Audio Color Documentation - Max Figura - 2024/10/15

The process used to create the videos here:

1. Acquire/create a full-color base video and the full audio that should be played over it, in valid FFmpeg formats. Also acquire/create an audio file in mono-channel WAV format for each track to be visually incorporated in the video.

2. Open `audiocolor.py`, set `filename` to that of one of the WAV files of choice, adjust the normalisation rate if desired, then run the program to create a command list corresponding to the audio's envelope. Use the `similarity` attribute of `colorhold` for one track and `colorkey` for all others.

3. Make sure the resulting `.cmd` files, full video, and full audio are all in the same directory. Then within that directory run the following, adjusting the colors and the number of track as desired:

```
ffmpeg -i FULL_VIDEO.mp4 -filter_complex "split=3 [in1][in2][in3];
[in1]sendcmd=f=PART1.cmd,colorhold@b=0x0000ff:0.01:0[out1];
[in2]sendcmd=f=PART2.cmd,colorkey@g=0x00ff00:0.01:0, negate='a'[out2];
[in3]sendcmd=f=PART3.cmd,colorkey@r=0xff0000:0.01:0, negate='a'[out3];
[out1][out2]overlay[out12]; [out12][out3]overlay[out]"
-i FULL_AUDIO.wav -map "[out]" -map 1:a:0 FINAL.mp4
```

   In order, this:
   (a) Copies the video input to the number of tracks
   (b) For the first track, flattens all colors not similar enough to the first color to greyscale
   (c) For each subsequent track, sets alpha to 0 of all colors not similar enough to the corresponding color

   (d) Overlays each track above each other, with the first on the bottom, such that all pixels not similar enough to one of the desired colors are shown as greyscale
   (e) Applies full audio input to resultant video

4. Experiment! There are several possibilities for different effects created with the same engine that I never got around to trying. One could potentially map audio to any FFmpeg filter easily enough, and similar methods could be used to attempt automating filters according to some other timeline.