Optimizing Number of Keyframes to Improve Compression of MPEG-4 Files

Daniel Wilson
Department of Computer Science,
Earlham College,
dlwilson13@earlham.edu

ABSTRACT

Improvements in video resolution and color mean more hard drive space is required to store the necessary data, increasing the importance of minimizing the storage requirements of video files. The MPEG file format uses keyframes to lessen storage requirements by representing frames as changes from previous frames instead of building the frame from scratch. One problem that currently exists is that it is difficult to determine the optimal number of keyframes to use. Adding keyframes is computationally intensive, and their effectiveness at lowering storage requirements is very inconsistent. In this paper, we perform experiments to evaluate the effectiveness of keyframes in different types of videos, to find the optimal number of keyframes in various situations. We measure the compression ratio for different videos at varying keyframe values, and the time it takes to compress them using ffmpeg. We show that how much a video benefits from keyframes depends on the image content of the video. Finally, we demonstrate that we can apply this knowledge to improve MPEG codecs to better calculate the optimal number of keyframes for various videos.

Keywords

compression, MPEG-4, keyframes, video

1. INTRODUCTION

As better technology becomes available, the quality of videos increases. Resolution, frame rates, and color options can all be improved to provide a better user experience. These advancements come at the cost of data, since higher quality video will require more storage space. This means video files will take up more space on hard drives and take longer to send over network connections. The motivation for this project is to combat these issues by examining how keyframes are allocated in MPEG files to further reduce the space required to store data. While many file formats and codecs have been researched as methods of compressing video files, this project will focus on the open-source project FFMPEG and the MPEG-4 file format due to their accessibility and popularity.

We will be preforming an experiment where we compress various MP4 videos with different numbers of consecutive allocated keyframes. We will record the compression ratio at each number of keyframes, and compare how well different types of videos compress with different numbers of keyframes. We will use this information to develop an algorithim designed to detect the optimal number of keyframes for a video, and allocate an appropriate number of them accordingly.

This paper will be organized as follows. First, we will introduce the concepts of motion compensation, transform coding, and entropy encoding since it is important to establish a basic understanding of MP4 compression to clarify the rest of this project. Second, we will provide detailed description of the problem, the software used, and the design of the project. After that, we will describe the results of our experiments. To conclude, we will show how the results of our experiment can be applied in an algorithm to better allocate keyframes.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

2. RELATED WORK

2.1 Motion Compensation

Since this project deals with the compression of MPEG-4 files, it is critical to understand their architecture. MPEG files consist of three major frame types, which are called I frames, P frames, and B frames [9]. These frames minimize the space required to store a video file through a process known as motion compensation. I frames, often referred to as independent frames, contain all the necessary data to display that particular image on the screen. They do not rely on information from any previous frames to be displayed. P frames, by contrast, take advantage of the fact that a frame in a video is likely to be similar to previous frames [9]. Since the colors and general locations of images are likely to be resemble previous frames, P frames use motion vectors to represent the movement of blocks of pixels. Instead of storing all of the information representing the color of each pixel, P frames are able to represent much of the information in the frame as positional changes from the previous frame. B frames are similar to P frames except that they use bidirectional prediction. This means they not only use motion vectors to predict the motion of future blocks of pixels, but they also use backward prediction to represent the locations of blocks at previous frames. While more predictive frames use less space, it is necessary to use independent frames as

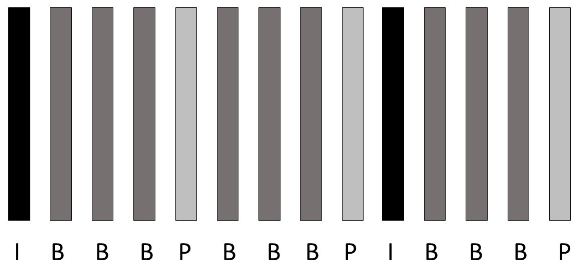

Figure 1: A visual representation of the sequence of frames in an MP4 file [8].

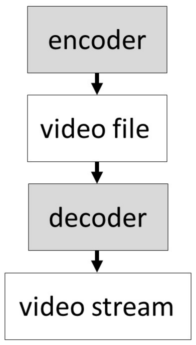

Figure 2: The relationship between the encoder, decoder, and the video the user views [9].

a starting off point from which predictions can be made. Thus, MPEG files contain a sequence of I, P, and B frames, as is exemplified in the figure.

2.2 Transform Coding

Another important compression technique MPEG files use is known as transform coding [9]. Transform coding divides an image into 8 x 8 pixel blocks. Instead of storing the color code for each pixel in the image, transform coding uses a discrete cosine transformation to represent the approximate color for each of the pixels. For small size blocks of pixels, the difference isn't noticeable to the human eye, but it largely reduces file size.

2.3 Entropy Encoding

The final technique MPEG files use to compress data is entropy encoding. Entropy encoding is a method of compression that takes advantage of the fact that certain sequences of bits may appear more often than others by representing more common sequences with fewer bits [9]. It replaces longer sequences of bits that appear often in a file with a shorter sequence specifically meant to represent the longer sequence. This reduces the overall file size, since the most common information requires the least space to store.

2.4 Features

One important aspect of compression algorithms for MPEG files is that they must allow a certain set of core features. Compressing a video file is far less useful if it compromises the ability to use basic features one would expect when watching a video. For example, a good compression algorithm must allow for random access [4]. From a practical standpoint, users watching a video will often want to skip to a particular part in the video. While minimizing the file size may be helpful when storing or transferring the information, the user still needs to be able to access random parts of the video for the compression algorithim to be practical. Other important features of an MPEG compression algorithim include forward searches, reverse playback, audio-visual syncronization, and robustness to errors [4].

3. DESIGN

3.1 Problem Definition

The problem with data compression is that it is nearly always a trade off. A video file, for instance, could be easily reduced to require much less space, but this would require significantly lowering its quality. Conversely, a video that it higher quality may look better, but will take up more space. Data compression algorithms attempt to do a reasonable job of both; they attempt to maintain a decent level of quality while also keeping the file size as small as possible. To address this problem, people are always experimenting with better methods of storing data.

3.2 FFMPEG

The software that we use is called FFMPEG, which is a free, open source, command line software which compresses MP4 files. We will use it as a starting point, and will attempt to further its compression capabilities using the process previously described. FFMPEG contains many encoders and decoders that are used when working with MP4 files. Below is a visual representation of how the encoder, data file, and decoder interact to store and play video. For our project, we are working with the h264 codec.

3.3 Project Design

The use of different frame types is one of the most important ways MP4 files reduce space requirements. Independent frames require more space, but a certain number of them must be included in a file to provide a starting point for prediction frames. Prediction frames, or keyframes, are smaller, but they require an initial starting point to predict from. This creates an interesting interaction between the number of independent frames and the number of keyframes. Too many independent frames increases the size of the file, since independent frames are larger. Too many keyframes can also reduce the file size, since prediction frames become larger is there is less information to use as the starting point as the prediction. This means that there is an optimal number of consecutive keyframes that should be used, where the video file uses as few independent frames as necessary while still using enough to allow the keyframes to be small in size. Our project aims to reduce the size of MP4 files by finding the optimal number of consecutive keyframes for various types of videos.

To accomplish this goal, we have designed an experiment with FFMPEG where we will test videos to find their optimal allocations of keyframes. We have chosen six sam-

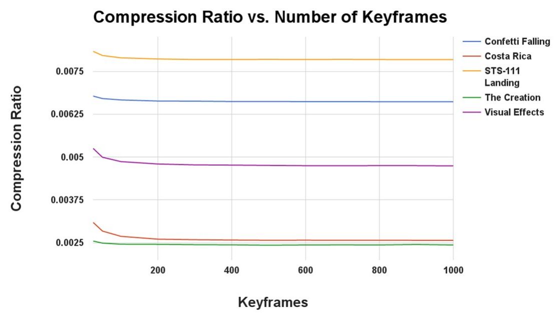

Figure 3: Graph 1: A graph showing the compression ratios for the different types of videos with various numbers of forced consecutive keyframes.

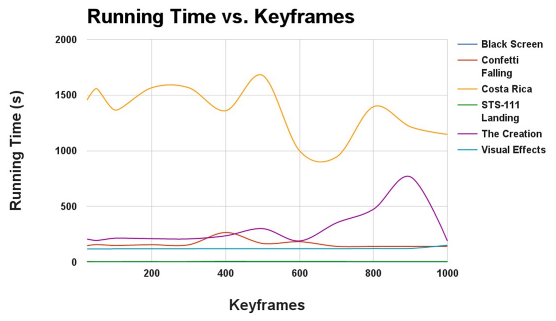

Figure 4: Graph 2: A graph showing the running times for the different types of videos with various numbers of forced consecutive keyframes.

ple videos which vary dramatically in file size, resolution, color, and length. Using FFMPEG and the h264 MP4 codec, we will conduct tests where we compress the sample video files using controlled values for the number of consecutive keyframes. With this information, we will make charts showing how the compression ratio and the time required to compress a file depends on the number of consecutive keyframes used. With this data, we will draw conclusions about how to optimally allocate keyframes for different types of videos. Finally, we will use this information to modify a codec to better allocate keyframes based on the results we find from this experiment.

4. PRELIMINARY RESULTS

When conducting our tests, we recorded the relationship between the number of keyframes used and the compression ratio. We then placed this information in a table to represent the relationship between the number of keyframes and ratio of compression of each video tested. Next, we recorded the run time to encode the video using the h264 codec for ffmpeg each number of keyframes, and compiled this information into the tables displayed below.

4.1 Compression Ratios

Using FFMPEG, we were able to compress the sample video files with specific numbers of consecutive keyframes, resulting in a different compression ration each time. This information displayed in table 1 and graph 1..

4.2 Running Times

We also tracked the time required to compress each file for each number of consecutive keyframes. We placed this information in table 2 and graph 2.

5. CONCLUSIONS

5.1 Average Videos

The videos of confetti falling and visual effects only seemed to benefit from increasing the number of keyframes out to around 250 frames. Due tot the unpredictable changes in the video as the confetti falls, it is more difficult to represent this information using keyframes. Thus, the video has less reduction in file size form increased use of keyframes.

5.2 Running Time

The relationship between the type of video and the time required to compress the file is less consistent, but there are a couple of important conclusions to note. First of all, most videos seem to take the longest to compress at around 400 keyframes, suggesting that we should avoid 400 keyframes when compressing files. Second, most videos took less time to compress around 200 and around 700 keyframes. The run time usually remained low up until 1000 keyframes as well. Thus, we want to attempt to compress within these values when possible.

5.3 Algorithim

Using the conclusions described above, we can approximate the optimal number of keyframes for various types of videos, as is shown in the table below.

Video Type	Optimal Keyframes
Consistent Color	1000
Average Color Change	500-600
Random Color Changes	200-250

We can use this information to improve the quality of compression employed by MP4 codecs. Since the optimal number of keyframes depends on how rapidly and randomly the colors change in the video, we can modify codecs to evaluate the change in colors between frames to figure out how many keyframes to use in different situations. Furthermore, we can program codecs to break the video into specific sections, and dynamically allocate keyframes based on what is optimal for that specific section of the video. A video might have a title at the beginning, for instance, where the colors stay very consistent. After this, it may contain camera footage, but specific parts of the footage may contain more changes in color than others. The codec could choose a number of keyframes for different parts of the video based on our conclusions regarding how many keyframes should be used in certain situations.

To accomplish this task, we have developed the algorithim displayed in figure 5. For each macroblock in frame A, we compare it to the corresponding macroblock in frame B. If they are the same, we increment our counter by 1. After comparing all the blocks in the pair of consecutive frames, we divide the counter by the total number of blocks. For example, if a video had 9 macroblocks per frame, and 2 of

Keyframes	100	200	300	400	500	600	700	800	900	1000
Black Screen	0.03083	0.03068	0.03068	0.03068	0.03068	0.03052	0.03052	0.03068	0.03068	0.03068
Costa Rica	1.844	1.844	1.834	1.867	1.874	2.214	1.870	1.914	1.857	1.991
Confetti Falling	0.06666	0.06634	0.06630	0.06620	0.06621	0.06616	0.06617	0.06614	0.06615	0.06613
Spaceship Landing	0.07894	0.07860	0.07845	0.07845	0.07849	0.07845	0.07849	0.07845	0.0784	0.07841
Greyscale Animation	0.02457	0.02453	0.02443	0.02438	0.02427	0.02435	0.02438	0.02435	0.02448	0.02434
Visual Effects	0.04866	0.04797	0.04771	0.04765	0.04756	0.04748	0.04747	0.04749	0.04749	0.04742

Table 1: A table showing the compression ratios for the different types of videos with various numbers of forced consecutive keyframes.

Keyframes	100	200	300	400	500	600	700	800	900	1000
Black Screen	1.844	1.844	1.834	1.867	1.874	2.214	1.870	1.914	1.857	1.991
Costa Rica	1365.26	1566.24	1566.24	1359.25	1676.90	998.54	943.57	1396.15	1214.76	1147.02
Confetti Falling	149.25	155.44	154.63	265.48	166.66	182.92	140.84	140.84	140.84	141.50
Spaceship Landing	3.895	3.957	3.947	7.038	5.217	5.567	5.145	4.767	4.876	4.908
Greyscale Animation	214.28	209.84	207.69	235.46	300.00	189.25	352.17	474.23	765.59	189.25
Visual Effects	118.09	118.56	119.03	119.50	119.50	119.50	119.50	121.44	122.43	151.50

Table 2: A table showing the running times for the different types of videos with various numbers of forced consecutive keyframes.

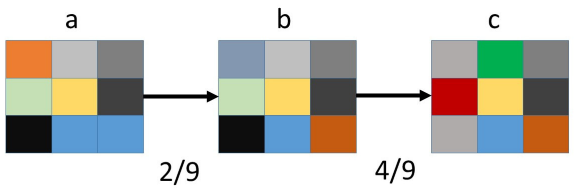

Figure 5: An diagram of the algorithim used to find how much the color in the video is changing.

them changed between frames, then we would calculate a value of 2/9. We would then compare each macroblock in B to the corresponding block in C, and again count the number of blocks that changed. If 4 blocks had changed, then our value for the proportion of macroblocks that changed between frames would be 4/9.

Once we have calculated the proportion of macroblocks that have changed between frames, we will allocate a corresponding number of keyframes to the video. As was described in the previous table, videos with consistent color compress better with greater amounts of keyframes. Thus, a video with a low proportion of change in the macroblocks will be allocated more keyframes. Videos with greater change in color benefit considerably less from keyframes, and thus we will allocate fewer keyframes for videos where we calculate greater values for the proportion of macroblocks that change from one video to the next. We will then repeat this procedure for each section in the video, as is displayed in figure 6.

6. FUTURE WORK

The next step in our project, which is currently being worked on, is to modify a codec to better choose how many keyframes to use based on the contents of the video. We are developing an algorithim that samples a video's frames and compares how a video's color changes from one frame to

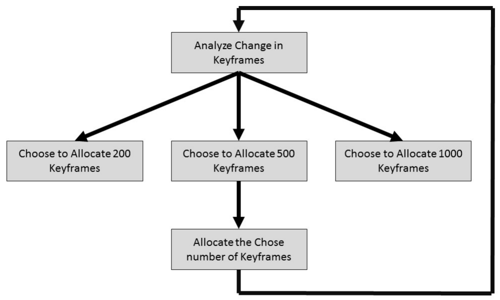
Analyze Change in Keyframes
Choose to Allocate 200 Keyframes
Choose to Allocate 500 Keyframes
Choose to Allocate 1000 Keyframes
Allocate the Chose number of Keyframes

Figure 6: A diagram representing the process by which the algorithim analyzes video an allocates keyframes.

another. Based on this, our code will dynamically allocate frames.

Another area for future work would be to test the repeatably of these results, and perform more tests for a wider variety of videos. This might allow one to find more specific values for the optimal number of keyframes, allowing the data compression to be slightly more efficient.

Finally, videos with more random patterns of color could be tested with consecutive keyframes over 1000. The other types of videos in the experiment did not benefit much from more than 700 keyframes. The videos with more random effects, however, were still considerably reduced in size even around 1000 keyframes. It is possible that the optimal number of keyframes for that type of video may be even greater, so future testing should be conducted.

7. REFERENCES

[1] B. Fazzinga, S. Flesca, F. Furfaro, and E. Masciari. Rfid-data compression for supporting aggregate queries. *ACM Trans. Database Syst.*, 38(2):11:1–11:45, July

2013.

[2] S. Gringeri, R. Egorov, K. Shuaib, A. Lewis, and B. Basch. Robust compression and transmission of mpeg-4 video. In *Proceedings of the Seventh ACM International Conference on Multimedia (Part 1)*, MULTIMEDIA '99, pages 113–120, New York, NY, USA, 1999. ACM.

[3] J. Katto and M. Ohta. Mathematical analysis of mpeg compression capability and its application to rate control. In *Proceedings of the 1995 International Conference on Image Processing (Vol.2)-Volume 2 - Volume 2*, ICIP '95, pages 2555–, Washington, DC, USA, 1995. IEEE Computer Society.

[4] D. Le Gall. Mpeg: A video compression standard for multimedia applications. *Commun. ACM*, 34(4):46–58, Apr. 1991.

[5] D. A. Lelewer and D. S. Hirschberg. Data compression. *ACM Comput. Surv.*, 19(3):261–296, Sept. 1987.

[6] U. Manber. A text compression scheme that allows fast searching directly in the compressed file. *ACM Trans. Inf. Syst.*, 15(2):124–136, Apr. 1997.

[7] R. Mantiuk, A. Efremov, K. Myszkowski, and H.-P. Seidel. Backward compatible high dynamic range mpeg video compression. *ACM Trans. Graph.*, 25(3):713–723, July 2006.

[8] K. Mayer-Patel, B. C. Smith, and L. A. Rowe. The berkeley software mpeg-1 video decoder. *ACM Trans. Multimedia Comput. Commun. Appl.*, 1(1):110–125, Feb. 2005.

[9] K. Patel, B. C. Smith, and L. A. Rowe. Performance of a software mpeg video decoder. In *Proceedings of the First ACM International Conference on Multimedia*, MULTIMEDIA '93, pages 75–82, New York, NY, USA, 1993. ACM.