Videout: A programming language for media editing

Brian Rodríguez Badillo, Christian Pérez,
Alejandro Reyes, Lexdyel Méndez Ríos

Introduction

Videout is a programming language that allows the user to generate simple and customizable videos using programming concepts instead of film editing programs. This is due to the high complexity of this programs, and the computational demand of them. Of course, by simplifying this process, it is bound to eliminate many features; like video stabilization, rotoscoping, keying, etc. Yet at the same time, it does keep the most essential and used tools; such as trimming, audio importation, title generation, concatenation, etc.

language reference Manual

When it comes to **rendering**, the user can render both videos and gifs. Unlike many video editing software, the steps required to do the rendering are very simple.

```
>> clip = video from "C:\\Users\\user\\Videos\\LoveIsWar.mp4" between 0,1 and
0,30
>> renderVid clip
```

a. Crop:

The crop method works by adjusting the aspect ratio of a video. Keeping in mind that this method will cut part of the video if it’s to a smaller aspect ratio. The user can decide between any of this different aspect ratios.

```
>> crop clip by widescreen
```

b. Resize:

The resize method is a method that changes the size, but unlike crop, it keeps the entire image of the video. An example of how it works would be:

```
>> resize clip by 3
```

c. Audio:

The user can edit the audio of the video at any desired part of it. This works by taking a file with the desired audio and setting it with the boundaries of where its going to play in the video.The user can also decide what part of the audio should be used inside of a time frame start,end
It's important to add the path as a string with " " and that the dashes are double \\.

```
>> addAudio "C:\\User\\user\\Music\\dropbeat.mp3" to clip between 3,7
```

There is also a function that allows you to extract the audio from a clip, and add that audio to another desired clip. Making the re-use of audio much easier than using addAudio each time.

```
>> extractAudio clip2 to clip1 between 0,15.
```

d. Gif:

From any desired video, the user has the chance of turning it into a gif. This means that the video will constantly loop. The rendering of this type of media takes as input, a variable that already holds the video.

```
>> renderGif from clip
```

e. Photo:

Another function that can be done is making a variable hold an image. Later that image can be concatenated at any clip already made. This can be done for titles or just an ending image of the video. Making this variable is just as making the one for videos, but instead of a time frame, the user decides how much it's going to last that image being shown when its concatenated. The time is taken in consideration as seconds.

```
>> pic = photo from "C:\\User\\user\\Photos\image.jpg" lasting 5
```

f. Text:

Lastly, one of the features is adding text to a desired video. This process is just as simple as deciding what text you want to add to what video, and then the desired time that the user wants the text to appear at.

```
>> addText "I like trains..." to clip to bottom
```

g. Concatenation:

Lastly, there is also a function that allows you to concatenate 2 videos (or video and image), making a new clip of it. The order of the concatenation matters, since its going to be added from left to right. This means that if you use the video before a picture the result will be a video with that image at the end.

```
>> concat = concatenateClip clip and pic
```

If the user wants to add the image first, it must be specified that way

```
>> concat = concatenateClip pic and clip
```

Language Development

a.Translator Architecture

After defining our lexical tokens and keywords, in our parser we define different methods that will be called to translate whatever the user is writing. These methods comply with PLY parsing standards, and they are called on execution by the user.

Whenever the parser receives an input, it searches within these methods the configuration of parameters needed to fulfill such method. Then, after the parser finds which method belongs to the configuration of the executed program, it will then run a series of commands, issued under the very methods used to translate. These commands tie up the parser to the intermediate code and executes the desired function, as per the instruction and functionality given.

For the initialization, we have different concatenated methods which will check upon the keywords written out by the user and, if approved, will assign to whatever variable the user defined with either a string, number, picture clip or video clip. This is fundamentally used to declare variables and store them for later use and, as it is noted, it already connects to the intermediate code to create such video and picture clips. These variables will stay on the system and can be used to edit said clips and ultimately render them, which is done the same way: the parser methods validates the keywords written by the user then executes the command on the intermediate code to edit and render the clips successfully.

b. Modules

The intermediate code is designed to work by itself on command, which will call a series of methods defined by us which points to the python library used: moviepy. Two base classes are made which interact with each other to work out all the editing methods as well as the final rendering of the clips. These classes are then imported to our language parser which then will be used to execute whatever commands the user writes as per our architecture and design.

This way, the code is kept simple and easier to understand, while keeping its functionality at its highest.

c. Environment

Since the chosen language was python, it was open to what IDE we each would use. For example, some would use an actual IDE such as PyCharm, while others would use text editors like Atom or VS Code. At first, this freedom was working perfectly, but when we started to all noticed the advantages of PyCharm for version control, we decided too to use that one in common. This of course also helped on the organization of the code, specially the area of the lexer. In some cases, text editors such as Atom, need plugins like *beautify* to make the tabs all look organized. Issues like this one were not necessarily affecting the functionality of the code, but it was affecting the readability. Of course, once we all started to use PyCharm, this issue was automatically fixed without the need of any plugins.

d. Test methodology

Testing was done by first running aa series of commands internally on the intermediate code so that we could work out all the functions in a way that our language didn't need many definitions and parameters to decrease complexity and increase functionality efficiency. After this testing was complete and the intermediate code worked as expected, we proceeded to individually test each token and keyword inside the parser by adding pointers such as prints and returns to ensure the methods were being accessed correctly. After changing the lexical and parsing structure various times, we were able to define a structure which would consistently execute the desired methods by just entering the correct lime of keywords and tokens. After this was set, it was proceeded to integrate both modules together, which were then tested by running actual code such as:

```
clip = video from "C:\\path" between 0,5 and 0,15
addAudio "C:\\path" to clip between 15,30
renderVid clip

resize clip by 2
crop clip by phone
addText "Sample" to clip to top
renderGif clip
```

After this code was run successfully without any problems, it was decided that the tests were being executed correctly, since this would touch every module that was created for the language on a smaller scale.

Conclusion

Video editing tend to be a very expensive process. It takes a lot of knowledge, complex programs and a lot of computational power. Using Videout allows people that do not have any of the above, make their very own simple videos. It allows the use of normally complex functions such as concatenation, text insertion, audio importation and exportation, cropping, resize, etc. This functions are enough to render nice looking videos, in just a few very simple lines of code. It follows the idea of simplifying the process by taking just the necessary amount of tokens as input in a very high-level language aspect.