COMP 50CP Project: Parallel Video Content Matching

Matthew Yaspan, Isabelle Sennett, Mitchell Katz, Alex King

Design Story

For our final project, we will be implementing a *video content matching* program; effectively a search-by-video search engine. This is an interesting and open problem in computing, and it has very practical applications in areas such as copyright protection.

To match the content of video, there are several possible approaches. Many videos include video and audio, so some sort of programmatic matching of video frames and/or audio needs to take place. We have chosen to focus on video content matching rather than audio content matching. In order to detect similarity between two videos, the content needs to be mathematically represented, and two representations need to be evaluated for closeness. One such way of matching video content is with motion vectors, which are a way of detecting position of objects and their movement during a video. This is a powerful mechanism that leads to versatile matching, but it is complex and beyond the scope of what we feel we can accomplish in this project's timeframe.

Instead, we have chosen a more straightforward approach: match videos by matching a sequence of frames as images. If we can define similarity between two still images, we can average it over a sequence of images and make a prediction as to whether two videos share the same content or not. This pushes our solution to rely on one of several different image similarity algorithms. We want to strike a balance between designing the algorithm ourselves and achieving comparison accuracy that is as good as possible.

We have a few ideas for image comparison modules, and we plan on implementing increasingly complex ones as time allows. The first and most naïve module will be a root-mean-square error comparison much like the *ppmdiff* program from COMP 40. This is a brute force mathematical similarity between two images of the same size. It works well for detecting subtle changes due to compression, but can easily fail if slight transformations are applied to the image. A second, more resilient algorithm will be a histogram-of-quadrants approach, where subsections of an image are tallied for color distribution, and each quadrant is compared for similarity. A third, further resilient algorithm would focus on more attributes of an image, such as edge and point detection. This type of comparison becomes more complicated. We will not demonstrate the use of any predefined image comparison libraries unless we have already designed a few of our own.

We will store a local database of videos that a producer will scan and analyze for scannable subsections of a proper size, which will become jobs. Multiple worker threads will process these jobs, searching small subsequences of the video database, checking for similar frames and returning an overall score of similarity, which will then be "interpreted" by our program, eventually returning a score and information to the user.

We are using Python, with the MoviePy library to use FFMPEG for video frame decoding. We will compare images directly as NumPy RGB arrays. We will use the multiprocessing module for

better parallel processing speed. We currently plan on using the Flask framework to make a web-based GUI to interact with our comparison engine.

We have learned about the multiprocessing mechanisms available to us for process communication. Right now we are using the *Value* class within the frame comparison module. A *Value*, in this case declared as a C long integer type and protected by a multiprocessing Lock, will have a persistent value across multiple Python processes. This allows us to safely compute the difference between two frames while using 16+ unique processes. We will use similar protocols to get return information from our video chunk comparison module.

Organization and Language

We have four modules in our design: *frame_diff*, *chunk_compare*, *chunk_video*, and the *gui*. We define a frame difference module to be a Python interface/function to return a similarity score between two NumPy RGB arrays. This definition will allow us to experiment with multiple comparison modules as time allows. We define a chunk of a video to be a video filename and a starting and ending timestamp. The timestamp may be a *frame number* or an *hours:minutes:seconds:milliseconds* timestamp; it is easy to move between the two, and we need both representations to work with the provided API for FFMPEG. We define a *video database* to be a local directory with one or more .mp4 video files inside.

Class and Object Diagrams

Provided alongside this document.

Timeline and Work Division

We have an RMSE frame comparison module working that operates directly on NumPy frame arrays given to us via the MoviePy FFMPEG API. Our video *chunk_compare* module successfully utilizes this *frame_diff* module to iterate through a video chunk and check for similar frames. We also have a *chunk_video* module that iterates through a directory of video files, creates bounds for each search, and creates a *chunk_compare* process for each section.

We're communicating via messaging online, as well as a private GitHub repository where we are hosting our code.

We have revised our timeline to better reflect our progress towards the end goal.

Timeline:

- 11/9: Initial Design due, lock in Python libraries and frameworks that will be used
- 11/16: Have *frame_diff* module working, have video chunk comparison outlined, have "producer" video comparison module outlined
- 11/23: Refined Design due, GUI mockups in progress, have *chunk_compare* and *chunk_video* modules mostly functional
    - Finish designing and implementing GUI
    - Complete *chunk_compare* and *chunk_video*

- 11/30: Have GUI working with backend
    - Prepare PowerPoint presentation and demo
- 12/7: Be prepared to present our work
    - Present, turn in final report

Possible Improvements

1) Optimizations: heuristics for quitting a sequence search early, matching on only every nth frame, better seeking to specific areas in videos, etc.
2) Other stronger comparison modules, or checking against various video filters (flipping, cropping, windowboxing, speeding up)