Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

КУРСОВАЯ РАБОТА

Использование библиотеки FFmpeg-Python
по дисциплине «Архитектура ЭВМ»

Выполнил
студент гр. в3530904/00030 В.С. Баганов

Руководитель
профессор, к.т.н. С.А. Молодяковв

«____»_____ 202__ г.

Санкт-Петербург
2023

Содержание

1. Библиотека FFmpeg

FFmpeg - набор свободных библиотек с открытым исходным кодом, которые позволяют записывать, конвертировать и передавать цифровые аудио- и видеозаписи в различных форматах. Он включает libavcodec - библиотеку кодирования и декодирования аудио и видео, и libavformat - библиотеку мультиплексирования и демультиплексирования в медиаконтейнер. Название происходит от названия экспертной группы MPEG и FF, означающего «fast forward»

2. Функции FFmpeg

ffmpeg состоит из следующих компонентов:

ffmpeg — утилита командной строки для конвертирования видеофайла из одного формата в другой. С её помощью можно также захватывать видео в реальном времени с TV-карты.

ffplay — простой медиаплеер, основанный на SDL и библиотеках FFmpeg.

fprobe — консольная утилита, позволяющая собирать и отображать информацию о медиафайлах (как MediaInfo) мультимедиапотоках, доступных устройствах, кодеках, форматах, протоколах и др.

ffserver — HTTP- (RTSP в настоящее время разрабатывается) потоковый сервер для видео- или радиовещания.

libavcodec — библиотека со всеми аудио/видеокодеками. Большинство кодеков было разработано «с нуля» для обеспечения наилучшей производительности.

libavformat — библиотека с мультиплексорами и демультиплексорами для различных аудио- и видеоформатов.

libavutil — вспомогательная библиотека со стандартными общими подпрограммами для различных компонентов ffmpeg. Включает Adler-32, CRC, MD5, SHA1, LZO-декомпрессор, Base64-кодер/декодер, DES-шифровальщик/расшифровщик, RC4-шифровальщик/расшифровщик и AES-шифровальщик/расшифровщик.

libpostproc — библиотека стандартных подпрограмм обработки видео.

libswscale — библиотека для масштабирования видео.

libavfilter — замена vhook, которая позволяет изменять видеопоток между декодером и кодером «на лету».

3. Функции FFmpeg-Python

ffmpeg.compile(stream_spec , cmd = 'ffmpeg' , overwrite_output = False)

Создает командную строку для вызова ffmpeg.

ffmpeg.probe(имя файла , cmd = 'ffprobe' , ** kwargs)

Запускает ffprobe для указанного файла и возвращает JSON-представление вывода.

class ffmpeg.Stream(upstream_node , upstream_label , node_types , upstream_selector = None)

Представляет исходящую границу восходящего узла; может использоваться для создания большего количества нисходящих узлов.

ffmpeg.input(имя файла , ** kwargs)

URL входного файла (-i опция ffmpeg)

ffmpeg.merge_outputs(* потоки)

Включить все данные выходы в одну командную строку ffmpeg

ffmpeg.output(* streams_and_filename , ** kwargs)

URL выходного файла

ffmpeg.overwrite_output(поток)

Перезаписывать выходные файлы без запроса (-уопция ffmpeg)

ffmpeg.probe(имя файла , cmd = 'ffprobe' , ** kwargs)

Запустить ffprobe для указанного файла и верните JSON-представление вывода.

ffmpeg.compile(stream_spec , cmd = 'ffmpeg' , overwrite_output = False)

Создать командную строку для вызова ffmpeg.

ffmpeg.get_args(stream_spec , overwrite_output = False)

Создать аргументы командной строки для передачи в ffmpeg.

ffmpeg.run(stream_spec , cmd = 'ffmpeg' , capture_stdout = False , capture_stderr = False , input = None , quiet = False , overwrite_output = False)

Вызвать ffmpeg для предоставленного графа узлов.

ffmpeg.run_async(stream_spec , cmd = 'ffmpeg' , pipe_stdin = False , pipe_stdout = False , pipe_stderr = False , quiet = False , overwrite_output = False)

Асинхронно вызывать ffmpeg для предоставленного графа узлов.

ffmpeg.view(stream_spec , detail = False , filename = None , pipe = False , ** kwargs)

ffmpeg.colorchannelmixer(поток , * аргументы , ** kwargs)

Отрегулировать кадры видеовхода путем повторного смешивания цветовых каналов.

ffmpeg.concat(* Streams , ** kwargs)

Объединить аудио- и видеопотоки, объединяя их друг за другом.

ffmpeg.crop(поток , x , y , ширина , высота , ** kwargs)

Обрезать входное видео.

ffmpeg.drawbox(поток , x , y , ширина , высота , цвет , толщина = None , ** kwargs)

Нарисовать цветную рамку на входном изображении.

ffmpeg.drawtext(поток , текст = Нет , x = 0 , y = 0 , escape_text = True , ** kwargs)

Нарисовать текстовую строку или текст из указанного файла поверх видео, используя библиотеку libfreetype.

ffmpeg.filter(Stream_spec , FILTER_NAME , * Args , ** kwargs)

Применить собственный фильтр.

ffmpeg.filter_multi_output(Stream_spec , FILTER_NAME , * Args , ** kwargs)

Применить настраиваемый фильтр с одним или несколькими выходами.

ffmpeg.hflip(поток)

Перевернуть входное видео по горизонтали.

ffmpeg.hue(поток , ** kwargs)

Изменить оттенок и / или насыщенность ввода.

ffmpeg.overlay(main_parent_node , overlay_parent_node , eof_action = 'repeat' , ** kwargs)

Наложить одно видео поверх другого.

ffmpeg.setpts(поток , выражение)

Изменить PTS (временную метку представления) входных кадров.

ffmpeg.trim(поток , ** kwargs)

Обрезать ввод так, чтобы вывод содержал одну непрерывную часть ввода.

ffmpeg.vflip(поток)

Перевернуть входное видео по вертикали.

ffmpeg.zoompan(поток , ** kwargs)

Применить эффект масштабирования и панорамирования.

Некоторые фильтры ffmpeg отбрасывают аудиопотоки, и необходимо соблюдать осторожность, чтобы сохранить звук в окончательном выводе. Операторы .audio и .video могут использоваться для ссылки на части аудио / видео потока, чтобы их можно было обрабатывать отдельно, а затем повторно комбинировать позже в конвейере. Эта дилемма присуща ffmpeg, и ffmpeg-python старается не вмешиваться, в то время как пользователи могут обратиться к официальной документации ffmpeg, чтобы узнать, почему определенные фильтры пропускают звук.

Независимая обработка аудио и видео частей потока:

Подключаем библиотеку FFmpeg. С помощью функции .input загружаем ролики.

```
import ffmpeg

inVideo0 = ffmpeg.input("reg2-1.mp4")
inVideo1 = ffmpeg.input("reg2-2.mp4")
inVideo2 = ffmpeg.input("reg2-3.mp4")
```

С помощью функций video.filter и audio.filter разделяем на потоки каждое видео. Параметр trim для видео позволяет управлять началом потока, через start = 0 - назначить время запуска, duration = 5.7 управлять продолжительностью. Для контроля аудио потока используется atrim с аналогичными параметрами start = 0 и duration = 5.7. Благодаря разделению каждого видео на отдельные потоки мы можем контролировать и использовать нужные видео и аудиопоки. В результате из 3 видео мы получили 3 аудиопотока и 3 видеопотока.

```
v0_1 = inVideo0.video.filter('trim', start=0, duration=5.7)
a0_1 = inVideo0.audio.filter('atrim', start=0, duration=5.7)
v0_2 = inVideo0.video.filter('trim', start=0.7, duration=8.8)
a0_2 = inVideo0.audio.filter('atrim', start=0.7, duration=8.8)
v0_3 = inVideo0.video.filter('trim', start=0.0,
↪  duration=8.5).setpts('PTS-STARTPTS')
a0_3 = inVideo0.audio.filter('atrim', start=0.0,
↪  duration=8.5).filter_('asetpts', 'PTS-STARTPTS')
```

Параметр ‘scale' позволяет масштабировать видео под заданные размеры. Все исходные видео имеют одинаковый размер 640x360 пикселей. В данном случае video1 мы масштабируем под координаты равные половине высоты видео 640x180 т. е. получатся видео поток половины высоты основного видео. Далее video2 масштабируем на половину 640x180, чтобы получить квадратное видео 320x150.

```
# width = 640
# height = 180
video1 = inVideo1.video.filter('scale', 640, 180)
audio1 = inVideo1.audio
# width = 320
# height = 150
video2 = inVideo2.video.filter('scale', 320, 150)
audio2 = inVideo2.audio
# video2 = inVideo2.video.filter('hflip')
```

Функция ffmpeg.overlay позволяет наложить потоки друг на друга. joined_video1 = ffmpeg.overlay(v0_2, video1, тут мы наложили на основное видео половину от второго видео (640x360). В результате получаем видео с 2 потоками которые вертикально разделяют изображение на 2 части. joined _video2 = ffmpeg.overlay(joined_video1, video2, тут мы к обновленному потоку накладываем 3 поток (320x150, так как мы не задаем позиционирование x и y, то видео будет в левом верхнем углу). Далее ffmpeg.concat позволяет объединить все потоки в один поток и через функцию ffmpeg.output выводим/записываем новое получившиеся видео, результат которого, можно посмотреть на скринах. Функция run () запускает процесс исполнения программы.

```
joined_video1 = ffmpeg.overlay(v0_2, video1, eof_action='endall')
joined_video2 = ffmpeg.overlay(joined_video1, video2,
↪  eof_action='endall')
# joined_video2 = ffmpeg.hflip(joined_video1, video2,
↪  eof_action='endall')
joined_audio = ffmpeg.filter([a0_2, audio1], "amix")
# pre_output = ffmpeg.concat(v0_1, a0_1, joined_video2, joined_audio,
↪  v=1, a=1).node
pre_output = ffmpeg.concat(v0_1, a0_1, joined_video2, joined_audio, v=1,
↪  a=1).node
v3 = pre_output [0]
a3 = pre_output [1]
ffmpeg.output(v3, a3, "videoNEW.mp4").overwrite_output().run()
```

4. Листинг программы FFmpeg-Python

```
import ffmpeg

inVideo0 = ffmpeg.input("reg2-1.mp4")
inVideo1 = ffmpeg.input("reg2-2.mp4")
inVideo2 = ffmpeg.input("reg2-3.mp4")
v0_1 = inVideo0.video.filter('trim', start=0, duration=5.7)
a0_1 = inVideo0.audio.filter('atrim', start=0, duration=5.7)
v0_2 = inVideo0.video.filter('trim', start=0.7, duration=8.8)
a0_2 = inVideo0.audio.filter('atrim', start=0.7, duration=8.8)
v0_3 = inVideo0.video.filter('trim', start=0.0,
↪  duration=8.5).setpts('PTS-STARTPTS')
a0_3 = inVideo0.audio.filter('atrim', start=0.0,
↪  duration=8.5).filter_('asetpts', 'PTS-STARTPTS')
# width = 640
# height = 180
video1 = inVideo1.video.filter('scale', 640, 180)
audio1 = inVideo1.audio
# width = 320
# height = 150
video2 = inVideo2.video.filter('scale', 320, 150)
# video2 = inVideo2.video.filter('hflip')

audio2 = inVideo2.audio
joined_video1 = ffmpeg.overlay(v0_2, video1, eof_action='endall')
joined_video2 = ffmpeg.overlay(joined_video1, video2,
↪  eof_action='endall')
# joined_video2 = ffmpeg.hflip(joined_video1, video2,
↪  eof_action='endall')
joined_audio = ffmpeg.filter([a0_2, audio1], "amix")
# pre_output = ffmpeg.concat(v0_1, a0_1, joined_video2, joined_audio,
↪  v=1, a=1).node
pre_output = ffmpeg.concat(v0_1, a0_1, joined_video2, joined_audio, v=1,
↪  a=1).node
v3 = pre_output [0]
a3 = pre_output [1]
ffmpeg.output(v3, a3, "videoNEW.mp4").overwrite_output().run()
```

5. Скрины полученного видео