千 Line으로 비디오플레이어 만들기 6

멀티미디어 : 2009/03/03 19:21

ffmpeg 천줄로 비디오플레이어 만들기
http://www.dranger.com/ffmpeg/ffmpeg.html
http://hybridego.net/entry/천줄로-비디오플레이어-만들기
http://hybridego.net/entry/千-Line-으로-비디오플레이어-만들기-1
http://hybridego.net/entry/千-Line-으로-비디오플레이어-만들기-2
http://hybridego.net/entry/千-Line-으로-비디오플레이어-만들기-3
http://hybridego.net/entry/千-Line-으로-비디오플레이어-만들기-4
http://hybridego.net/entry/千-Line-으로-비디오플레이어-만들기-5
http://hybridego.net/entry/千-Line으로-비디오플레이어-만들기-6
http://hybridego.net/entry/千-Line으로-비디오플레이어-만들기-7
http://hybridego.net/entry/千-Line으로-비디오플레이어-만들기-8
http://hybridego.net/entry/千-Line으로-비디오플레이어-만들기-마무리
별로 중요하지 않은 부분중 생략된 부분이 있고 의역이 많이 있습니다.
군데군데 틀린 번역이 있을수 있습니다. (영어가 후달려서)

하지만 소스코드 설명 부분은 틀리지 않도록 노력했습니다.

위의 원문을 번역한 것입니다.
잘못된 부분이 있을지도 모르겠습니다.
혹시 잘못된 부분을 발견하시면 댓글로 알려주시면 감사하겠습니다.

Tutorial 06: Synching Audio

tutorial06.c

Synching Audio

자~ 이제 우리는 동영상을 볼수 있는 어였한 플레이어를 갖게 되었습니다.
그럼 이제 늘어져 있는 나머지 작업들을 보겠습니다. 지난번에는 약간의 싱크 차이를 그냥 얼버무렸었습니다. 즉 비디오를 오디오에 동기화 시키는 것보다 오디오를 비디오 클럭에 동시화 시켰습니다. 우리는 이것을 비디오와 같은 방법으로 하고있습니다. : 비디오와 오디오가 얼마나 차이가 나는지를 추적하는 내부 비디오 클럭을 만드는 방법. 다음에는 오디오와 비디오를 외부 클럭에 어떻게 싱크하는지 찾아보겠습니다.

Implementing the video clock

이제 우리는 비디오 클럭을 저번에 했던 오디오 클럭처럼 실행해야 합니다. : 하나의 지역변수에 지금 비디오가 플레이 되고 있는 것의 time offset을 줍니다. 우선 여러분은 간단하게 현재 PTS와 타이머를 업데이트 합니다. 하지만 밀리세컨드 레벨에서 비디오 프레임의 간격은 꾀 길수도 있다는 것을 잊으면 안됩니다. 해결 방법은 다른 값을 추적하는 것입니다. 그 비디오 클럭을 마지막 프레임의 PTS 로 설정합니다. 비디오 클럭의 현재 값은 PTS_of_last_frame + (current_time - time_elapsed_since_PTS_value_was_set) 입니다. 이 방법은 get_audio_clock 에서 우리가 했던것과 매우 유사합니다.

그래서 우리의 큰 구조체 안에 double video_current_pts 와 int64_t video_current_pts_time 을 넣습니다. 클럭 업데이팅은 video_refresh_timer 함수에 자리잡고 있다.

```
view plain  copy to clipboard  print  ?
01. void video_refresh_timer(void *userdata) {
02.
03.
04.
05.   if(is->video_st) {
06.     if(is->pictq_size == 0) {
07.       schedule_refresh(is, 1);
08.     } else {
09.       vp = &is->pictq[is->pictq_rindex];
10.
11.       is->video_current_pts = vp->pts;
12.       is->video_current_pts_time = av_gettime();
```

av_gettime()
stream_component_open 에서 초기화 하는것을 잊지 마세요.

```
view plain  copy to clipboard  print  ?
01. is->video_current_pts_time = av_gettime();
```

av_gettime()
이제 우리가 필요한 것은 정보를 얻는 방법입니다.

```
view plain  copy to clipboard  print  ?
01. double get_video_clock(VideoState *is) {
02.   double delta;
03.
04.   delta = (av_gettime() - is->video_current_pts_time) / 1000000.0;
05.   return is->video_current_pts + delta;
06. }
```

Abstracting the clock

왜 우리가 video clock을 사용하도록 강제했을까요? 우리는 비디오 싱크 코드를 변경할 것입니다. 그래서 오디오와 비디오가 서로 싱크하려 하지 않습니다. 만약 우리가 ffplay에서 Command line option을 만들라고 할때 문제점을 상상해 보세요. 개념은 이렇습니다. 우리는 새로운 wrapper function을 만들것입니다. get_master_clock 함수는 av_sync_type 변수를 체크합니다. 그리고 get_audio_clock, get_video_clock 또는 우리가 쓰고싶은 다른 clock들을 호출합니다. 우리는 컴퓨터의 clock도 사용할수 있습니다. 우리는 get_external_clock을 호출할것입니다.

```
view plain  copy to clipboard  print  ?
01. enum {
02.   AV_SYNC_AUDIO_MASTER,
03.   AV_SYNC_VIDEO_MASTER,
```

```
  AV_SYNC_EXTERNAL_MASTER,
};

#define DEFAULT_AV_SYNC_TYPE AV_SYNC_VIDEO_MASTER

double get_master_clock(VideoState *is) {
  if(is->av_sync_type == AV_SYNC_VIDEO_MASTER) {
    return get_video_clock(is);
  } else if(is->av_sync_type == AV_SYNC_AUDIO_MASTER) {
    return get_audio_clock(is);
  } else {
    return get_external_clock(is);
  }
}
main() {
...
  is->av_sync_type = DEFAULT_AV_SYNC_TYPE;
...
}
```

Synchronizing the Audio

여기는 어려운 부분입니다. 오디오를 비디오 클럭에 동기화 시킬 것입니다. 우리의 전략은 오디오가 어디인지를 측정하고 그것을 비디오 클럭과 비교한다음 얼마나 많은 샘플을 우리가 조정해야 하는지 알아내는 것입니다. 속도를 올려야 되는지, 샘플들을 건너뛰어야 하는지 아니면 속도를 줄여야 하는지 말이지요

우리는 synchronize_audio함수를 실행할 것입니다. 매번 audio sample 세트를 처리하고 그것을 알맞게 늘리거나 줄여야 합니다. 그러나 우리는 매번 동기화 하는것을 원하지 않습니다. 그것은 비디오 패킷보다 훨씬 더 자주 오디오를 처리하기 때문입니다. So we're going to set a minimum number of consecutive calls to the synchronize_audio function that have to be out of sync before we bother doing anything. 물론 지난번과 같이 "out of sync"의 의미는 오디오클럭과 비디오크럭같에 우리의 sync threshold를 벗어났다는 것을 말합니다.

그래서 우리는 fractional coefficient를 사용합니다. 그리고 out of sync 된 N개의 오디오 샘플 세트를 얻습니다. out of sync가 많으면 많은 변화가 일어날 수 있어서 우리는 각각의 out of sync된 것들의 평균값을 사용합니다. 예를 들면 첫번째호출은 아마 40ms 싱크가 어긋났고 다음번에는 50ms, 쭉~~ 이렇게 보일것입니다. 하지만 우리는 간단한 평균값을 갖지 않을것 입니다. 왜냐면 대부분 최근 값들은 이전 값들보다 더 중요하기 때문입니다. 그래서 우리는 fractional codfficient, say c, 그리고 다음 처럼 차이를 더한 것을 사용합니다. : diff_sum = new_diff + diff_sum*c 우리가 차이값을 찾을 준비가 되면 avg_diff=diff_sum*(1-c) 로 간단히 계산합니다.
Here's what our function looks like so far:

```
view plain  copy to clipboard  print  ?
```

```


int synchronize_audio(VideoState *is, short *samples,
                int samples_size, double pts) {
  int n;
  double ref_clock;

  n = 2 * is->audio_st->codec->channels;

  if(is->av_sync_type != AV_SYNC_AUDIO_MASTER) {
    double diff, avg_diff;
    int wanted_size, min_size, max_size, nb_samples;

    ref_clock = get_master_clock(is);
    diff = get_audio_clock(is) - ref_clock;

    if(diff < AV_NOSYNC_THRESHOLD) {

      is->audio_diff_cum = diff + is->audio_diff_avg_coef
```

```
      * is->audio_diff_cum;
        if(is->audio_diff_avg_count < AUDIO_DIFF_AVG_NB) {
      is->audio_diff_avg_count++;
        } else {
      avg_diff = is->audio_diff_cum * (1.0 - is->audio_diff_avg_coef);


        }
      } else {

        is->audio_diff_avg_count = 0;
        is->audio_diff_cum = 0;
      }
    }
    return samples_size;
}
```

우리는 꽤 잘하고 있습니다. 우리가 사용하고 있는 클럭이나 비디오로부터 오디오가 얼마나 차이나는지 대략 알고있습니다. 그래서 이제 어느정도의 샘플을 "Shrinking/expanding buffer code"코드에서 추가하거나 빼거나 할지 계산합니다.

view plain copy to clipboard print ?

```
if(fabs(avg_diff) >= is->audio_diff_threshold) {
  wanted_size = samples_size +
  ((int)(diff * is->audio_st->codec->sample_rate) * n);
  min_size = samples_size * ((100 - SAMPLE_CORRECTION_PERCENT_MAX)
                             / 100);
  max_size = samples_size * ((100 + SAMPLE_CORRECTION_PERCENT_MAX)
                             / 100);
  if(wanted_size < min_size) {
    wanted_size = min_size;
  } else if (wanted_size > max_size) {
    wanted_size = max_size;
  }
```

audio_length * (sample_rate * # of channels * 2) 는 오디오의 audio_length초 에 있는 샘플 수 라는것을 기억하세요. 그래서 샘플 수를 더하거나 빼고 오디오가 흘러간 양만큼 조정해야 합니다. 또한 크건 작건 보정한것의 한계를 설정해야 합니다. 왜냐면 버퍼가 너무 많이 바뀌면 사용자에게 혼란을 줄수 있기 때문입니다.

Correcting the number of samples

이제 우리는 오디오를 교정해야 합니다. 여러분은 아마 synchronize_audio 함수를 아실텐데 이 함수는 스트림으로 몇 바이트를 보내는지를 나타내는 sample size를 리턴합니다. 그래서 우리는 그냥 sample size를 wanted_size로 조정하기만 하면 됩니다. 이일은 sample size를 작게 만들기 위함입니다. 하지만 우리가 이것을 크게 만들고 싶다면 그냥 크게 만들수는 없습니다. 왜냐면 버퍼에 더이상의 데이터가 없기 때문입니다. 그래서 우리는 이것을 추가해야 합니다. 그런데 뭘 추가하죠? 이것은 바보같은 시도가 될것이고 오디오를 추측하는 것입니다.그래서 우리는 마지막 샘플의 값으로 이미 부풀려진 버퍼의 오디오를 그냥 사용합니다.

view plain copy to clipboard print ?

```
if(wanted_size < samples_size) {

  samples_size = wanted_size;
} else if(wanted_size > samples_size) {
  uint8_t *samples_end, *q;
  int nb;


  nb = (samples_size - wanted_size);
  samples_end = (uint8_t *)samples + samples_size - n;
  q = samples_end + n;
```

```
  while(nb > 0) {
    memcpy(q, samples_end, n);
    q += n;
    nb -= n;
  }
  samples_size = wanted_size;
}
```

이제 우리는 샘플 싸이즈를 리턴하고 함수를 끝마쳤습니다.
이제 우리는 이것들을 다 사용해야 합니다.

view plain copy to clipboard print ?

```
void audio_callback(void *userdata, Uint8 *stream, int len) {

  VideoState *is = (VideoState *)userdata;
  int len1, audio_size;
  double pts;

  while(len > 0) {
    if(is->audio_buf_index >= is->audio_buf_size) {

      audio_size = audio_decode_frame(is, is->audio_buf, sizeof(is->audio_buf), &pts);
      if(audio_size < 0) {

    is->audio_buf_size = 1024;
    memset(is->audio_buf, 0, is->audio_buf_size);
      } else {
    audio_size = synchronize_audio(is, (int16_t *)is->audio_buf,
                       audio_size, pts);
    is->audio_buf_size = audio_size;
```

우리가 했던 것은 synchronize_audio를 호출함으로서 입력됩니다. (또한 어디어 우리가 변수들을 초기화 하였는지 확실히 체크해야 합니다.)

끝마치기 전에 하나가 남았는데
we need to add an if clause to make sure we don't sync the video if it is the master clock:

view plain copy to clipboard print ?

```
if(is->av_sync_type != AV_SYNC_VIDEO_MASTER) {
  ref_clock = get_master_clock(is);
  diff = vp->pts - ref_clock;



  sync_threshold = (delay > AV_SYNC_THRESHOLD) ? delay :
                   AV_SYNC_THRESHOLD;
  if(fabs(diff) < AV_NOSYNC_THRESHOLD) {
    if(diff <= -sync_threshold) {
      delay = 0;
    } else if(diff >= sync_threshold) {
      delay = 2 * delay;
    }
  }
}
```

다 되었군요!
여러분은 어떤 변수를 초기화 할때 중복정의 되었는지 중복초기화 되었는지 전체 소스를 잘 체크하셔야 합니다.

```
gcc -o tutorial06 tutorial06.c -lavutil -lavformat -lavcodec -lz -lm`sdl-config --cflags --libs`
```

다음에는 뒤로감기와 빨리감기를 만들어 보겠습니다.

千 Line으로 비디오플레이어 만들기 6 조회수 418

9 아이유 열애설 상대 누구? '주변사람이 만류할 정...
12 착한남자, 가장 잔인했던 1초 문채원의 키스가 두...

'멀티미디어' 카테고리의 다른 글

千 Line으로 비디오플레이어 만들기 7 (0) 2009/03/09
千 Line으로 비디오플레이어 만들기 6 (0) 2009/03/03
千 Line 으로 비디오플레이어 만들기 5 (0) 2009/02/27
千 Line 으로 비디오플레이어 만들기 4 (0) 2009/02/17

Posted by Real_G