Cell-unit algorithm of compression for webcam

Tien V. Nguyen
Computer Science major
University of Wisconsin – Parkside
nguyen1@cs.uwp.edu

Hieu T. Nguyen
Computer Engineering major
University of Arkansas – Fayetteville
txn02@uark.edu

Abstract

This paper discusses a compression algorithm for webcam video data. The need for solutions to video conferencing has grown considerably. As a result, so has the need for applications that can transfer data between machines in reasonable time. Today there are many choices for video compression in terms of compression factor, quality, bit-rate and cost. The same idea employed by this algorithm can be applied to process any movie data and can be represented as a series of images. In this paper, we only use webcam movies as a basis to illustrate the experimental results and implementation of our algorithm. The idea is to take advantage of the fact that webcams are generally placed in fixed positions. Significant portions of the space do not change between two frames, so there is no need to retransmit these static portions. The algorithm has a linear run-time so it can combine with any movie or image compression algorithms to produce significantly smaller compressed data. The approach we are going to present has several improvements on how we handle data to produce efficient compression. Every part of data in the algorithm uses bits, with the size multiple of eight so that data will be stored in the least space as all primitive types have the size multiple of bytes. As videos' quality is inversely proportional with the size, we choose a value where both quality and data size are acceptable and also adjustable, depending on users' needs. It allows efficient compression, yet still keeps the quality of the videos so that people can transfer them through small bandwidth networks. At the end of this paper, we include the results of our experiments in which we apply the algorithm in combination with the JPEG image compression technique to compress three typical webcam movies into a size only 5-10% of the original data. In comparison with other movie compression standards, it shows better results in terms of compressed size.

Keywords
Webcam video compression, compression algorithm, heuristics.

Introduction

Video conferencing is becoming a popular method for holding meetings among people spread across diverse geographic locations. Most video conferencing relies on webcams for image collection. Low bandwidth Internet connections make it critical that we have good image compression for webcams. We present an algorithm that is designed specifically for compressing sequences of images generated by a webcam. Our algorithm is efficient and gives good compression when compared with others.

Our Approach

Evaluation of image compression techniques is based on different criteria: the compression ratio**,** the quality of the movies after being decompressed, and the performance speed. However, these factors usually limit each other. The more sophisticated the compression technique is, the slower the performance. Also, a better compression ratio is usually obtained at the cost of lower image quality. Therefore, the decision of which technique to use depends on the nature of the application. Webcam movies do not require high quality images. Thus, it is a better to use techniques that significantly reduce the amount of data transmitted, but still keep acceptable quality. Our goal was to combine the best of these techniques into one algorithm.

Vector Quantization

One idea that we apply is vector quantization.[12] Vector quantization is the process of dividing an image into small rectangular subunits, or unit images, that can be treated individually and recombined as needed. Webcams are usually put in fixed positions so they record images from the same space. We can re-use some unit images from the previous frames for the current frame, based on the similarities between the frames.

Figure 1: Frame is divided into 8x8 unit squares.

First, we divide the images into 8x8 unit images. We will be working on unit images instead of using the whole frame. There are two types of unit images: new unit-images and existing unit-images. Storing and re-using the similar unit images in different frames will reduce the size of the movie significantly.

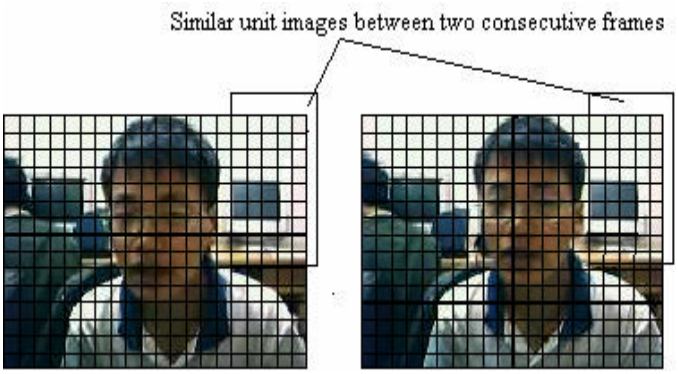
Lempel –Ziv-Welch (LZW) Compression

We also apply LZW compression to our images [11]. "The original Lempel Ziv approach to data compression was first published in 1977. Terry Welch's refinements to the algorithm were published in 1984. The algorithm is surprisingly simple. In a nutshell, LZW compression replaces strings of characters with single codes. It does not do any analysis of the incoming text. Instead, it just adds every new string of characters it sees to a table of strings. Compression occurs when a single code is output instead of a string of characters." [1]

The Algorithm

There are many similar unit images between two consecutive frames.

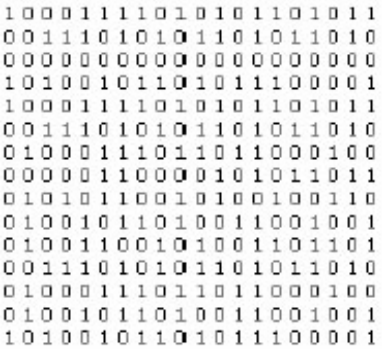
Figure 2: The repetitions in two consecutive frames



Each frame is represented by three elements, depending on the types of unit images it contains:

a. A bitmap of similar-unit distributions: a 2D binary array to determine whether a unit image exists in a previous frame or not (1 for new unit images, 0 for existing unit images)

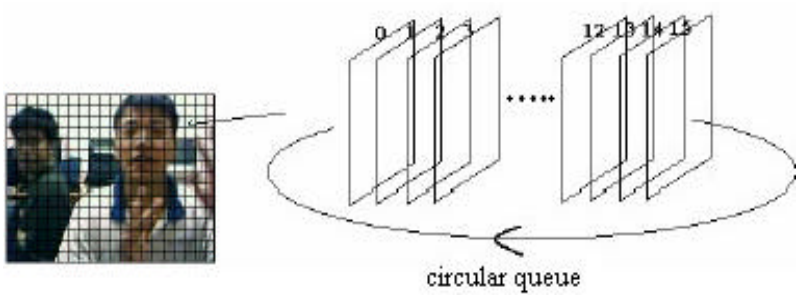
Figure 3: A sample bitmap according to a frame



b. A list of repeating unit images: points each repeating unit image to the similar unit in the previous frames.
c. A list of new unit images: contains compressed information about each new unit image.

In addition to that, a list of "storing images" will be created in order to support finding the similar-units. It is a circular queue with fixed length so we are able to store the latest similar-images for each square unit.

Figure 4: A circular queue for each unit image



The pseudocode to process on frame is:

```
Foreach unit image {
   Find the similar unit images
   If (found)
      Process repeating unit image
   Else
      Process new unit image
}
```

We discuss the details of each part separately.

Finding the similar unit images

We create matrix A to represent the pixels' brightness for each unit image.

The brightness of a pixel is considered according to this formula:

$$\text{Brightness} = 0.299xR + 0.587xG + 0.114xB \qquad [13]$$

Take the following inequality in consideration:

$$\sum_0^7 \sum_0^7 | Aij - Bij | \leq C \qquad (*)$$

A and B are two matrix representations for two unit images. C is called constant accuracy.

A and B are similar if (*) is satisfied. Two unit images are similar when their matrix representations are similar. The greater C is, the higher possibility that A and B are similar, but increasing C may cause diffusion in video quality.

The identification of similar units is based on the intensity level of the 64 pixels, and color information is ignored. If a similar unit is found, then a 4-bit index is transmitted in place of the 64 pixel values. During reconstruction, the 4-bit index is used to access a block at the corresponding spatial position from one of the preceding frames. In general, this will lead to reconstruction errors since the condition is only for similar, not necessarily identical, unit images. In itself, this would be acceptable, but may lead to cumulative errors, for example in a situation where the lighting conditions are varying. The similarity of units during compression may be small (since lighting would not generally vary greatly over 16 frames) and therefore units could be identified as being similar. The reconstruction process replaces similar units with units from earlier frames. This may occur many times in succession, possibly over many multiples of up to 16 frames, in which case errors would accumulate. However, in our algorithm, C defaults to 500. We arrived at this value empirically. Our experiments found this to be the most acceptable value.

Figure 5: A frame with new unit images only

Process repeating unit images

To store a repeating unit image, we only need 4 bits to represent the index of the similar unit in the list of “storing images”. This is appended to a vector of all repeating unit images we have encountered in this image.

Process new unit images

The new unit images are placed into a blank image frame and will be compressed in JPEG2000 format.

Figure 6: A frame with new unit images

Our data structure translates directly into an external file format.

- The first part contains the bitmap of similar-unit distributions. Values in the bitmap are only 0s and 1s so we only need a byte for every 8 units. It takes 38 bytes for every frame with size 160x120.
- The following part contains 4-bit numbers that represent the repeating unit-images.
- At the end of the file, we write out the list of new unit-images in compressed format. These images are in JPEG2000 format so they have same headers (623 bytes for each image), which we can remove.

Empirical Analysis

Table 1 shows the efficiency of the algorithm according to real data, compared to some other available compressions. All data sizes are measured in standard format of each compression. The algorithm is run on 3 test cases with different sizes. (There is a sample program to show the work.)

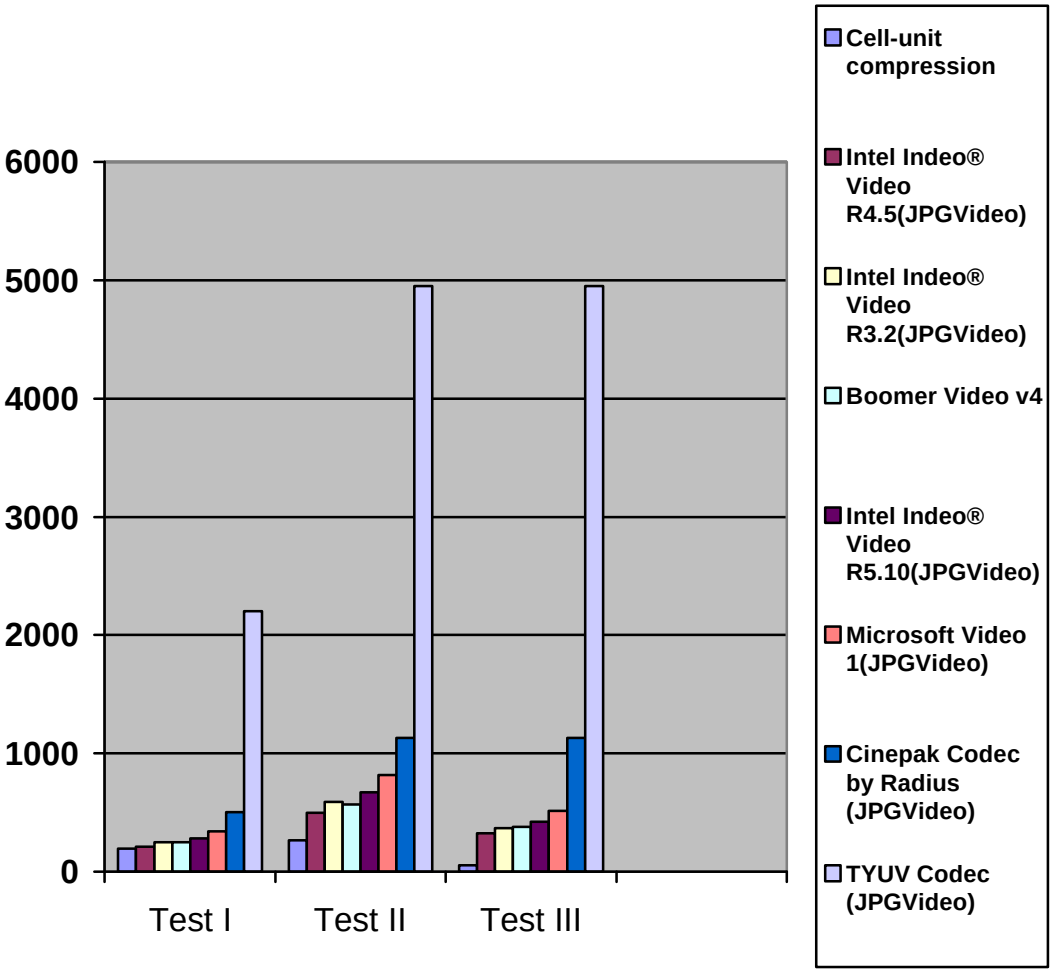
- Test I is 25-second long movie in bright environment with discontinuous motions.
- Test II is 1-minute long movie in bright environment with discontinuous motions.
- Test III is 1-minute long movie in dark environment with continuous motions.

As can be seen in the data, our algorithm compares favorably with the others. The best results are obtained when the similarities between the frames is maximized as in Test III.

Table 1: Space comparison of different algorithms
Numbers are the test sizes, shown in Kb

Video Compression Standard	**Test I**	**Test II**	**Test III**
Our Algorithm	195	264	52
Intel Indeo® Video R4.5(JPGVideo)	207	498	320
Intel Indeo® Video R3.2(JPGVideo)	246	592	369
Boomer Video v4	249	570	377
Intel Indeo® Video R5.10(JPGVideo)	281	670	422
Microsoft Video 1(JPGVideo)	337	816	516
Cinepak Codec by Radius (JPGVideo)	502	1130	1130
TYUV Codec (JPGVideo)	2200	4950	4950

Figure 7: Statistics for 3 sample test cases



References

1. Nelson M., *LZW Data Compression*, Dr. Dobb's Journal, Oct 1989
2. Christopoulos C.A, Askelof J. and Larsson M., *Efficient region of interest encoding techniques in the upcoming JPEG2000 still image coding standard*, Proceedings of IEEE Int. Conference on Image Processing (ICIP 2000), Vol II, pp. 41-44
3. Christopoulos C.A., Skodras A.N., Philips W., Cornelis J., and Constantinides A.G., *Progressive very low bit rate image coding,* Proceedings of the International Conference on Digital Signal Processing, pp. 433-438
4. Anson, Louisa. *Image Compression: Making Multimedia Publishing a Reality.* CD-ROM Professional 6, no. 5 (September 1993) pp. 16-18; 20-24; 26; 28-29
5. Cox, Jennifer, and Mohamed Taleb. *Images on the Internet: Enhanced User Access.* Database 17, no. 4 pp. 18-22; 24-26.
6. Takamura S. and Takagi M, *Lossless image compression with lossy image using adaptive prediction and arithmetic coding*, Proc. Data Compress. Conf., pp. 155-174; 166-174.
7. Gall D. Le, *MPEG: A video compression standard for multimedia applications,* Communications of the ACM, vol. 34, no. 4, pp. 46—58; 305-313
8. Shen K. and Delp E. J., "*A Spatial-Temporal Parallel Approach For Real-Time MPEG Video Compression*, Proceedings of the 25th International Conference on Parallel Processing, pp. II100-II107
9. Shen K and Delp E.J., *A parallel implementation of an MPEG encoder: Faster than real-time!*, Proceedings of the SPIE Conference on Digital Video Compression: Algorithms and Technologies, pp. 407-418
10. Jiang J., *A low-cost content adaptive and rate controllable near-lossless image codec in DPCM domain*, IEEE Trans. on Image Processing, Vol. 9, No. 4, pp. 543-554
11. Kida T., Takeda M., Shinohara A., Miyazaki M., and Arikawa S., *Multiple patterns matching in LZW compressed text.* In Proc. DCC'98, pp. 103-112
12. Gray R. M., *Vector Quantization*, IEEE ASSP Magazine, pp. 4--29, April 1984.
13. Foley J. D., van Dam A., Feiner S. K., Hughes J. F., Computer Graphics 2nd edition, 1990.

Acknowledgments

We thank Huy Nguyen for useful discussions and Ha Tran for providing us useful references. We are grateful for the helpful comments of Dr Stuart Hansen.