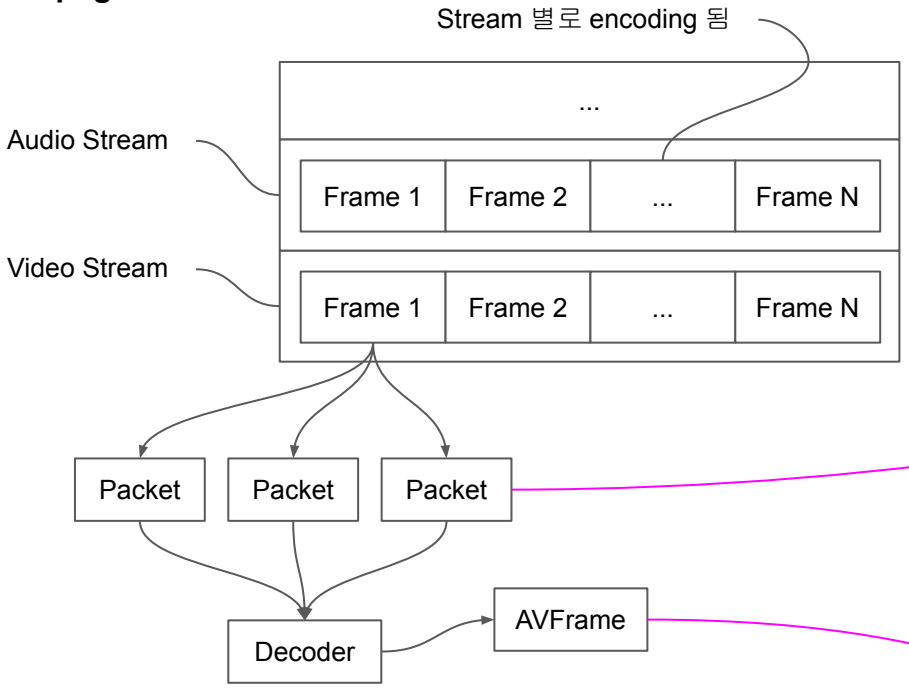
Video Player

FFmepg (release/7.1)

FFmpeg 구조



1

```
// AVPacket 획득
AVFormatContext *ic = avformat_alloc_context();
AVPacket *pkt = av_packet_alloc();

avformat_open_input(&ic, is->filename,
is->iformat/*NULL*/, &format_opts);
→ format_opts 에는 "scan_all_pmts", "1" 1쌍 들어 있음

av_read_frame(ic,pkt);
```

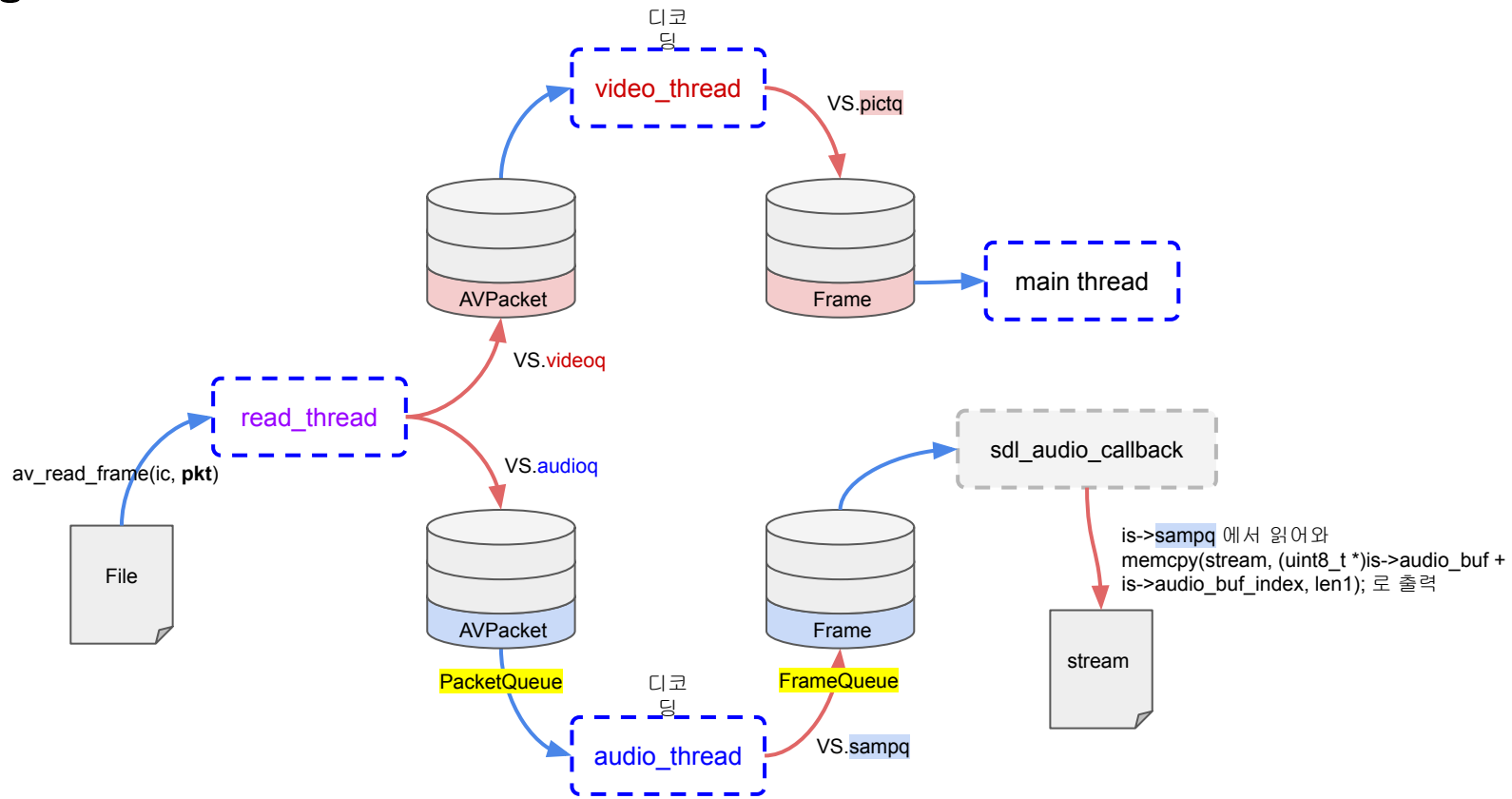
2

```
// AVFrame 획득
avcodec_send_packet(d->avctx,d->pkt);
AVFrame *frame = av_frame_alloc();
avcodec_receive_frame(d->avctx,frame);
```

3

1) 비디오의 경우, AVFrame::data 로 텍스쳐를 생성해 화면 출력
2) 오디오의 경우, AVFrame::data[0] 로 오디오 출력

Thread 구성


디코딩
video_thread
VS.pictq
AVPacket
Frame
main thread
VS.videoq
read_thread
av_read_frame(ic, pkt)
VS.audioq
sdl_audio_callback
is->sampq 에서 읽어와
memcpy(stream, (uint8_t *)is->audio_buf +
is->audio_buf_index, len1); 로 출력
File
AVPacket
Frame
stream
PacketQueue
디코딩
FrameQueue
audio_thread
VS.sampq

main()
> VideoState *is = stream_open(input_filename, file_iformat);
> **event_loop**(is);

void **event_loop**(VideoState *cur_stream)
> for (;;) refresh_loop_wait_event(cur_stream, &event);

void **refresh_loop_wait_event**(VideoState *is, SDL_Event *event)
> video_refresh(is, &remaining_time);
>> video_display(is);
>>> video_image_display(is); OR video_audio_display(is);

void **video_image_display**(VideoState *is)
> vp = frame_queue_peek_last(&is->pictq);
> SDL_RenderCopyEx(renderer, is->vid_texture, ...)

video_thread

VS.pictq

AVPacket

Frame

main thread

VS.videoq

read_thread

VS.audioq

av_read_frame(ic, **pkt**)

File

AVPacket

Frame

sdl_audio_callback

PacketQueue

FrameQueue

audio_thread

VS.sampq

stream

[video_thread]
AVFrame *frame = av_frame_alloc();
ret = get_video_frame(is, frame);
> **decoder_decode_frame**(&is->viddec, frame, NULL)
av_buffersrc_add_frame(filt_in, frame);
av_buffersink_get_frame_flags(filt_out, frame, 0); // 필터 결과
queue_picture(is, frame, pts, duration, frame->pkt_pos,
is->viddec.pkt_serial);
> frame_queue_push(&is->pictq); 로 frame 를 is->pictq에 추가!!!

int **decoder_decode_frame**(Decoder *d,
AVFrame *frame, AVSubtitle *sub)
> avcodec_send_packet(d->avctx, d->pkt)
> ret = avcodec_receive_frame(d->avctx, frame)
로 frame 에 디코딩 & ret(결과)가 리턴!

void sdl_audio_callback(void *opaque, Uint8 *stream, int len)
> 콜백시 stream 주소에 len 만큼 채워줘야함 !!!

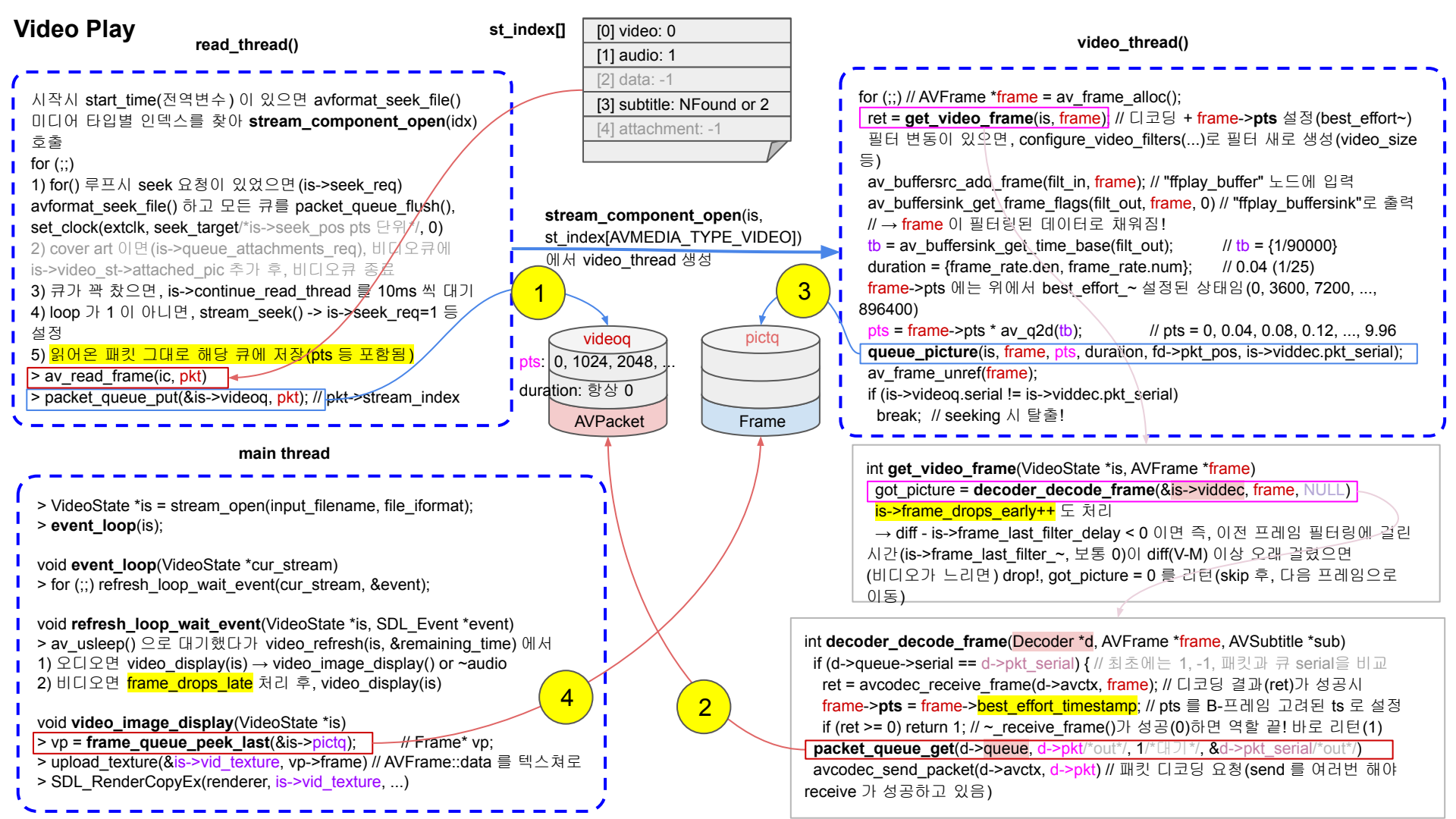
is->sampq 에서 읽어와
memcpy(stream, (uint8_t *)is->audio_buf + is->audio_buf_index, len1); 로 출력

[read_thread]
stream_component_open() 호출해서 video_thread, audio_thread 등등 생성 후,

for (;;) {
ret = av_read_frame(ic, pkt); 로 pkt 을 읽어와
packet_queue_put(&is->videoq, pkt); 등으로
PacketQueue 에 추가(is->audioq 등등)
}

[audio_thread]
AVFrame *frame = av_frame_alloc();
got_frame = **decoder_decode_frame**(&is->auddec, frame, NULL)
reconfigure 여부 체크해서 configure_audio_filters()
av_buffersrc_add_frame(is->in_audio_filter, frame) // 상동
av_buffersink_get_frame_flags(is->out_audio_filter, frame, 0)// 상동
frame_queue_push(&is->sampq); 로 frame 를 is->sampq 에 추가!!!

Video Play


read_thread()
st_index[]
[0] video: 0
[1] audio: 1
[2] data: -1
[3] subtitle: NFound or 2
[4] attachment: -1
video_thread()
시작시 start_time(전역변수) 이 있으면 avformat_seek_file()
미디어 타입별 인덱스를 찾아 stream_component_open(idx) 호출
for (;;)
1) for() 루프시 seek 요청이 있었으면(is->seek_req)
avformat_seek_file() 하고 모든 큐를 packet_queue_flush(),
set_clock(extclk, seek_target/*is->seek_pos pts 단위*/, 0)
2) cover art 이면(is->queue_attachments_req), 비디오큐에 is->video_st->attached_pic 추가 후, 비디오큐 종료
3) 큐가 꽉 찼으면, is->continue_read_thread 를 10ms 씩 대기
4) loop 가 1 이 아니면, stream_seek() -> is->seek_req=1 등 설정
5) 읽어온 패킷 그대로 해당 큐에 저장(pts 등 포함됨)
> av_read_frame(ic, pkt)
> packet_queue_put(&is->videoq, pkt); // pkt->stream_index
stream_component_open(is, st_index[AVMEDIA_TYPE_VIDEO])
에서 video_thread 생성
1
3
videoq
pts: 0, 1024, 2048, ...
duration: 항상 0
AVPacket
pictq
Frame
for (;;) // AVFrame *frame = av_frame_alloc();
ret = get_video_frame(is, frame); // 디코딩 + frame->pts 설정(best_effort~)
필터 변동이 있으면, configure_video_filters(...)로 필터 새로 생성(video_size 등)
av_buffersrc_add_frame(filt_in, frame); // "ffplay_buffer" 노드에 입력
av_buffersink_get_frame_flags(filt_out, frame, 0) // "ffplay_buffersink"로 출력
// → frame 이 필터링된 데이터로 채워짐!
tb = av_buffersink_get_time_base(filt_out); // tb = {1/90000}
duration = {frame_rate.den, frame_rate.num}; // 0.04 (1/25)
frame->pts 에는 위에서 best_effort_~ 설정된 상태임(0, 3600, 7200, ..., 896400)
pts = frame->pts * av_q2d(tb); // pts = 0, 0.04, 0.08, 0.12, ..., 9.96
queue_picture(is, frame, pts, duration, fd->pkt_pos, is->viddec.pkt_serial);
av_frame_unref(frame);
if (is->videoq.serial != is->viddec.pkt_serial)
break; // seeking 시 탈출!
int get_video_frame(VideoState *is, AVFrame *frame)
got_picture = decoder_decode_frame(&is->viddec, frame, NULL)
is->frame_drops_early++ 도 처리
→ diff - is->frame_last_filter_delay < 0 이면 즉, 이전 프레임 필터링에 걸린 시간(is->frame_last_filter_~, 보통 0)이 diff(V-M) 이상 오래 걸렸으면 (비디오가 느리면) drop!, got_picture = 0 를 리턴(skip 후, 다음 프레임으로 이동)
main thread
> VideoState *is = stream_open(input_filename, file_iformat);
> event_loop(is);
void event_loop(VideoState *cur_stream)
> for (;;) refresh_loop_wait_event(cur_stream, &event);
void refresh_loop_wait_event(VideoState *is, SDL_Event *event)
> av_usleep() 으로 대기했다가 video_refresh(is, &remaining_time) 에서
1) 오디오면 video_display(is) → video_image_display() or ~audio
2) 비디오면 frame_drops_late 처리 후, video_display(is)
void video_image_display(VideoState *is)
> vp = frame_queue_peek_last(&is->pictq); // Frame* vp;
> upload_texture(&is->vid_texture, vp->frame) // AVFrame::data 를 텍스쳐로
> SDL_RenderCopyEx(renderer, is->vid_texture, ...)
4
2
int decoder_decode_frame(Decoder *d, AVFrame *frame, AVSubtitle *sub)
if (d->queue->serial == d->pkt_serial) { // 최초에는 1, -1, 패킷과 큐 serial을 비교
ret = avcodec_receive_frame(d->avctx, frame); // 디코딩 결과(ret)가 성공시
frame->pts = frame->best_effort_timestamp; // pts 를 B-프레임 고려된 ts 로 설정
if (ret >= 0) return 1; // ~_receive_frame()가 성공(0)하면 역할 끝! 바로 리턴(1)
packet_queue_get(d->queue, d->pkt/*out*/, 1/*대기*/, &d->pkt_serial/*out*/)
avcodec_send_packet(d->avctx, d->pkt) // 패킷 디코딩 요청(send 를 여러번 해야 receive 가 성공하고 있음)

Audio Play

read_thread()

```
...
5) 읽어온 패킷 그대로 해당 큐에 저장
> av_read_frame(ic, pkt)
> packet_queue_put(&is->audioq, pkt)
```

stream_component_open(is, st_index[AVMEDIA_TYPE_AUDIO]) 에서 audio_thread 생성

audio_thread()

```
do {
  got_frame = decoder_decode_frame(&is->auddec, frame, NULL)
  if (reconfigure) 이면 (설정이 바꼈으면)
    configure_audio_filters(is, afilters, 1) // is->in_audio_filter 등 설정
  av_buffersrc_add_frame(is->in_audio_filter, frame); // "ffplay_abuffer" 에 입력
  av_buffersink_get_frame_flags(filt_out, frame, 0) // "ffplay_abuffersink" 출력

  tb = av_buffersink_get_time_base(is->out_audio_filter); // tb = {1/48000}
  af->pts = frame->pts * av_q2d(tb); // e.g, 1024 / 48000 = 0.021
  // → frame->pts=1024, 2048, ..., 480256, af->pts=0.0213, 0.0427, ..., 10.0053
  af->duration = av_q2d({frame->nb_samples/*1024*/,
frame->sample_rate/*48000*/}); // 0.0213
  av_frame_move_ref(af->frame, frame); // 비디오는 queue_picture()에서
  move
  frame_queue_push(&is->sampq);
  if (is->audioq.serial != is->auddec.pkt_serial)
    break;  // seeking 시 탈출!
} while (ret >= 0 || ret == AVERROR(EAGAIN) || ret == AVERROR_EOF);
```

1

VS.audioq

pts: 0, 1024, 2048, ...

duration: 항상 1024

AVPacket

3

VS.sampq

Frame

sdl_audio_callback thread

```
// 콜백시 stream 주소에 len 만큼 채워줘야함 !!!
void sdl_audio_callback(void *opaque, Uint8 *stream, int len) while (len
> 0) // len 길이만큼 소비
  audio_size = audio_decode_frame(is);
memcpy(stream, (uint8_t *)is->audio_buf + is->audio_buf_index, len1);
// → 디바이스에 출력됨 !!!
is->audio_write_buf_size = is->audio_buf_size - is->audio_buf_index;
```

```
int decoder_decode_frame(Decoder *d, AVFrame *frame, AVSubtitle *sub)
  if (d->queue->serial == d->pkt_serial) { // 최초에는 1, -1, 패킷과 큐 serial을 비교
    ret = avcodec_receive_frame(d->avctx, frame); // 전송한 d->pkt 에 대한 디코딩 결과
    AVRational tb = (AVRational){1, frame->sample_rate}; // {1, 48000}
    // frame->pts = 1024, 2048, ...
    frame->pts = av_rescale_q(frame->pts, d->avctx->pkt_timebase/*tb와 동일*/, tb);
    if (ret >= 0) return 1; // avcodec_receive_frame() 가 성공시 0 이라서 1이 리턴됨!
  packet_queue_get(d->queue, d->pkt/*out*/, 1/*대기*/, &d->pkt_serial/*out*/)
  avcodec_send_packet(d->avctx, d->pkt)
```

2

memcpy(stream, (uint8_t *)is->audio_buf + is->audio_buf_index, len1);

stream

4

Audio Sync

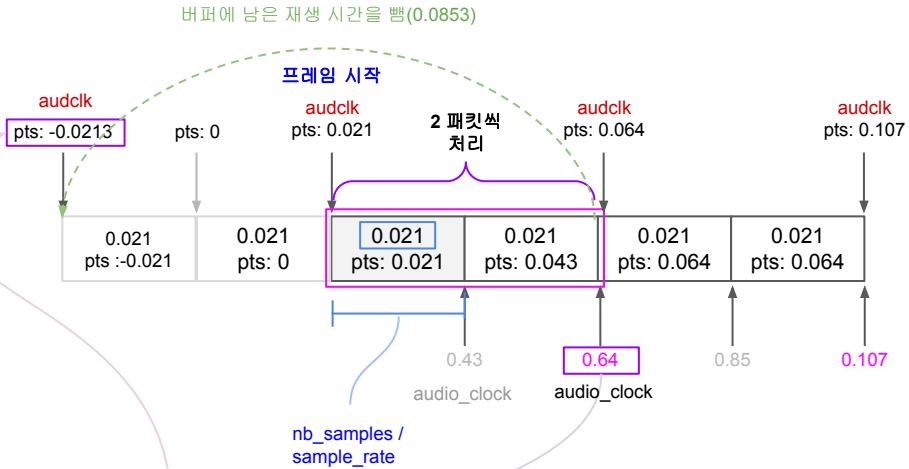
audclk 는 -0.0213 부터 0.043 씩 일정하게 증가함!!!

```
int audio_decode_frame(VideoState *is)
>
af = frame_queue_peek_readable(&is->sampq)
// audio_clock 에는 현재 프레임의 끝 시간을 저장
is->audio_clock = af->pts + (double) af->frame->nb_samples/*1024*/ /
af->frame->sample_rate/*48000*/; // 1024 / 48000 = 0.0213
오디오 리샘플링해서 is->audio_buf 에 저장

printf("audio: delay=%0.3f clock=%0.3f clock0=%0.3f\n",
      is->audio_clock - last_clock, // 현재 last_clock 와 audio_clock0 는 항상 같은 값
      is->audio_clock, audio_clock0); last_clock = is->audio_clock;
위 로그에서 delay 는 is->audio_clock - last_clock
--> 0.043(0.021 + 0.021 - 0), 0.021(0.043 + 0.021 - 0.043), 0.021(0.064 + 0.021 - 0.043),
```

```
audio: delay = 0.043 clock = 0.043 clock0 = 0.000 // 처음에만 delay = 0.043 로 시작, 이후 0.021
audio: delay = 0.021 clock = 0.064 clock0 = 0.043
audio: delay = 0.021 clock = 0.085 clock0 = 0.064
...
audio: delay = 0.021 clock = 10.005 clock0 = 9.984 // 마지막 pts 는 10.0053 였음
audio: delay = 0.021 clock = 10.027 clock0 = 10.005 // 재생 끝 시각은 0.021 더해서 10.027 로
스트림의 duration 값과 일치함
```

is->audio_clock 을 설정

```
void sdl_audio_callback(void *opaque, Uint8 *stream, int len/*보통 8192*/)
>
audio_size = audio_decode_frame(is);    // 2번씩 호출됨(nb_samples(1024) * 4 씩)
is->audio_buf 를 실제 디바이스로 출력 후, audclk 설정!!!
set_clock_at(&is->audclk,
    is->audio_clock -
    (double)(2 * is->audio_hw_buf_size/*8192*/ + is->audio_write_buf_size/*0*/)
    / is->audio_tgt.bytes_per_sec/*192000*/, // 버퍼에 남은 재생 시간을 뺌(0.0853 고정?!)
    is->audio_clock_serial, audio_callback_time / 1000000.0);
```



pts = audio_clock - 0.0853
= 0.064 - 0.0853
= -0.0213

audio_clock =
pts + nb_samples/sample_rate
= 0.043 + 0.021 = 0.064

```
// sdl_audio_callback() 에서 while (len > 0) 루프를 2번 돌고 나서, is->audclk 설정
audio_clock = 0.064, pts =-0.021, time = 1745807442.982   // 0.064 - 0.0853 = -0.0213
audio_clock = 0.107, pts = 0.021, time = 1745807443.023   // 루프 2번 -> 0.43 씩 증가
audio_clock = 0.149, pts = 0.064, time = 1745807443.062
audio_clock = 0.192, pts = 0.107, time = 1745807443.112
audio_clock = 0.235, pts = 0.149, time = 1745807443.152
...
audio_clock = 10.005, pts = 9.920, ~
audio_clock = 10.027, pts = 9.941, ~
```

video_refresh()

1 아직 frame_timer + delay 이전이면 sleep

vidclk 는 0 부터 0.04 씩 일정하게 증가함!!!

time: wall clock 시간
frame_timer: 누적 재생시간
고정크기 (delay) 만큼씩 증가

time
(e.g, 1000)

duration(→ delay)

0(40ms) pts: 0	1(40ms) pts: 0.04	2(40ms) pts: 0.08

lastvp
vp

frame_timer
(1000)

display 0

최초에 time
으로 설정

마스터 오디오와
diff 동기화된
delay(≠ 40ms)

*remaining_time

time (1000) | time (...) | time (1040)

최대
REFRESH_RATE(0.01)
이내로 반복 호출
프레임 끝 시간 이내인
경우, goto display

lastvp vp

0 pts: 0	1 pts: 0.04	2 pts: 0.08

frame_timer
(1000)

frame_timer +=
delay
(1040)

display 1

f->size - f->rindex_shown > 0 면
프레임 0 부터 출력

update_video_pts(is,
vp->pts/*0.04터*/, vp->serial/*1*/)
→ is->vidclk 설정

2 wall clock(time) 이 다음
프레임보다도 앞선 경우 drop !!!
(비디오가 느린 경우)

time
(1040)

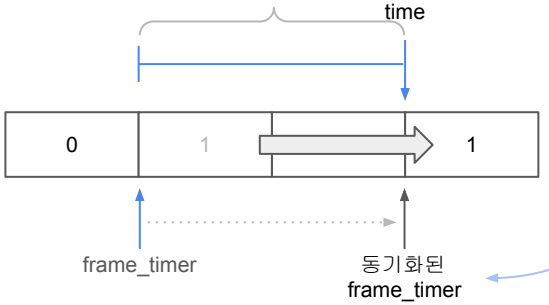
0 40 ms	1 drop	2 40 ms

lastvp vp nextvp

frame_timer
(1040)

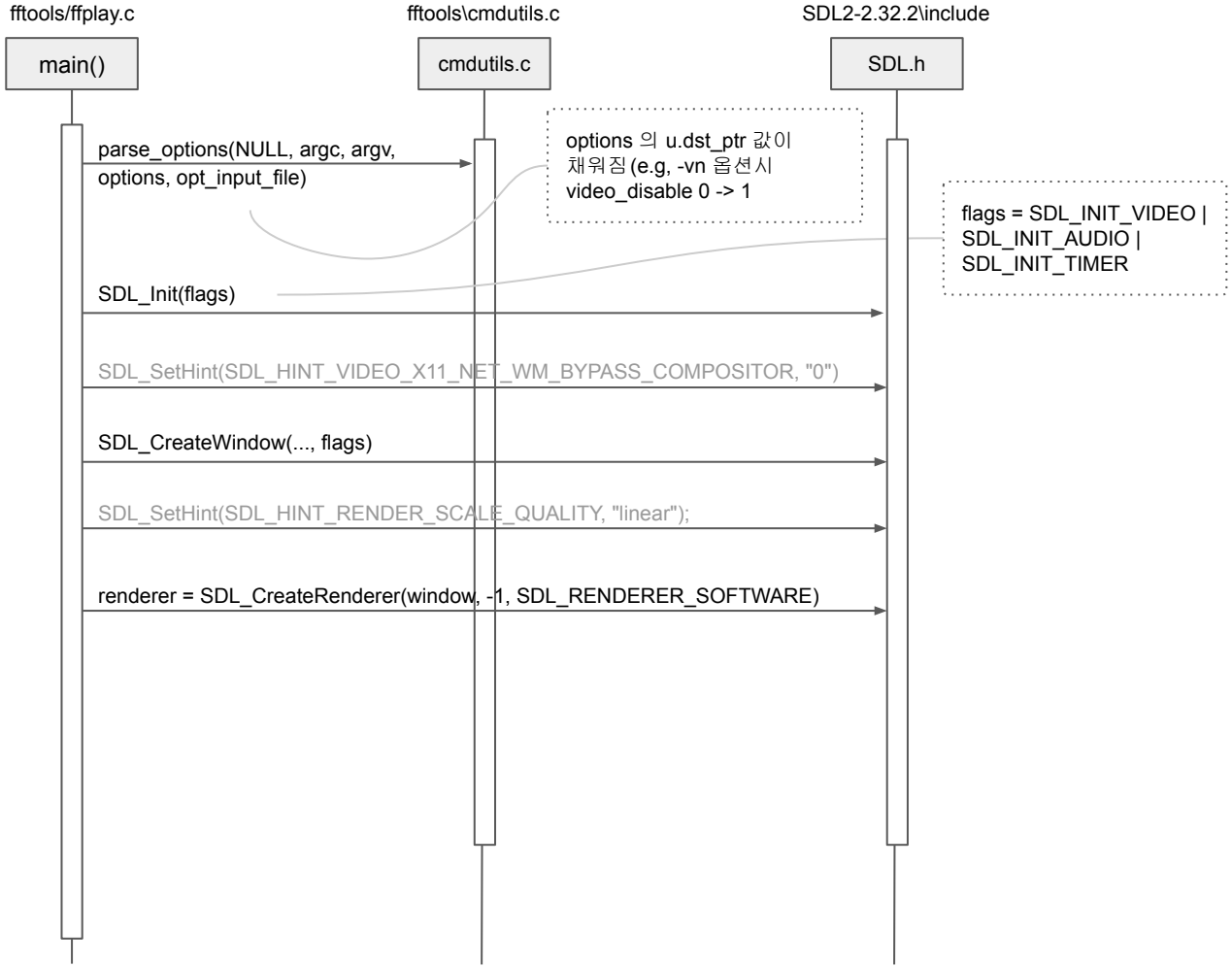
Skip 1

frame_timer 가 time 대비 0.1초를
초과하면 frame_timer = time 로 동기화

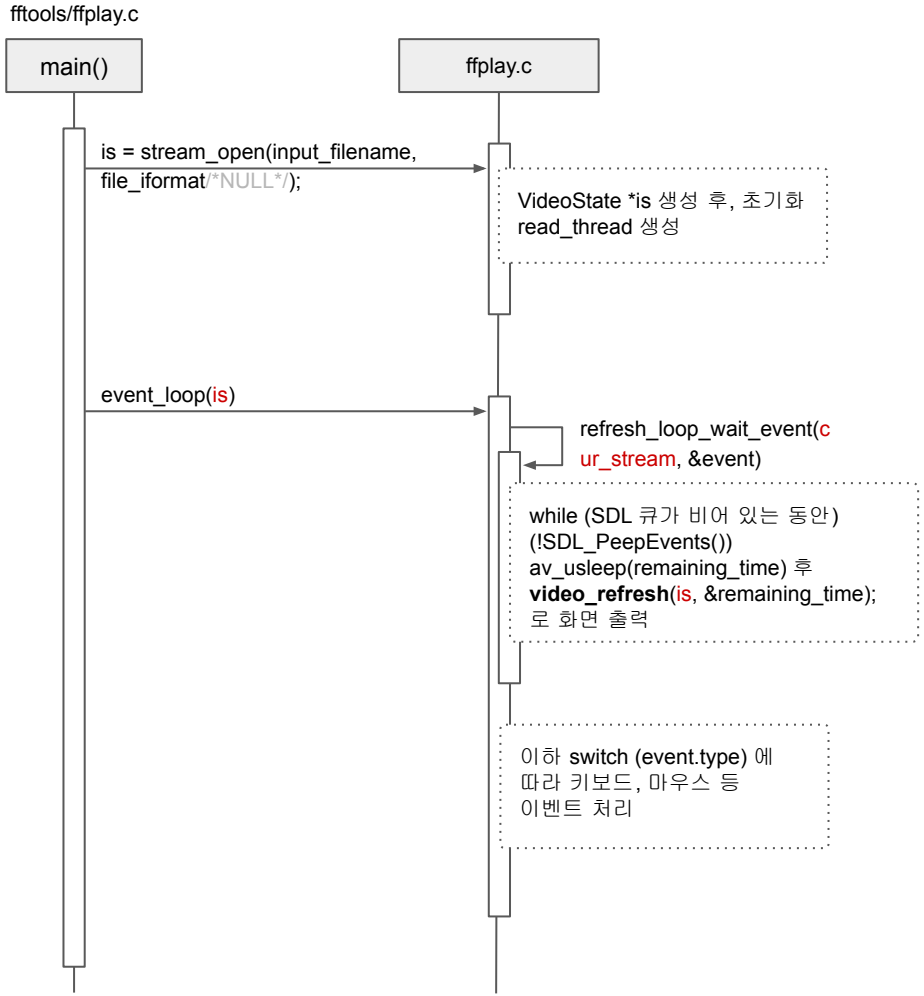


- 최초에 frame_timer 는 time 값(wall clock)보다 작아(0 + delay) time 으로 설정되고, update_video_pts(vp->pts) 와 frame_queue_next(&is->pictq) 후 첫 프레임(0th)을 표시함
- 다음번 진입시 time 이 아직 현재 프레임 끝시간(frame_timer(0) + delay) 이내이면 sleep (refresh_loop_wait_event() 의 while 루프에서 sleep 하면서 반복 호출중)
- 아니면 frame_timer += delay 후 frame_queue_next()
- time 이 다음 프레임 끝 시간을 넘어서면 drop
- frame_timer 는 누적된 재생 시간이며, time 보다 많이 느리면(0.1s 초과) time 으로 설정

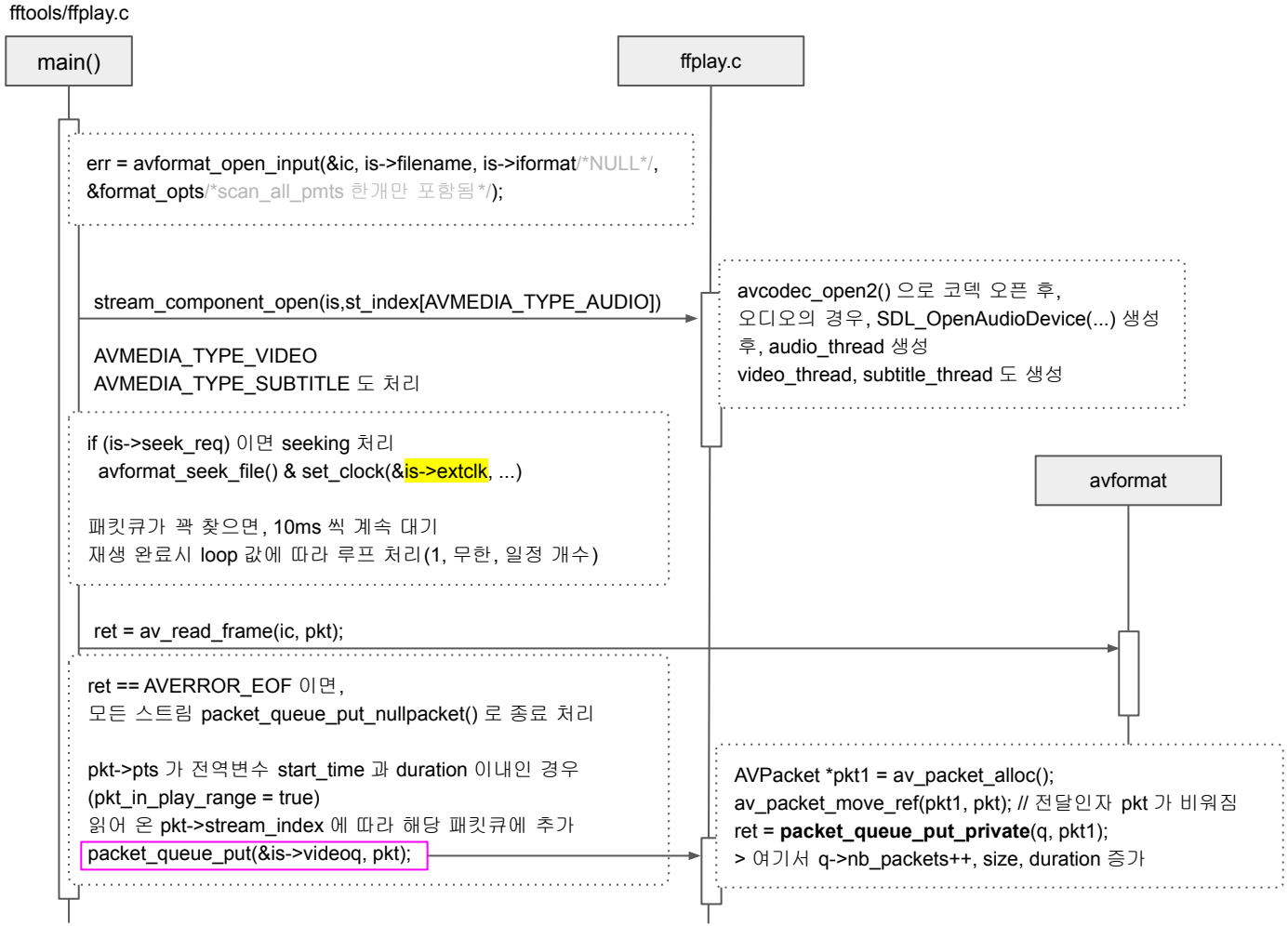
main() -- 초기화


fftools/ffplay.c
fftools\cmdutils.c
SDL2-2.32.2\include
main()
cmdutils.c
SDL.h
parse_options(NULL, argc, argv,
options, opt_input_file)
options 의 u.dst_ptr 값이
채워짐 (e.g, -vn 옵션시
video_disable 0 -> 1
flags = SDL_INIT_VIDEO |
SDL_INIT_AUDIO |
SDL_INIT_TIMER
SDL_Init(flags)
SDL_SetHint(SDL_HINT_VIDEO_X11_NET_WM_BYPASS_COMPOSITOR, "0")
SDL_CreateWindow(..., flags)
SDL_SetHint(SDL_HINT_RENDER_SCALE_QUALITY, "linear");
renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_SOFTWARE)

main()


fftools/ffplay.c
main()
ffplay.c
is = stream_open(input_filename,
file_iformat/*NULL*/);
VideoState *is 생성 후, 초기화
read_thread 생성
event_loop(is)
refresh_loop_wait_event(c
ur_stream, &event)
while (SDL 큐가 비어 있는 동안)
(!SDL_PeepEvents())
av_usleep(remaining_time) 후
video_refresh(is, &remaining_time);
로 화면 출력
이하 switch (event.type) 에
따라 키보드, 마우스 등
이벤트 처리

read_thread()


fftools/ffplay.c
main()
ffplay.c
avformat
err = avformat_open_input(&ic, is->filename, is->iformat/*NULL*/,
&format_opts/*scan_all_pmts 한개만 포함됨*/);
stream_component_open(is,st_index[AVMEDIA_TYPE_AUDIO])
avcodec_open2() 으로 코덱 오픈 후,
오디오의 경우, SDL_OpenAudioDevice(...) 생성
후, audio_thread 생성
video_thread, subtitle_thread 도 생성
AVMEDIA_TYPE_VIDEO
AVMEDIA_TYPE_SUBTITLE 도 처리
if (is->seek_req) 이면 seeking 처리
avformat_seek_file() & set_clock(&is->extclk, ...)
패킷큐가 꽉 찾으면, 10ms 씩 계속 대기
재생 완료시 loop 값에 따라 루프 처리(1, 무한, 일정 개수)
ret = av_read_frame(ic, pkt);
ret == AVERROR_EOF 이면,
모든 스트림 packet_queue_put_nullpacket() 로 종료 처리
pkt->pts 가 전역변수 start_time 과 duration 이내인 경우
(pkt_in_play_range = true)
읽어 온 pkt->stream_index 에 따라 해당 패킷큐에 추가
packet_queue_put(&is->videoq, pkt);
AVPacket *pkt1 = av_packet_alloc();
av_packet_move_ref(pkt1, pkt); // 전달인자 pkt 가 비워짐
ret = packet_queue_put_private(q, pkt1);
> 여기서 q->nb_packets++, size, duration 증가

video_thread()

fftools/ffplay.c

main()

ffplay.c

ffplay.c

```
for (;;)
  ret = get_video_frame(is, frame);
```

```
if ((got_picture = decoder_decode_frame(
  &is->viddec, frame, NULL)) < 0) return -1;
```

```
int decoder_decode_frame(Decoder *d, AVFrame *frame,
AVSubtitle *sub)
>
if (d->queue->abort_request)      return -1;
case AVMEDIA_TYPE_VIDEO: 이면
ret = avcodec_receive_frame(d->avctx, frame);
// frame->pts=0, 3600, 7200, ...
frame->pts = frame->best_effort_timestamp; // 이미 동일하더라

case AVMEDIA_TYPE_AUDIO: 이면
AVRational tb = (AVRational){1, frame->sample_rate/*48000*/};
ret = avcodec_receive_frame(d->avctx, frame);

// frame->pts=1024, 2048, ..., 480256(비율 일정함)
frame->pts = av_rescale_q(frame->pts, d->avctx->pkt_timebase,
tb); // 둘다 48000 로 동일해서 frame->pts 값 변동 없음

d->next_pts = frame->pts + frame->nb_samples/*1024*/;

if (ret == AVERROR_EOF) // 파일 끝에 도착한 경우
    d->finished = d->pkt_serial;
    avcodec_flush_buffers(d->avctx);
    return 0;
if (ret >= 0)             return 1; // 디코딩 성공시 리턴!!!
ret 가 AVERROR(EAGAIN) 이면, 위 루프 계속

이하는 블럭킹 모드로 packet_queue_get() 로 꺼내서
avcodec_send_packet(d->avctx, d->pkt)
만약 이 함수 리턴값이 AVERROR(EAGAIN) 이면 반복 시도
```

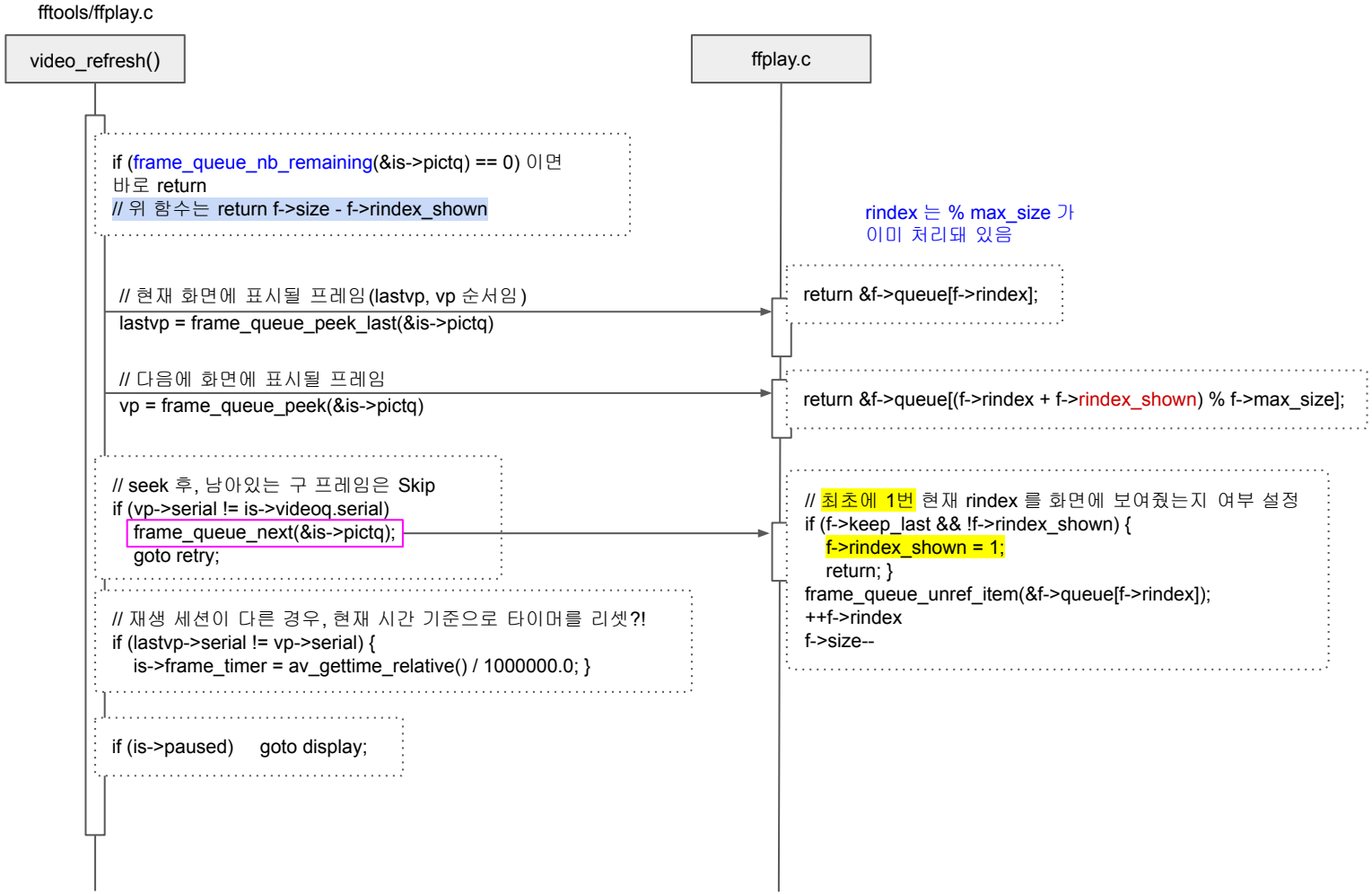
```
frame->format 등이 바뀐 경우,
1) configure_video_filters(graph, is, ...)
> configure_filtergraph(graph, vfilters, filt_src,
last_filter)
2) frame_rate =
av_buffersink_get_frame_rate(filt_out);
```

```
// 디코딩에 성공했으면
if (got_picture)
오디오 마스터인 경우,
이전 프레임 획득이 느렸으면
if 현재 프레임 남은 재생 시간(diff) < 이전
프레임 획득에 걸린 시간
(is->frame_last_filter_delay)
is->frame_drops_early++ // 프레임 드랍
got_picture = 0; 를 리턴
```

```
// frame 은 reference 를 뺏기고 reset 된다.
ret = av_buffersrc_add_frame(filt_in, frame);

while (ret >= 0) {
  is->frame_last_returned_time = av_gettime_relative() / ~;
  // 필터링된 frame 이 채워짐
  ret = av_buffersink_get_frame_flags(filt_out, frame, 0);
  is->frame_last_filter_delay 에 시간 차이 저장

  tb = av_buffersink_get_time_base(filt_out); // tb = 1/90000
  duration = {1, 25} // 0.04

  // frame->pts = 0, 3600, 7200, 10800, ..., 896400
  pts = frame->pts * av_q2d(tb);
  // pts = 0, 0.04, 0.08, 0.12, ..., 9.96 초
}
ret = queue_picture(is, frame, pts, duration, fd->pkt_pos : -1,
is->viddec.pkt_serial);
```

```
vp = frame_queue_peek_writable(&is->pictq)
> f->size >= f->max_size 이면 대기했다가
> return &f->queue[f->windex];
vp->pts 등등 설정 후,
frame_queue_push(&is->pictq); 로 추가
(++f->windex)
```

video_refresh() -- called from main thread



video_refresh()

fftools/ffplay.c

video_refresh()

ffplay.c

// 재생 지속 시간
last_duration = vp_duration(is, lastvp, vp); // 보통 0.04

nextvp->pts - vp->pts 가 유효하면 이를 리턴. 아니면 return vp->duration;

// 마스터 클럭과 동기화
delay = **compute_target_delay**(last_duration, is);

마스터 싱크가 비디오이면 return, 아니면(오디오 싱크) diff(비디오 - 마스터) 저장 후
sync_threshold = FFMAX(AV_SYNC_THRESHOLD_MIN/*0.04*/, FFMIN(~_MAX/*0.1*/, delay))

1) 비디오(↓)가 (0.04 ~ 0.1) 이상 느리면
if (diff <= -sync_threshold)
  delay = FFMAX(0, delay + diff); // delay(재생 지속시간)를 diff 만큼 줄임(빠르게 재생)

2) 비디오(↑)가 (0.04 ~ 0.1) 이상 빠르며, delay 가 0.1 초 이상 넉넉한 경우, diff 만큼 늘려 느리게 재생
else if (diff >= sync_threshold && delay > AV_SYNC_FRAMEDUP_THRESHOLD/*0.1*/) {
  delay = delay + diff; // delay 를 diff 만큼 키움(느리게 재생)
  // e.g, delay(0.15) = **delay(0.11)** + diff(최소 0.04), delay(100.11) = **delay(0.11)** + diff(최대 eg,100)

3) duration(=delay) 이 충분하지 않으면, **2배만 느리게 재생**
else if (diff >= sync_threshold) delay = 2 * delay; // e.g, delay(0.2) = 2 * **delay(0.1)**
return delay;

time= av_gettime_relative()/1000000.0; // 현재 시간 저장(초단위)
if (time < is->frame_timer + delay)
  *remaining_time = FFMIN(is->frame_timer + delay - time, *remaining_time);
  goto display; // 현재 시간이 느릴 경우, sleep 으로 대기!
is->frame_timer += delay; // 총 재생한 시간 저장

// 시간차가 0.1 초 이상 크면, frame_timer = time 으로 강제 싱크
if (delay > 0 && time - is->frame_timer > AV_SYNC_THRESHOLD_MAX/*0.1*/)
  is->frame_timer = time;

// 마지막 프레임의 vp->pts(시작시간) 를 vidclk 에 설정 & 이어서 출력
update_video_pts(is, vp->pts, vp->serial); // vp->pts = 0, 0.04, ...

void **update_video_pts**(VideoState *is, double pts, int serial)
> set_clock(&is->vidclk, pts, serial); // 설정 위치는 거의 여기가 유일!!! (stream_toggle_pause() 도)
> sync_clock_to_slave(&is->extclk, &is->vidclk);

// time 이 다음 프레임의 재생시간을 넘어선 경우, 현재 프레임 drop!!!
if (... && time > is->frame_timer + duration)
  is->frame_drops_late++; // 프레임 드랍
  frame_queue_next(&is->pictq); goto retry;

- video_refresh() 시의 Frame::pts는(e.g, vp->pts) 디코딩시의 frame->pts = 0, 3600, 7200, ..., 896400 값에 av_q2d(tb) = (1 / 90000) 를 곱한 값(tb 는 av_buffersink_get_time_base(filt_out) 값임)인 pts 이며, 디코딩시 pts = 0, 0.04, ... 로 queue_picture() 함수의 pts 에 전달됐었으며, 여기서는 이 전달된 값이 사용되고 있음

자막이 있는 경우(is->subtitle_st), 시킹이 됐거나 비디오가 현재 자막 끝 위치를 지났거나, 다음 자막 시작 위치를 지났거나 하면
현재 자막 이미지(is->sub_texture) 영역을 투명하게 채운 후, 다음 프레임으로 이동

frame_queue_next(&is->pictq); // 여기서 최초에 is->pictq.rindex_shown 0 → 1, 이후 ++rindex
is->force_refresh = 1;

video_refresh()

fftools/ffplay.c

video_refresh()

display:
if (!display_disable && is->force_refresh && is->show_mode == SHOW_MODE_VIDEO && is->pictq.rindex_shown)
    video_display(is);

is->force_refresh = 0;

if (show_status) 이면, 디버깅 로그 출력

"%7.2f %s:%7.3f fd=%4d aq=%5dKB vq=%5dKB sq=%5dB \r"
// e.g, 7.39 A-V: -0.028 fd= 51 aq= 30KB vq= 1KB sq= 0B

순서대로
마스터 클럭: get_master_clock(is),
A-V 오차,
프레임 드랍 개수: is->frame_drops_early + is->frame_drops_late
오디오 큐 크기: aqsize / 1024
비디오 큐 크기: vqsize / 1024,
자막 큐 크기: sqsize

-- Finished --

ffplay.c

// 첫 호출시, 윈도우 표시
if (!is->width)
    video_open(is); // SDL_ShowWindow(window) 호출

if (is->audio_st && is->show_mode != SHOW_MODE_VIDEO)
    video_audio_display(is); // 오디오 파형(SHOW_MODE_WAVES) 이나 SHOW_MODE_RDFT 출력
else if (is->video_st)
    video_image_display(is);

void video_image_display(VideoState *is)
>
vp = frame_queue_peek_last(&is->pictq); // rindex 프레임을 꺼내와 그려줌!!!
자막 있으면 표시
calculate_display_rect(&rect, is->xleft, is->ytop, is->width, is->height, vp->width, vp->height, vp->sar);

if (!vp->uploaded)
    upload_texture(&is->vid_texture, vp->frame)
    vp->uploaded = 1;
    vp->flip_v = vp->frame->linesize[0] < 0;

SDL_RenderCopyEx(renderer, is->vid_texture, NULL, &rect, 0, NULL, vp->flip_v ? SDL_FLIP_VERTICAL : 0); // 비디오 프레임 표시!

자막 있으면, SDL_RenderCopy(renderer, is->sub_texture, NULL, &rect);

sdl_audio_callback() -- called from SDL thread

fftools/ffplay.c

sdl_audio_callback()

```
audio_callback_time = av_gettime_relative(); // 콜백 시간 저장
while (len > 0) {
  if (is->audio_buf_index >= is->audio_buf_size) { // 첨엔 둘다 0
    audio_size = audio_decode_frame(is);

if (audio_size < 0) 이면, is->audio_buf = NULL 로 해서 무음 출력
아니면, update_sample_display(...) 로 is->sample_array 설정

// 오디오(디바이스) 출력
len1 = is->audio_buf_size - is->audio_buf_index;
memcpy(stream, (uint8_t *)is->audio_buf + is->audio_buf_index, len1);

len -= len1; // 보통 len = 8192, len1 = 4096 라서 2번 루프 돔
is->audio_buf_index += len1; // 쓴만큼 늘어남

// audio_callback_time 시점에 오디오 클럭(audclk)은 프레임 끝 시간 -
아직 출력안된 남은 버퍼 시간(현재 항상 0.0853) 위치를 재생중이어야
함?!
set_clock_at(&is->audclk,
   is->audio_clock - (double)(2 * is->audio_hw_buf_size/*8192*/ +
is->audio_write_buf_size/*0*/) / is->audio_tgt.bytes_per_sec/*192000*/,
is->audio_clock_serial,
audio_callback_time / 1000000.0);

sync_clock_to_slave(&is->extclk, &is->audclk);
```

ffplay.c

```
while (frame_queue_nb_remaining(&is->sampq) == 0) { // 오디오 프레임큐가 empty 하면
av_usleep (1000) 로 대기하는데 (1ms), 총 대기 시간이
오디오 콜백 한번 처리할 시간(is->audio_hw_buf_size(8192) / is->audio_tgt.bytes_per_sec(192000)
= 42.6ms)의 절반을 초과하게 되면 -1 에러 리턴(caller 에서 무음을 출력함)

af = frame_queue_peek_readable(&is->sampq) 로 오디오 프레임을 하나 얻어온 후,
frame_queue_next(&is->sampq);

wanted_nb_samples = synchronize_audio(is, af->frame->nb_samples);
> 오디오 마스터가 아닐 경우, 마스터와의 누적된(20번 누적) 클럭 차이(is->audio_diff_cum)가
is->audio_diff_threshold/*0.04266 = 42ms*/ 보다 크면, nb_samples 를 조절해 리턴

af->frame->format 등이 바뀐 경우,
software resampler 새로 생성(swr_init(is->swr_ctx))

if (is->swr_ctx) 이면, // 여긴 안 타고 있음
swr_set_compensation() 로 서서히 싱크를 맞춤
uint8_t **out = &is->audio_buf1;
len2 = swr_convert(is->swr_ctx, out, out_count, in, af->frame->nb_samples); // is->audio_buf1(out)
에 변환됨(리샘플링)
resampled_data_size = len2 * 4 를 저장
아니면
is->audio_buf = af->frame->data[0]; // af->frame->data 배열의 크기 = 8
resampled_data_size = data_size;

// af->pts(재생 시작 시간)에 샘플 재생시간을 더해 끝 시간을 저장
is->audio_clock = af->pts + (double) af->frame->nb_samples / af->frame->sample_rate;
is->audio_clock_serial = af->serial;
return resampled_data_size;
```

EOD

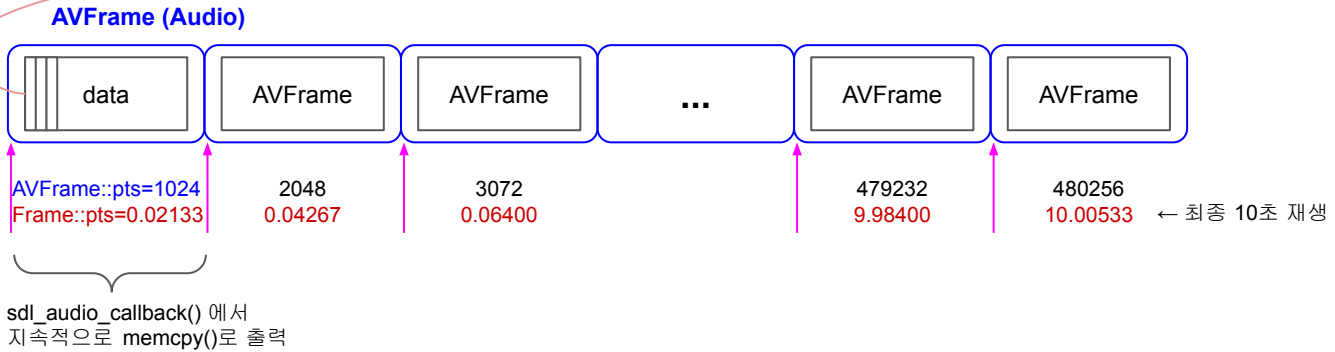
AVFrame

is->audio_buf 는 디코딩된 오디오 버퍼로 설정(audio_decode_frame() 에서)
is->audio_buf = af->frame->data[0]

오디오 1개 프레임 재생시간은

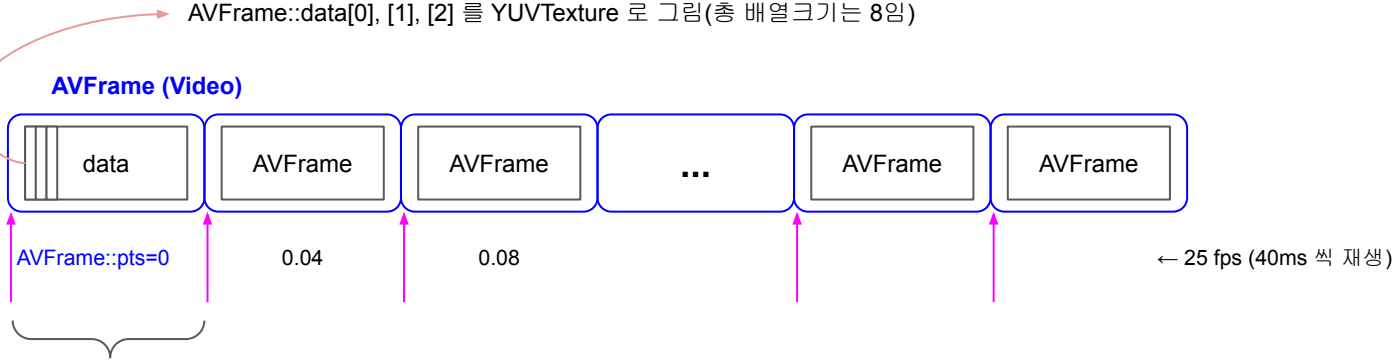
tb = {1/48000} 는
av_buffersink_get_time_base()
리턴값임

af->pts(Frame::pts) =
frame->pts * av_q2d(tb)
= 1024 * {1/48000}
= 0.02133, ...
즉, 1개 프레임 재생시간은 약
21.3ms

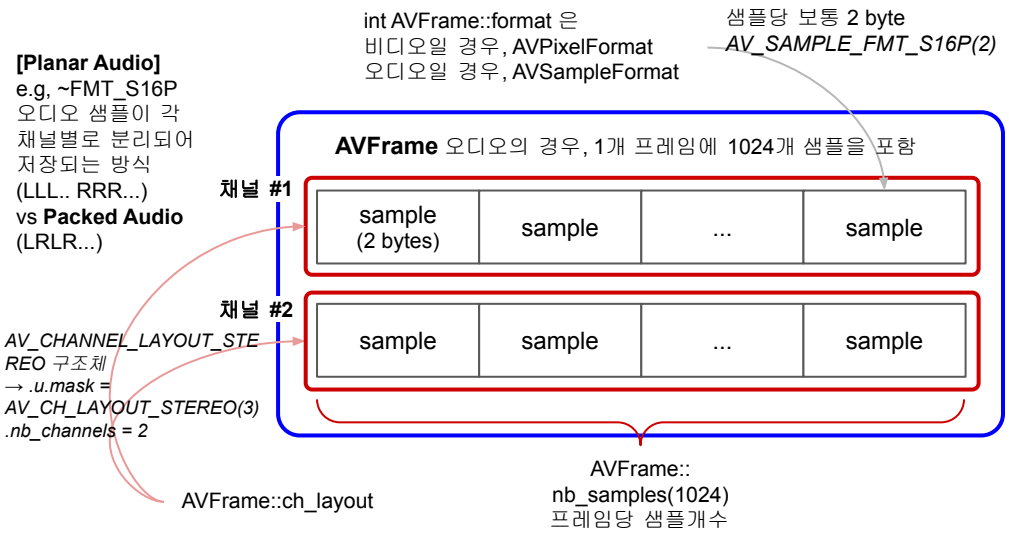


AVFrame::data[0], [1], [2] 를 YUVTexture 로 그림(총 배열크기는 8임)

frame->pts =
frame->best_effort_timestamp



*1개 오디오 프레임 크기(4096 Byte) = nb_samples(1024) * channels(2) * bytes_per_sample(2)*



audio_open(is, &ch_layout/*2ch*/, sample_rate/*48000*/, &is->audio_tgt/*out*/) 를 호출해
원하는 사양(ch, sample rate)의 is->audio_tgt 을 획득
이 함수에서 강제로 is->audio_tgt->fmt = AV_SAMPLE_FMT_S16 (1) 로 설정중임

audio_thread() 에서 디코딩 중이며, sdl_audio_callback() 에서 오디오가 출력됨

stream_component_open() 에서
ret = audio_open() → SDL_OpenAudioDevice(NULL, 0, &wanted_spec, &spec, ...)
ret 는 wanted_spec 로부터 얻은 spec.size/*8192*/
→ is->audio_hw_buf_size = ret/*8192*/; // SDL 하드웨어 오디오 버퍼 크기를 설정

audio_open(&is->audio_tgt) 시에만 ~.bytes_per_sec(192000) 등이 딱한번 설정
audio_hw_params->bytes_per_sec = av_samples_get_buffer_size(NULL,
audio_hw_params->ch_layout.nb_channels(2),
audio_hw_params->freq(48000), audio_hw_params->fmt(2), 1); // 192000
→ is->audio_tgt.bytes_per_sec 와 동일체 // 초당 오디오 데이터 크기

이후 is->audio_src.freq 와 af->frame->sample_rate 등이 달라질 경우,
swr_alloc_set_opts2() 로 리샘플링되지만 현재 호출되는 경우 X

```
// 오디오 콜백 sdl_audio_callback() 호출시
audio_callback_time = av_gettime_relative();  // 현재 시간 저장

1) audio_decode_frame() 에서 is->sampq 큐가 빈상태이면
av_usleep (1000/*1ms*/) 로 루핑하며 대기하다가 흘러간 시간이 오디오 콜백 한번 처리할 시간
(is->audio_hw_buf_size(8192) / is->audio_tgt.bytes_per_sec(192000) = 42.6ms)
의 절반을 초과하게 되면 -1 에러 리턴
→ 이 경우, 호출했던 sdl_audio_callback() 에서 무음 출력
// is->audio_clock 를 오디오 끝나는 시간(다음 출력 시간)으로 설정
is->audio_clock = af->pts + (double) af->frame->nb_samples / af->frame->sample_rate;

2) 위의 원래 오디오 PTS(is->audio_clock)에서, 아직 재생되지 않은 오디오 데이터의 시간을 빼서
is->audclk 보정(오디오 버퍼의 지연(buffering delay)을 고려하여(빼서) 오디오 클럭(is->audclk)을 조정)
set_clock_at(&is->audclk,
    is->audio_clock - (double)(2 * is->audio_hw_buf_size + is->audio_write_buf_size) /
    is->audio_tgt.bytes_per_sec, is->audio_clock_serial, audio_callback_time / 1000000.0);
→
is->audio_hw_buf_size: SDL 하드웨어 버퍼 크기(8192 byte).
is->audio_write_buf_size: 현재 남아 있는 출력되지 않은 오디오 데이터 크기.
is->audio_tgt.bytes_per_sec: 초당 재생되는 오디오 데이터 크기(192000).
→ 이를 나누면 버퍼에서 오디오가 실제로 재생되는 데 걸리는 시간(audio driver 에서 2 periods 가정).

// 출력되지 않은 버퍼 데이터를 계산해 오디오 클럭을 현재 실제 재생 중인 위치로 조정
// 오디오 데이터가 많이 버퍼링될수록(is->audio_write_buf_size 가 클수록)
// 오디오 클럭(pts 인자)을 앞으로 당겨(과거로 이동) 비디오와 동기화되도록 조정됨
```