Video Segmentation

A Machine Project in CSC741M

Richard Louie T Orilla

College of Computing Education

University of the Mindanao

Davao City, Philippines

*Abstract*— Video Segmentation process of partitioning a video sequence into disjoint sets of consecutive frames that are homogeneous according to some defined criteria. In the most common types of segmentation, video is partitioned into shots, camera-takes, or scenes. Video Segmentation is used in a lot of fields but particularly popular in thumbnail generation where in which the program will try to obtain key frames and consider one of them as the thumbnail to be used for the video.

Keywords—Video Segmentation, Key Frame, Average Histogram, Shot Boundary

# Introduction

 Video files are electronic representation of moving visual images in the form of encoded digital data. Video files contain thousands of images that is stored in a compressed form in order to reduce file size. In information retrieval a process called video segmentation is used as a means to partition video into a series of important sequences that is happening on a video (Matthias Grundmann, n.d.).

In this project, a video segmentation program is implemented on a video. The frames contained in the video file are extracted which then is feed to the program and processes the extracted frames in-order to obtain its histogram which is then used to compare its histogram to other neighboring frames.

The result will then be used to obtain the threshold value which is then be used to determine whether or not the frame is a key-frame and if it is such it will then be displayed in the program.

# Design and Implementation

The retrieval of frames in the video is extracted through the use of FFMPEG where each frame is grabbed and stored into a particular folder (Documents\Cache). After FFMPEG gives the exit signal, the program will then traverse the directory and insert every image it can find to the program’s memory.

Each iteration will then have the series of images have its histogram computed by first resizing the image to 100x100 which is done in order to speed up the histogram computation processes. The next step is then converting each RGB pixel that the images contains to LUV color space. The Colourful library is used in converting .NET framework’s Color class which is RGB based to LUVColor. Then finally having each colors stored in a bin and then count the number of colors of a particular color is present in the image.

The pre-caching is done by computing the histograms as they’re loaded into the program for future use. The program is slow in loading these frames as it is done the moment the program opens a new video file. The histograms are then used in preparation in computing the difference of each histogram of each frames using the following algorithms

## Histogram Comparison

Histograms are generated by storing each unique LUVColor in a Dictionary and then count the number of unique LUVColors present in the image.

Histogram Comparison is done by subtracting the number of LUVColors present in one histogram to the other. For example, if we have two histograms that contains the following values

|  |  |
| --- | --- |
| Histogram 1 from Frame 1 | Count |
| L | U | V |  |
| 87.75 | -83.01 | 106.8 | 5 |
| 12 | 15 | 23 | 4 |

|  |  |
| --- | --- |
| Histogram 2 from Frame 2 | Count |
| L | U | V |  |
| 12 | 15 | 23 | 2 |
| 5 | 5 | 5 | 1 |

$$SD\_{i}=\sum\_{j=1}^{G}|H\_{i}\left(j\right)-H\_{i}+1\left(j\right)|^{}$$

Using the Histogram comparison formula, we can obtain the difference between the histograms of frame 1 and frame 2 obtaining the result of the following:

$$SD\_{(f1, f2)}=7$$

Where it tries to find the sum of the number of particular colors in one histogram is subtracted to the other histogram. The histograms being compare are from one frame to the next frame. The results of each computation are stored in an array which will then be used to compute the threshold value.

## Threshold Selection

A spike of values will occur during the comparison of histograms. There will be cases where we want to know the threshold value necessary in order to know if the frame is a keyframe or not. The keyframe is determined when the SD value exceeds the threshold value. In order to obtain the threshold value, the following formula is used.

$$T\_{b}= μ+ ασ$$

Where $μ$ represents the mean of all SD’s that is stored in the array, $α$ is the alpha value that is defined during computation and $σ$ is the standard deviation of all SD’s stored in the array.

All frames that contains an SD value exceeding the threshold value ($T\_{b}$) is considered a keyframe and the next frame after that is the boundary shot.

In addition, another threshold, $T\_{s}$ is used, as a second threshold in order to accommodate gradual transitions. $T\_{s}$ is simply $T\_{b}$ with an added value on top. Any frame that is between $T\_{b} and T\_{s}$ are considered transition frames and those that exceed $T\_{s}$ becomes a keyframe. In the program given, $T\_{s}$ is simply the second threshold which is value that can be adjusted during computation.

# Analysis and Results

This section covers the optimization techniques utilized during the implementation of the program.

The first optimization technique utilized is calling FFMPEG.exe from the program using the Process class. This is done instead of using a wrapper function because the wrapper functions are slower than just unpacking all frames in a particular directory and letting the program gather all the unpacked frames in one load time. The following commands are used.



The second optimization made is through the creation of the ImageInfo class. The ImageInfo class contains the histogram of a particular frame. The constructor function of the ImageInfo computes the histogram of the Image and then unloads the Bitmap, effectively reducing memory consumption of the program as it loads each frame. The Bitmap is reloaded only if the frame is question is a keyframe or a transition frame which is made possible by saving the exact file path of the frame before it completely unloads the Bitmap image.

# Conclusion

The program is developed using C# with the .NET framework. Only a few libraries where used during the development and majority of the algorithms are self-made which probably requires few optimizations including the utilization of pre-cached values.

The slow performance of the program is only noticeable during the opening of the media file. In particular, the longer the video is, the longer the user has to wait for finishing the program.

1. Evaluation Results.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **300** | **400** | **500** | **600** | **700** | **800** |
| **Load Time** | 3 minutes | 5 minutes | 7 minutes | 10 minutes | 14 minutes | N/A |
| **Memory Consumption** | 1gb | 1gb | 1.5gb | 1.5gb | 1.5gb | N/A |

Loading 800 or more frames will have very long load times and thus it is not recommended to be used in loading videos that are more than 20 seconds long unless there is a willing to wait in obtaining the results. It should be noted that the executable is compiled as a 32bit program and thus it is limited to the 4gb barrier unless the program is recompiled in 64-bit mode.

If in such case that the program really requires to process long videos, a recompilation of the FFMPEG with a modified command is necessary from –r 29 to –r 1 which means it generates only 1 frame per second instead of 29 frames per second.

It should be noted that there are no more problems in computing the shot boundaries given that the program has moved on after this phase.

# Conclusion

 Even with the optimizations in using FFMPEG and hand-made data structure, the program still shows problems in regarding extracting frames from the video and as such it is much better if the preprocessed frames are obtained instead of extracting the frame and obtaining its histogram in real time.

Thus, it would be better if a script is created that aims to obtain all frames from a video before a program is used to process each frame’s histogram.