Video File Uniqueness Final Project Report

Daniel Barry, Cole Flemmons, Rayhan Hossain, Jack Povlin, and Cole Schwerzler

I. OBJECTIVE

Our team has developed a systematic method for selecting the most unique frame in each video in a collection of videos to serve as its thumbnail. Through this methodology, we hope to provide organization-aware custodians of data a tool to more efficiently index video data by focusing on data content versus traditional metadata.

II. PROBLEM STATEMENT

There is a need to develop an automated means of selecting truly representative thumbnail frames which are distinctive for the each video in a collection. In order to achieve this, two conditions must be met. The first condition is the selected frame must be different from the selected frames from other videos. The second condition is the selected frame must be a “good” representation of the video from which it is chosen. The meaning of “goodness” is chosen to mean distinguished color and geometric features of the frames.

III. MOTIVATION

When trying to recall a scene or event that occurred in a video recording amongst a collection of videos, it can be a challenge to locate the video and temporal location in the video in which the event occurred. This problem is exacerbated by the tendency of files to be organized by metadata (i.e. timestamps, file size, file name), which do not explicitly indicate the events or scenes of the video. Metadata that does accomplish indicating the content, to some extent, is the video thumbnail. Manually traversing a video by jumping to various temporal locations can be tedious and often unproductive. Furthermore, certain online video-hosting websites, such as Youtube, allow users to select a ”custom” thumbnail for video content. This can be misleading because such a ”custom” thumbnail need not be an actual frame from the video it represents, further complicating a manual traversal through videos to find an event.

IV. DATA USE

In this section, we emphasize the methods we used to acquire, process, integrate, and validate the video data.

A. *ACQUISITION*

We utilized one particular means of obtaining data: capturing video gameplay from console video games. These were manually obtained, but this collection method allowed us to collect relatively consistent video data; such metadata as resolution and frame rate were similar. Although our project will operate agnostic to these features, it allows us to solve a slightly more constrained problem: comparing data which has good visual contrast (due to animated nature) and providing an abundant amount of relatively similar data. This was not an automated means for collecting video data, nor does the code we wrote collect the data. However, this is not an issue because the purpose of this project is to serve as a provisional tool to those who are already in possession of a collection of video data. The collection of data we use is merely to assist in the production of a proof of concept.

B. *PROCESSING*

After we collected the video data but before we proceed to the processing, we preprocess it by sampling frames, skipping a constant number of frames between each sample. This greatly reduces the number of frames which will need to be processed while still capturing the main content of the videos. Then we perform a K-means operation (we used k=5 by default) on each frame’s pixels to determine the *most likely* colors to occur. Next we convert each frame to gray-scale to condense the data into a single visual channel – intensity. This is necessary to perform the next step, in which we run a Discrete Fourier Transform (DFT) on the frame and determine the real and imaginary components corresponding to the top five most intense frequencies present in the spectrum. These values provide insight as to the most common geometric properties of the frames. The red-green-blue (RGB) values of the pixels from the K-means step and the real and imaginary components of the most intense frequencies from the DFT step are then sent to our database of frame data.

Once the frame data has been stored in the database, we perform K-means on the first two principal components (PCs) of the sample frames on a per-video basis. Then we retain only the frames whose first two PCs are the most central to each cluster. This leaves only as many candidate thumbnails frames per video as there are clusters. In this sense, we are discretizing the frames to capture the most common scenes in the video and reduce the eventual amount of cross-video comparison operations.

Once the candidate thumbnails for each video are chosen, we then aggregate the original frame data (all dimensions) and once more perform PCA on the data. Then we compute the sum of the distances of each frame of a given video to all other frames of only the other videos. The final thumbnail is chosen to correspond to the largest sum of distances.

C. INTEGRATION

This output of this program is dependent on the contents of the video data in the input collection. That is, in general, the resulting thumbnails vary when different video collections are analyzed. This is due to the distance maximization in the principal-component space, which is discussed later. The reason for this is that the principal components are determined by the data that is present in a given analysis. Thus, the output thumbnails are not necessarily the same as videos are added to or removed from the collection because the program's results are truly data-driven.

D. VALIDATION

Since we use K-means as the primary tool for selecting frames, there is no validation because K-means is an instance of unsupervised learning. In addition, K-means is unstable in the sense that the means have the propensity to converge to different values per trial.

Furthermore, due to the notion of the thumbnails needing to be maximally unique with respect to one another, the results need to be validated by visual inspection by the development team. The difficulty in relying on manual inspection is the scale at which this is viable. A team of humans cannot readily remember hundreds of images and recall if each has a certain threshold of contrast.

V. MODELS AND ALGORITHMS

We will be using image processing to develop an automated method that collects metadata from a video and analyzes the video for distinct frames in it. In doing so, there is a need for a data structure that can accommodate the high dimensionality of the frame data and easily allow this data to be stored and retrieved. In addition, there are segments of code to ensure that the data flow between the routines is possible by maintaining the necessary data formats.

A. STORAGE

As data is extracted from a video it must be stored in a database so it can be referenced easily later. Furthermore we wish to minimize the size of this data structure as much as possible. Video data is typically very large, and for a single video we will be pulling out a sizable number of frames, thus the more we can reduce the amount of data we are holding on to in the database the better. We used the built in python library of SQLite in order to create this database. As SQLite is included by default with all python installations and the database it produces is a single locally stored file, this makes it an optimal tool given who our target user is. Within SQLite we built two tables that we stored data in. The first was a simple list of each video that we processed with an indexed ID number. This table only contains the file name and address of a video. This table is used to maintain a running list of the videos that we have analyzed and to be used for reference later to tell what video a given frame belongs to.

The second table is used to store frame information and has twenty eight columns. Each row of the frame table first and foremost contains an ID number which indexes the list of frames and distinguishes between them. Following that a frame stores the id number of the video it came from. Storing the id number of a video is less space intensive than storing a string of the video's name and this is the primary reason we maintain the first table of videos. Following this information pertaining to the visual qualities of the frame is stored. As was mentioned prior, we reduce the RGB color data of the image to only the five most prominent colors. Each of these colors has a red, a green, and a blue value, meaning that we store fifteen elements. Additionally we are storing the output from our Discrete Fourier Transform operation, which provides an additional five numbers each of which posses both a real and an imaginary component which are stored in separate columns of the table adding the final ten elements for the frame.

As we are generating the data we want to store per frame it is easiest to hold on to it in the form of separate arrays. Thus we had to implement specialized functions that could take in these numerous arrays and then place them in the table correctly. Data for a given frame is always returned as a single list of all twenty seven integers, the index of the red values, the blue values and so on is the same every time thus it is easy to read and process our data. Several other specialized functions were created that would allow various different parts of our program to search the database to get the information they need with minimal effort for other team members.

B. PREPROCESSING AND DATA FLOW

Some additional preproccessing needed to be done before we could start our image processing. Initially, the first preproccessing operation we had considered was noise reduction. Reducing the noise in our video data would produce higher quality video data. However, our goal was to select the most unique frame from a video, and noise reduction would not effect the uniqueness of the frames. Ultimately, we decided that noise reduction was unnecessary and would be implemented if we had more time.

The primary preproccessing operation that we needed to perform was making the data more manageable. Our video data consisted of 30-second to 2-minute videos that were all recorded at 30 frames per second. For a 2-minute video, overall processing could take over 3 minutes to complete. Because our processing involved such expensive operations, we needed to break up the videos into shorter, 5 to 15 second clips. With shorter clips, processing the data would be much faster and allow for easier experimentation. This operation was done by using VLC's record feature to break a larger video up into smaller chunks. Those are all of the preproccessing operations that we performed. Next, we would like to briefly explain the flow and structure of our data.

For each video, we aggregate the sampled frame data into a matrix, the rows of which correspond to the sampled frames and the columns are the attributes which are parsed from the frames. These attributes are explained in detail in the following sections. Once this matrix is constructed, it is passed as an argument into the CLUSTER() routine of the Kmeans.py module. The call to this routine returns the ID's of the thumbnail candidate frames, which are contained in a matrix with the other videos' candidate frames. After this process has been repeated for each video, the candidate matrix contains a collection of subsets of the frame data samples contained by the video-specific frame data matrices. This refined frame data sample matrix is then passed as an argument to the PicVframes() routine of the Kmeans.py module, which returns the final set of thumbnail frames.

C. *IMAGE PROCESSING: PHASE I*

The data pre-processing of our project essentially starts with the image processing phase. Processing the images and extracting the proper information was done in two rounds. The phase I of the image processing can be split into four interrelated sequential sections: take the input video files, extract the key frames from the videos, apply the color quantization using the K-means algorithm, and finally store top five colors' RGB values into the database. Figure 1 explains the four steps for our image processing phase I.

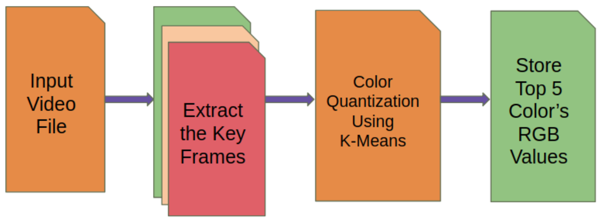
Input Video File
Extract the Key Frames
Color Quantization Using K-Means
Store Top 5 Color's RGB Values

Fig. 1. Image Processing Phase-I steps.

A "keyframe" in videos is a drawing that defines the starting and ending points of any smooth transition. The images are called "frames" because their position in time is measured in frames on a strip of film. A sequence of key frames defines which movement the viewer will see, whereas the position of the key frames on the film, video, or animation defines the timing of the movement. Extracting the information of several key frames might reveal useful information about a given video file. There are several published works for efficiently identifying the key frames from a video. Also, people are putting extra efforts into this domain for better outcomes. However, for the sake of simplicity of development, we developed our simple algorithm to choose the key frames. Every 20th frame is chosen as the key frame from the video.

In the next step, color quantization is applied to the extracted key frames. Color quantization decreases the number of colors used in an image. Color quantization is useful for displaying a multicolor image on devices that support a limited number of colors and for efficiently compressing certain kinds of images. Most bitmap editors and many operating systems have built-in support for color quantization. Popular modern color quantization algorithms include the nearest color algorithm (for fixed palettes), the median cut algorithm, and an algorithm based on octrees. In this project, color quantization is principally done to choose the top five colors which describe the image. To extract the top five colors from a keyframe K-means algorithm is used with the value k=5. Here, the RGB values for the top five centroids represent the most common five RGB values of the image in terms of likeliness to occur. In short, color quantization using K-means converts the original image into a k-color image. Figure 2 shows the result of color quantization for k = 2, 4, and 8.

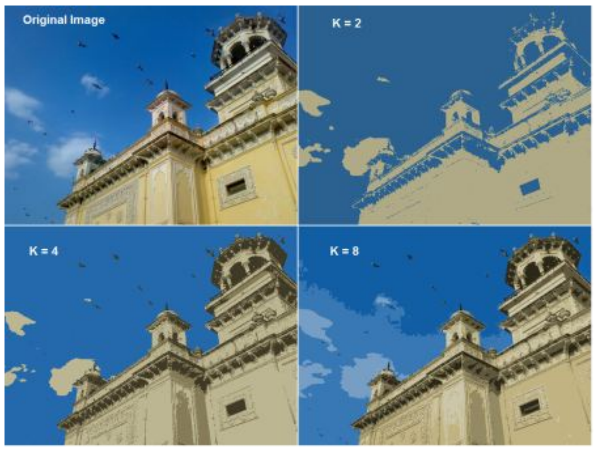

Fig. 2. Color Quantization for K = 2, 4, 8.

After applying the K-means algorithm, a tuple of fifteen elements is generated from the the RGB values of five colors. Finally, one tuple for each frame is saved into the database for later analysis.

D. *IMAGE PROCESSING: PHASE II*

A second and crucial image-processing operation needed was the two-dimensional (2D) DFT. This operation is necessary for this project because it allows one to programmatically inspect the geometric data within an frame.

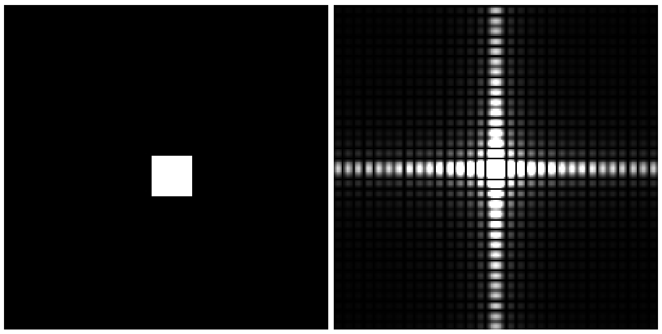

Fig. 3. Example 1 of the 2D DFT operation [1].

In the above example [1], the input image is on the left and the magnitude image of the output of the 2D DFT operation is on the right. The most intense pixels in the output image lie on the most central horizontal and vertical lines because the most prominent geometric features of the input image are the horizontal and vertical edges of the white square. In the above example [1], the input and output images are again on the left and right, respectively. The most intense pixels in the output image lie on the most central diagonal lines because the most prominent geometric features of the input image are the diagonal edges of the white square.

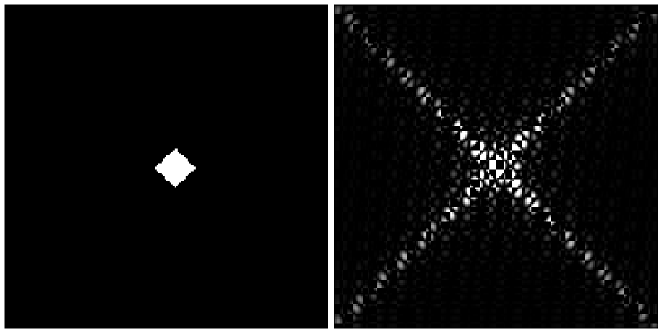
Fig. 4. Example 2 of the 2D DFT operation [1].

There is an implementation of this algorithm provided by the OpenCV Python library, **cv2**. This implementation was chosen for this project due its compatibility with the other OpenCV functions and data structures. The code to apply this operation to a frame is contained in the geom.py file.

The 2D DFT can only be applied to a matrix; however, the frames are color images, which are rank-three tensors, which contain a matrix of each of the red, green, and blue channels of the frame. Therefore, the first step taken by the code in the geom.py module is to convert the color frame into a grayscale image, which has only one channel – the intensity. The implication of this conversion is that the color information is lost, which would be a problem in comparing frames, but the color information does not have significance in the context of examination of the geometric attributes of the frames. Furthermore, the frames' color information has already been processed by the K-means colorspace discretization.

Once the frame has been converted to grayscale, there is one more requirement to be met before the 2D DFT operation can be applied to it. The pixel intensities must be in the interval of [0,1], but the pixels from the grayscale frame are in the interval of [0,255] by virtue of the raw data. This is common for video and image data. To remedy this incompatibility, each pixel is scaled to to within the [0,1] range. Such is simply accomplished by dividing each pixel intensity by '255.0' in the implementation of the geom.py module. This scaling does not scale from the true minimum and maximum pixel intensity values in the grayscale frame. This is intentional because that would require more processing time for each frame, and it is not necessary for the purposes of examining the geomertric features.

Once the scaling is complete, the 2D DFT is applied to the frame. There are two outputs of this operation. The first is the matrix of the real components of the complex-valued geometry spectrum of the scaled-down input grayscale frame. The second is the matrix of imaginary components of the geometry spectrum of the augmented input frame. To analyze the most prominent geometric features of the frame, the complex spectrum is converted to magnitude and phase matrices.

Then, the top five most intense pixels in the magnitude matrix are found. This is done by creating a buffer to contain five floating point values, all of which are initialized to '0.0'. Next, each pixel in the magnitude matrix is traversed. If it is larger than a pixel in the buffer, then the value in the buffer is replaced by it. By the end of the traversal of the entire magnitude matrix, the top five largest intensities are stored in the buffer, and their corresponding real and imaginary values are tracked in two other buffers, each of which contain five values, namely either the real or imaginary components. The real and imaginary components of the complex spectrum corresponding to the top five most intense pixels in the magnitude plot are then returned by the function of the geom.py module, effectively resulting in a total of ten output values.

There are certain cases in image processing wherein there is "noise" in geometric data in a frame. If there is noise successfully detected, then it can be removed by a filter – highpass, lowpass, bandpass, etc. – which is a mask of sorts that gets applied to the magnitude matrix. Once this process is complete, the augmented magnitude matrix can be used with the phase matrix to reconstruct augmented real- and imaginary-component matrices for the complex spectrum. Next, the 2D inverse DFT applied to these matrices can reconstruct an augmented frame with the noise removed.

This was not implemented in the geom.py module for several reasons. The first and most significant reason is that the overarching goal of this project is to select a frame of the video in a collection, not a cleaned-up or noise-reduced version of the frame. Doing so would entail "selecting" a frame that is not truly a frame of the video because it would have been modified before selection. Another reason to not implement this is that it would approximately *double* the amount of time spent in the geom.py module's routine due to the complexity of the 2D inverse DFT being similar to that of the 2D DFT. Such inclusion of a noise removal feature in the geom.py module would also consume more memory for the filter mask and augmented magnitude frames, which might impede the use of this program on computers with less memory available. Intuitively, it does not make sense to include this feature because if there is some type of noise present in all the frames of a video in the collection and not in other videos, then that noise is perhaps a significant feature of that video and *should not* be removed.

The 2D DFT is has a computationally expensive time complexity. A naive implementation of the 2D DFT has a time complexity of O(N^3), and it lends itself well to parallel execution. Fortunately, there is an algorithm called the Fast Fourier Transform (FFT), which is implemented in the **cv2**'s 2D DFT routine, which has a time complexity of O($NlogN$) globally. It is possible that parallel processing could improve the performance of the FFT, but it is not obvious how or whether such could work with the **cv2**'s 2D DFT implementation. An alternative to using the OpenCV DFT could be a custom implementation. The two issues that arise from this are the incompatibility with the other OpenCV functions and possible overall loss of performance due to a less-than-OpenCV-par-optimized custom-made implementation. Since **cv2** is already imported into the code, it makes sense to use its features to minimize the disk space required for the program.

E. PCA, KMEANS, RECALL OF ORIGINAL DIMENSIONS, AND FURTHER PCA

The goal of the K-means clustering is to find frames that are most representative of the video and most distinct from others. Once we have calculated the color and geometry data for every frame of a video, we can use K-means clustering to determine the most representative frames of a video. Using Principal Component Analysis (PCA) we reduce the dimensions of the data into the principal components, and use the first two principal components for each frame to perform clustering. We do this because the data retrieved from previous steps includes 25 dimensions of data. Once the K-means clustering is finalized, we can then select K number of candidate frames for the best representative frame for the video. The frames chosen for this are the frames closest to the center of the finalized clusters, as they best represent the average of the most related frames. This gives us a group of frames from which we can select the best frame, as similar videos might have similar frames. The figure below shows one of these clusters. Once all videos have been processed and their best frames extracted, we perform another clustering algorithm on all of the frames by video. Another PCA reduction is performed on all the frames in order to get them into the same space. Then we find the the frame with the largest sum of distances to all other frames in different videos and select that as the best frame for that video. All other frames from that video are removed from the data set and the process is repeated until all videos have a unique frame.

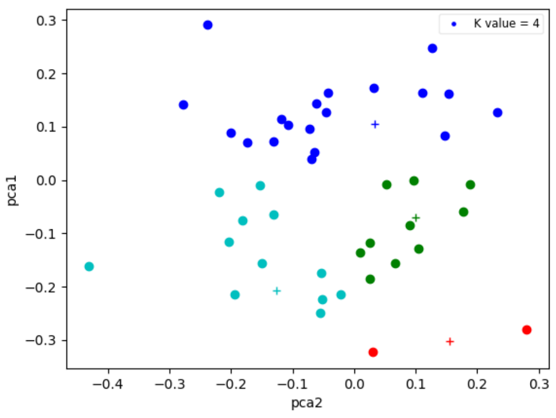

Fig. 5. Results from the K-means clustering on a sample video.

A sketch of the algorithm is as follows:

```
For each video i:
  For each frame j in video i:
    quantize_image(frame j) $\rightarrow database[i][j].color$
    top5geo(frame j) $\rightarrow database[i][j].geometric$
  database[video i] $\rightarrow frame\_matrix$
  CLUSTER($frame\_matrix$) $\rightarrow candidate\_matrix$
PicVframes($candidate\_matrix$) $\rightarrow final\_thumbnails$
For each final thumbnail i:
  Write thumbnail i to a PNG image file on disk.
  Print the thumbnail file name along with the name
      of the video to which it belongs.
```

Fig. 6. The basic format of our algorithmic approach to this problem.

VI. RESULTS

There are two main categories of use cases for the program created in this project. The first is the *heterogeneous* case, in which the separate videos in the collection have relatively *different* geometry and color data per frame on average. The results for this are shown below: In the above figure, the first row corresponds to a three-video heterogeneous collection. The second and third rows correspond to four- and five-video collections, respectively. The thumbnail in the leftmost column changes per trial, but the thumbnails in the second and third columns from the left remain the same. This is due to the leftmost thumbnail from each successive trial having geometry and color that is too similar to the thumbnails in the fourth and fifth column (from the left) – they are collectively too close (in principal-component space) to the frames in each others' videos. Thus, the program performed sufficiently well in the heterogeneous case, and these results seem to improve with *larger* video collections.

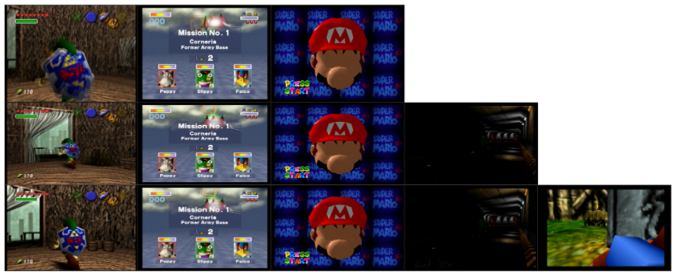

Fig. 7. The output thumbnails of three-to-five-video collections (heterogeneous case).

The second is the *homogeneous* case, in which the separate videos in the collection have relatively *similar* geometry and color data per frame on average. The results for this are shown below: In the above figure, each frame corresponds to a video in a three-video homogeneous collection. Since the collection is homogeneous, this means content of the video was relatively similar in scenery and hence also similar in

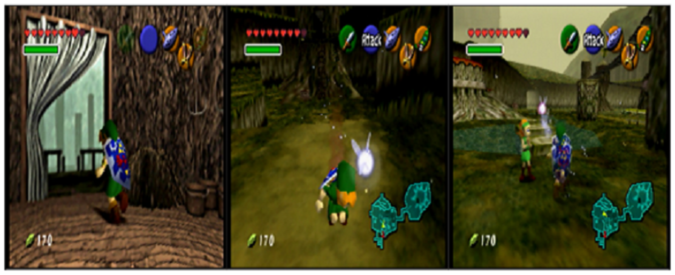

Fig. 8. The output thumbnails of a three-video collection (homogeneous case).

geometry and color. However, the thumbnails are drastically different in color and geometric features. Thus, the program performed sufficiently well in the homogeneous case.

VII. LIMITATIONS AND IMPROVEMENTS

Throughout the development of this project, there were several limitations discovered, some of which can be remedied via improvements. The first and foremost issue is the execution time. The trials used for this report entailed sampling every 20th frame; however, for 12-to-15-second videos, each video takes about three minutes to process. This makes sense due to the amount of image processing taking place, but there is a potential solution to this problem. There are pieces of the code which lend themselves well to parallel execution, so a parallel implementation – possibly even ported to a graphics-processing language – could see some improvement in the sense that the overall execution time would be lowered.

There were also totally black frames and mostly black frames, which are "far" away from other frames of other videos in the principal-component space, but only trivially so. These clearly do not represent meaningful content, which composes the majority of the sample videos in the collections in the experimental trials. In this regard, a potential improvement would be to remove mostly or entirely black frames from the pool of thumbnail candidates.

In capturing some of the sample videos, there was resampling which caused the frames to be interlaced. This resulted in the program selecting frames which were interlaced because geometrically, there are a lot of horizontal lines present in these frames. Thus, a video's actual content may not be accurately represented by the selected thumbnail with interlacing present. There exist many different deinterlacing algorithms, such as discard, mean, blend, bob, linear, x, yadif, yadif (2x), phosphor, film NTSC (IVTC), and others, which could possibly remove the dominating horizontal-line geometric content from the frames, but every deinterlacing algorithm has the potential to remove other content, which may be pertinent to differentiation between two videos, thus skewing the chosen thumbnails unfavorably.

Some further improvements could be the inclusion of a database-access minimizing mechanism and a dynamic hyperparameter optimization mechanism for the frame-sampling rate, the number of principal components to retain, and the number of means for the K-means implementation. The former could possibly shorten the execution time of the program, and the latter could potentially improve the quality of the resulting thumbnails.

VIII. RESPONSIBILITIES AND PROJECT MANAGEMENT

A. *Responsibilities*

- **Workflow and Software Maintenance**:
  Daniel Barry was responsible for ensuring that the development environment was up-to-date and all necessary software libraries and infrastructure were available. He also developed the main driving program for the workflow.
- **Data Analysis**:
  Jack Povlin was responsible for developing the data analysis algorithms and the functional implementation thereof.
- **Image Processing**:
  Rayhan Hossain was responsible for ensuring the function and accuracy of image processing techniques used in the project.
- **Data Management**:
  Cole Flemmons was responsible for ensuring that the data was usable and formatted such that the program could read it.
- **Storage**:
  Cole Schwerzler was responsible for developing and maintaining the data structures needed to complete the project.

The workflow for developing the program in this project was managed by encapsulating each team members' responsibilities in individual module development. Once the data was collected, it was assessed by the the team member tasked with data management. If modifications to the video samples' duration or overall quantity needed to be made, then this member would make the changes accordingly. The Kmeans.py, quantizxn.py, geom.py, and storage.py modules were developed simultaneously. This was accomplished continuously by testing the main driver.py program, which invoked the routines of the modules. If an error or incompatibility occurred or was detected from a routine call from a certain module, then the team member tasked with the function of that module via their respective responsibilities was assigned the issue on GitHub. If there were software packages or libraries need by modules, then those were installed.

ACKNOWLEDGMENT

The development team is sincerely grateful for the guidance of Dr. Audris Mockus in the formulation of a solution methodology to accomplish the goals of this project.

REFERENCES

[1] '2D Frequency Domain Filtering and the 2D DFT', 2018. [Online]. Available: https://www.clear.rice.edu/elec301/Projects01/image_filt/examples.html. [Accessed: 9-Dec-2018].