Archtecture Assignment: MoviePy[1]

1 Overview

The MoviePy is a Python API library that helps to edit movies. The input could be images or videos and output could be videos of different formats. This API is built based on the following several dependencies:

- *numpy* - calculations and manipulations
- *imageio* - reading and writing images of different format
- *Decorator* - sync manipulation over videos on masks
- *tqdm* - showing video progress progress on command-line UI
- *youtube_dl* - interacting with Youtube
- *Sphinx* - creating *MoviePy* documentation
- *requests* - opening URL
- *ffmpeg* - processing video.

The Main components and their building blocks including:

- VideoClip - Image, Color, and Text
- AudioClip - Audio File and Composite Audio
- videofx - Video effects
- audiofx - Fade effect, looping, and volume
- videotools - Credits, drawing, segmenting, subtitles, and tracking
- audiotools - Not implemented
- ffmpeg - ffmpeg Tools for audio extraction, sub-clip extraction, merger video and audio, form video by frame, and resizing a video
- decorators - apply functions to mask and video

[1]This review is done under docker MoviePy environment.

2 Error Handling

The Moviepy usually handle errors with if statment by rasing exceptions. This is the best way to handle errors. Using try and catch is the first stage of handling errors. After some time, a mature way of handling errors should be deterministic as most of the Moviepy error handling technique. The expcetion is usually raised with specific error message suggesting users a readable error message. An example is attached in the flowing:

```
        if change_end:
            self.end = None if (t is None) else (self.start + t)
        else:
            if self.duration is None:
                raise Exception("Cannot change clip start when new"
                                "duration is None")
            self.start = self.end - t
```

Other error handling technique such as return success are used to make error handling a deterministic process with readable error messages. The current implementation, however, is lack of version control over its dependencies. The major dependency error comes from ffmpeg and Imageio where the versions mismatch error is not handled. Since it is a open source project, the open source community has developed several version and deleted some of the beginner friendly part. The Moviepy no longer supports dependency auto-installation and with it the dependency version control problem shows. A leading error from this problem is the video reading error which could come from unsupported or deprecated ffmpeg version.

There are still a lot of room for improvement on the error handling matter. Some error are not generated because the process is considered successful but the output is not. For example, videos generated becomes unreadable and broken, checked all available decoding method, while there is no error message from the API.

3 Architecture

The movie editing test came out with interaction with API components as shown in Figure 1.

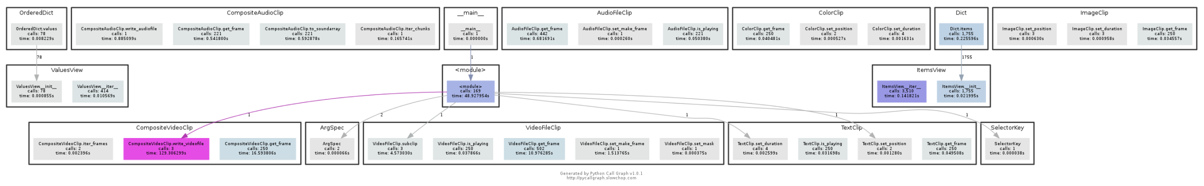
OrderedDict
OrderedDict.values calls: 78 time: 0.008229s
CompositeAudioClip
CompositeAudioClip.write_audiofile calls: 1 time: 0.885099s
CompositeAudioClip.get_frame calls: 221 time: 0.541800s
CompositeAudioClip.to_soundarray calls: 221 time: 0.592878s
CompositeAudioClip.iter_chunks calls: 1 time: 0.165741s
__main__
__main__ calls: 1 time: 0.000000s
AudioFileClip
AudioFileClip.get_frame calls: 442 time: 0.681691s
AudioFileClip.set_make_frame calls: 1 time: 0.000260s
AudioFileClip.is_playing calls: 221 time: 0.050380s
ColorClip
ColorClip.get_frame calls: 250 time: 0.040481s
ColorClip.set_position calls: 2 time: 0.000527s
ColorClip.set_duration calls: 4 time: 0.001631s
Dict
Dict.items calls: 1,755 time: 0.225596s
ImageClip
ImageClip.set_position calls: 3 time: 0.000630s
ImageClip.set_duration calls: 3 time: 0.000958s
ImageClip.get_frame calls: 250 time: 0.034557s
78
ValuesView
ValuesView.__init__ calls: 78 time: 0.000855s
ValuesView.__iter__ calls: 414 time: 0.010569s
1
<module>
<module> calls: 169 time: 48.927954s
1755
ItemsView
ItemsView.__iter__ calls: 3,510 time: 0.141821s
ItemsView.__init__ calls: 1,755 time: 0.021995s
1
2
1
1
1
1
CompositeVideoClip
CompositeVideoClip.iter_frames calls: 2 time: 0.002396s
CompositeVideoClip.write_videofile calls: 3 time: 129.306299s
CompositeVideoClip.get_frame calls: 250 time: 16.593806s
ArgSpec
ArgSpec calls: 2 time: 0.000066s
VideoFileClip
VideoFileClip.subclip calls: 3 time: 4.573030s
VideoFileClip.is_playing calls: 250 time: 0.037866s
VideoFileClip.get_frame calls: 502 time: 10.976285s
VideoFileClip.set_make_frame calls: 1 time: 1.513765s
VideoFileClip.set_mask calls: 1 time: 0.000375s
TextClip
TextClip.set_duration calls: 4 time: 0.002599s
TextClip.is_playing calls: 250 time: 0.031698s
TextClip.set_position calls: 2 time: 0.001280s
TextClip.get_frame calls: 250 time: 0.049508s
SelectorKey
SelectorKey calls: 1 time: 0.000038s
Generated by Python Call Graph v1.0.1
http://pycallgraph.slowchop.com

Figure 1: Architecture

This architecture flow graph outlines the function calls going through each involved component of the API. The test file resized a video input and output into a video in a different format. The main function source code is presented as the following.

```
#!/usr/bin/env python3
from moviepy.editor import *
```

```
video = VideoFileClip("myHolidays.mkv").subclip(50,60)

# Make the text. Many more options are available.
txt_clip = ( TextClip("My Holidays 2013",fontsize=70,color='white')
            .set_position('center')
            .set_duration(10) )

result = CompositeVideoClip([video, txt_clip]) # Overlay text on video
result.write_videofile("myHolidays_edited.webm",fps=25) # Many options...
```

The flow graph starts from the middle where the *__main__* module calls for components (Class): Composite Video Clip, Video Clip, and Text Clip. Within each class, several functions can be invoked. This specific main function invokes one to two functions in each class. However, inside the class, each function may be called by each other for several times and is shown on the flow graph. The ones not directly called but triggered by other functions in this specific main function are listed at the top.

4 Issue #640

The issue #640 is created regarding clip resizing function in file the "resize.py". This function plays an important role in video processing procedure. The issue for the function is considered bug and was proposed a fix for it. It is more of a pull request than an issue since a solution is proposed.

The problem code piece is attached as follow:

```
newsize2 = lambda t : trans_newsize(newsize(t))

       if clip.ismask:

           fun = lambda gf,t: (1.0*resizer((255 * gf(t)).astype('uint8'),
                                          newsize2(t))/255)

       else:

           fun = lambda gf,t: resizer(gf(t).astype('uint8'),
                                    newsize2(t))

       return clip.fl(fun, keep_duration=True,
                    apply_to= (["mask"] if apply_to_mask else []))
```

.
The issuer claims that the function fails for mask frames in the *else* argument. However, the *else* statement should not process any clip with a mask as the leading *if* statement is directing all the clip with mask into the first statement. The *else* statement should not handle any clips with a mask.

Nevertheless, the proposed solution is attached as below. The issuer copied handle from *ismask* statement to the else in an attempt to band-aid the error.

```
 newsize2 = lambda t : trans_newsize(newsize(t))
```

```
if clip.ismask:

    fun = lambda gf,t: (1.0*resizer((255 * gf(t)).astype('uint8'),
                                  newsize2(t))/255)

else:

    fun = lambda gf,t: resizer(gf(t).astype('uint8'),
                      newsize2(t)) if len(gf(t).shape) == 3 else
                          1.0*resizer((gf(t)*255).astype('uint8'),newsize2(t))/255

return clip.fl(fun, keep_duration=True,
              apply_to= (["mask"] if apply_to_mask else []))`
```