Fast Source-based Dithering for Networked Digital Video

Tuong Q. Nguyen^{*}, Jonathan Kay, Joseph Pasquale

Computer Systems Laboratory
Department of Computer Science and Engineering
University of California, San Diego
San Diego, CA 92093-0114

{tnguyen, jkay, pasquale}@cs.ucsd.edu

ABSTRACT

Source-based dithering is a set of techniques designed to maximize the performance of real-time networked digital video systems that encode and decode video entirely in software. Usually frame grabber hardware presents frames in a 24 bit per pixel (bpp) format. However, most hosts are only equipped with single or eight bit deep displays and thus the color depth of the video must be reduced at some point. If the encoder reduces the color depth, the bandwidth required to carry the video on the network is lowered by a factor of 24 or 3 respectively, and the computational load is lightened on the receiving hosts. The color depth reduction algorithm must be efficient since the resulting frame rate, and thus the degree to which the illusion of motion is preserved, depends on how quickly a pixel can be processed. We use dithering algorithms chosen for efficiency and a contrast enhancement algorithm to improve image quality.

1.0 INTRODUCTION

Most modern real-time networked digital video systems are either completely unable to keep up with real time video or require expensive special-purpose hardware in each video participant; thus, performance is very important to such systems. We discuss a series of technique, collectively called *source-based dithering*, designed to improve the performance of and minimize the loss of video quality in real-time networked digital video systems that encode and decode video in software.

Usually frame grabber hardware presents individual images to a CPU in a relatively “deep” format such as “true color.” The true color format typically requires at least 24 bits to represent each pixel: 8 bits for each of the red, green, and blue components. True color display hardware is expensive, so most hosts are only equipped to display images in an eight bit deep color format or a single bit deep monochrome format. The depth of a 24 bpp image must be reduced to either 8 bpp or 1 bpp for it to be displayed on such machines.

The pixel depth could in theory be reduced at either the source or destination hosts. Depth reduction at the source carries the advantages that the bandwidth required to carry the video on the network is reduced and the processing burden is reduced on receiving hosts. The source host is somewhat compensated for the work of dithering by a reduction in volume of network output processing. Depending on whether the video is reduced to 8 bpp or to 1 bpp, this technique reduces bandwidth by a factor of either 3 or 24 respectively.

It is important that the color depth reduction algorithm be efficient because the resulting frame rate, and thus the degree to which the illusion of motion is preserved, depends on how quickly a pixel can be processed. Thus, we use dithering algorithms chosen for efficiency. Dithering is a technique for reducing color depth by placing a small combination of pixels with different colors within a small neighborhood so that, from a distance, the combination looks like the original color. We use a contrast enhancement algorithm to enhance the resultant picture.

This paper is organized as follows. Section 2 discusses related work. Section 3 presents the dithering and contrast enhancement algorithms. Section 4 describes the performance of our system, including both cost and quality of different dither methods. Section 5 concludes the paper.

2.0 RELATED WORK

A variety of techniques have been developed to reduce color depth, the most notable being quantization and dithering algorithms. Foley⁵, Jarvis⁸, Netravali¹², Stoffel¹⁵, and Ulichney^{17, 18} survey dithering techniques for monochrome displays. Heckbert¹¹ discusses some well-known quantization techniques such as popularity and median cut algorithms for color displays.

The most common way of reducing bandwidth requirements for transmitting digital video is to use compression. The standard formats include JPEG¹⁹, MPEG⁷, and H.261 or px64¹⁰. These standards are now used in a number of experimental research projects, including J-Video², Multimedia Multiparty Teleconference³, the MPEG Software Decoder¹⁴, and the INRIA Videoconferencing System¹⁶. However, non-standard formats for special applications are still being developed, such as that used in NV⁶.

Many existing networked digital video applications require the receiver to do a significant amount of work to decode the received video data. Receiving JPEG or MPEG compressed video requires special-purpose hardware or significant processing power for decompression. Even NV leaves it to the receiver to reduce the color depth if necessary.

Our primary focus is on supporting digital video on inexpensive computers with limited processing and display capabilities. Source-based dithering reduces color depth at the source, greatly reducing the work required at the receiver and lowering the required network bandwidth. This makes video available to many low-end systems that would otherwise be unable to keep up with reasonable-quality digital video.

3.0 DITHERING AND CONTRAST ENHANCEMENT ALGORITHMS

3.1 System Overview

Processing video from an analog source, e.g. a camera, to display in some reduced format on a workstation requires a number of stages (see Figure 1).

Figure 1: Block diagram of processing stages for encoding and decoding video in our system.

In our system, video is captured from an analog source such as a camera or a VCR by a RasterOps TX/PIP frame grabber. The frame grabber scales the video image to a desired size and digitizes it into a 24-bpp format with 8 bits for each of the red, green, and blue, components.

For monochrome dithering, there is a preprocessing stage where the gray-scale intensity value is computed from the RGB components. These gray-scale intensity values are then dithered to produce a monochrome bit-map image.

For color dithering, the digitized image is reduced from the 24-bpp RGB format down to 8 bpp. Each resulting 8 bpp encoding actually represents an index into a color map. Using dithering techniques to reduce color depth yields a good balance between performance and quality. This is in contrast to using quantization techniques such as Popularity or Median Cut algorithms¹¹, which are based on significantly more time consuming algorithms.

After the dithering stage, the resulting data can then be compressed (e.g. using an entropy compression method). However, this would require both the source and receiver to do more work. Even without additional compression, source-based dithering yields a 3:1 reduction in size for color, and 24:1 for monochrome. The use of compression methods in addition to dithering is outside the scope of this paper.

The stages described above reside at the source and constitute the video encoder. The last stage constitutes the decoder, located at the receiver. If the data is a monochrome bitmap image, it can be immediately displayed using the X Window System XCopyPlane() procedure. If the data consists of indices to a color map, the decoder performs the lookup in the display process and displays the resulting image.

3.2 Preprocessing: Converting RGB to Gray Scale Intensity

The preprocessing stage is only used for monochrome dithering. We must first compute the gray scale intensity value from the RGB components. The intensity is given by the 3-dimensional distance function (EQ 1).

$$I = \sqrt{\frac{R^2 + G^2 + B^2}{3}} \quad (\text{EQ 1})$$

However, computing the square root is expensive. The rough approximation given by (EQ 2) suffices for most applications. The conversion based on (EQ 2) is often used in image processing to convert from RGB to intensity in the HSI color format²⁰.

$$I = \frac{R + G + B}{3} \quad (\text{EQ 2})$$

This formula requires two addition operations and one costly division operation. However, note that the intensity value is only used in a comparison with a threshold value to determine if a pixel is black or white. By scaling up the threshold value by a factor of three, which can be accomplished at program startup, the division becomes unnecessary. Therefore, this process requires only two additions:

$$I = R + G + B \quad ; \text{ (with threshold scaled up by 3)} \quad (\text{EQ 3})$$

3.3 Monochrome Dithering

Monochrome dithering is a technique that makes use of the property that gray scale levels can be simulated by a series of adjacent black and white dots. There are many algorithms for dithering^{8, 15, 17, 18}. They belong in one of two categories: adaptive and non-adaptive. In an adaptive algorithm, the decision that a pixel be black or white is based not only on that pixel's original value, but also on the values of neighboring pixels. Some well-known adaptive techniques are constrained average and error diffusion⁴.

In a non-adaptive algorithm, the decision that a pixel be black or white depends only on the original value of that pixel. Thus, the computation is simpler and faster. Some well-known non-adaptive techniques include Random Dither^{13,17} and Ordered Dither^{1,9}.

We investigated both random dither and ordered dither techniques because of their speed. In addition, we use contrast enhancement which improves the quality of the dithered image without any additional cost.

3.3.1 Review of Random Dither

Random dither^{13, 17} uses randomness to ensure that blocks of dithered pixels seem to have the same intensity as the original pixels. The assumption is that if the average gray scale intensity I of a block is a certain fraction of the maximum intensity I_{max} , then that same fraction of pixels in the block will be white while the rest are black. This suggests determining a pixel's value according to the following probability function (EQ 4):

$$P\{\text{dithered pixel is white}\} = I / I_{max} \quad (EQ 4)$$

The dither is implemented by comparing the intensity of the original pixel against a threshold chosen from a uniform random distribution. The dithered pixel is white iff the intensity is greater than the random threshold.

In our implementation, time is saved by choosing the random threshold ahead of time for each pixel position. Thus, the same threshold is always used for a pixel at a particular position. Also, since the intensity was computed without dividing by 3, we have to scale up the random value selected from the uniform distribution by a factor of 3 (EQ 5). The resulting distribution is shown in Figure 3a.

$$I_{max} = 3 \times 255 = 765 \quad (EQ 5)$$

The C code for generating the random threshold is

```
threshold(x,y) = random() mod 765;
```

and the C code for determining whether the resulting pixel is black or white is

```
result(x,y) = (intensity(x,y) > threshold(x,y)); /* 1 = white */
```

Thus, the overall cost to evaluate each pixel is the time necessary to complete two additions to determine the intensity value, and one comparison to determine whether the result is black or white.

3.3.2 Review of Ordered Dither

Ordered dither¹ trades off spatial resolution for intensity resolution. This method uses a threshold matrix in which the threshold values are optimally placed. Figure 2 shows an unscaled 8x8 threshold matrix in which the threshold values are computed according to Judith's method⁹.

These thresholds are scaled up to fit in the range of the gray scale intensities 0-765, which are then compared with the pixel intensities (of an 8x8 block) to determine whether the result is black or white. The C code which implements this dither for each pixel is as follows:

```
result(x,y) = (intensity(x,y) > matrix(x,y)); /* 1 = white */
```

The cost to evaluate each pixel is the same as the cost for the random dither: two addition operations and a comparison operation (the two additions come from getting the intensity from RGB format). The ordered dither method generally yields a better quality image than the random dither method.

3.4 Contrast Enhancement

If dithering to a single bit is done just as described above, the resultant quality is not very good. The random method would yield a very "muddy" image, in which details are obscured by random noise, as in Figure 4a. The

0,	32,	8,	40,	8,	40,	16,	48
48,	16,	56,	24,	56,	24,	64,	32
12,	44,	4,	36,	20,	52,	12,	44
61,	28,	42,	20,	68,	36,	40,	28
12,	44,	20,	52,	4,	36,	12,	44
40,	28,	68,	36,	32,	20,	60,	28
24,	56,	16,	48,	16,	48,	8,	40

Figure 2: Unscaled threshold matrix.

ordered dither method yields better detail, but it introduces regular patterns which are not part of the original image (see Figure 5a).

The idea of contrast enhancement is to increase the likelihood that low-intensity pixels are dithered to black and that high-intensity pixels are dithered to white. This reduces “erroneous” dithering of low and high intensity pixels, and still subjects the middle range of intensities to the dithering algorithm.

To incorporate contrast enhancement into the random dither algorithm, we alter the uniform random distribution so that all intensities below a lower cut-off point are evaluated to black and all intensities above an upper cut-off point are evaluated to white (see Figure 3b). The cut-off points are tunable parameters.

Figure 3a: A uniform probability distribution. This distribution yields a “muddy” picture with low contrast.

Figure 3b: A contrast enhanced probability distribution.

Contrast enhancement is incorporated into the ordered dither algorithm in a similar way. The threshold matrix is scaled so that the threshold values are within the mid-range of the distribution.

Figure 4a shows the results of random dither without contrast enhancement, and Figure 4b shows the results with contrast enhancement. The contrast enhancement removes many of the erroneous artifacts resulting from random noise and produces an image with finer details.

Figure 5a shows the results of ordered dithering without contrast enhancement, and Figure 5b shows the results with contrast enhancement. The contrast enhancement removes many of the regular patterns typical of ordered dithers and shows better detail.

Figure 4a: Results of random dither algorithm without contrast enhancement. Random noise makes the image look “muddy” and obscures many details.

Figure 4b: Results of random dither algorithm with contrast enhancement. Much of the random noise is eliminated and details are easier to distinguish

3.5 Color Dithering

Color dithering uses the same general principles as monochrome dithering. However, instead of using the overall intensity of a pixel, each of the red, green, and blue components is used in a separate color component dithering computation. Recall that each color component has 8 bits describing its level of intensity, which means that 256 different levels of intensity may be described. Our algorithm dithers each color component from 256 levels of intensity down to 5 levels. Since there are 3 components, this results in 125 (5^3) colors per pixel. Using 125 colors requires just under half of the entries in a typical 256 entry color map, thus leaving the rest of the unused entries for colors defined by other applications.

The algorithm works as follows. For each 8-bit color component, the top two bits are extracted and the remaining lower 6 bits are dithered to either 0 or 1 by methods similar to those described for monochrome. The dithered value is then added to the top two bits, resulting in a value between 0 and 4 inclusive. The resulting five levels of each color component are packed in a 8-bit result pixel by multiplying by a power of five. This is the base-five analog to using bit-shifting to pack a power-of-two value into a binary word. Below is the C code for the dither to 8-bit color based on random dithering.

```
/* Dither red component */
I = pixel->red;                                /* extract red component */
```

Figure 5a: Results of ordered dither algorithm without contrast enhancement. This image contains dither patterns that are not part of the original image.

Figure 5b: Results of ordered dither algorithm with contrast enhancement. Many of the dither patterns are eliminated. The dark jacket is all black without the dither artifacts visible in Figure 5a.

```

dithered = (I >> 6);                                /* take top two bits */
dithered += ((I & 63) > *++rndnum);                  /* add dithered extra bit */
result = 25 * dithered;                               /* pack into 8-bit pixel */

/* Dither green component */
I = pixel->green;                                     /* extract green component */
dithered = (I >> 6);                                /* take top two bits */
dithered += ((I & 63) > *++rndnum);                  /* add dithered extra bit */
result += 5 * dithered;                               /* pack into 8-bit pixel */

/* Dither blue component */
I = pixel->blue;                                       /* extract blue component */
dithered = (I >> 6);                                /* take top two bits */
dithered += ((I & 63) > *++rndnum);                  /* add dithered extra bit */
result += dithered;                                   /* pack into 8-bit pixel */

```

Instead of dithering, many “cheap” implementations just extract the top 2 or 3 most significant bits from each color component and combine them to make an index into a color map (e.g. a 3-3-2 scheme). This results in a selection of uniformly-spaced colors which is usually suboptimal, as it produces visibly distinct steps in a continuously changing color pattern. Dithering allows for a selection of non-uniformly spaced colors, optimized for the color patterns of a particular image.

4.0 PERFORMANCE MEASUREMENTS

4.1 Experimental Setup

In our study we use a DECstation 5000/240 workstation (27.9 SPECints) with a MIPS R4000 processor running at 40 MHz. The source workstations are equipped with TX true-color frame buffers and PIP frame grabber daughter-cards. The frame grabber provides 24-bit true color frames via the Xv (frame grabber control) extension to the X Window System. Our program actually retrieves them by using the shared memory pixmap extension to the X Window System, which reduces the amount of copying needed to transfer data between frame grabber and user program. Grabbed frames are actually in a 32-bit format - 24 bits of color plus 8 bits of padding to provide word-alignment of pixels. The frame size we capture is 256x256 pixels.

The software is implemented as two programs: an encoder and a decoder (see Figure 1). The encoder applies the algorithms described in this paper to grabbed frames and outputs dithered frames in either bitmap format for monochrome or indices to a color map for color. Output can go to a file or over the network to other machines for decoding. Our decoder runs on DECstations (a 24-bit-deep frame buffer is not required) and Sun Sparcstations. The decoder should be portable to any platform supporting the X Window System, shared memory segments, and the ability to display 8 bit color or monochrome.

4.2 Performance Measurements

Table 1 contains data on our dithering software's performance. The displayed frame size is 256x256 pixels.

Table 1: Dithering Performance

Color Depth	Time to Dither	Maximum Frame Dithering Rate	Actual Generation Speed	Playback Speed
1 bit	0.8 usec/pixel	19 frames/sec	6.3 frames/sec	100 frames/sec
8 bit	1.4 usec/pixel	10 frames/sec	5.2 frames/sec	9 frames/sec

The maximum frame dithering rate shows that it is possible to dither monochrome video at 19 frames per second or color video at 10 frames per second. However, the actual frame generation rates are only 6.3 and 5.2 frames per second, respectively. The main reason for this difference is that moving high volume of video data is very expensive. We found that copying video data from the frame grabber into our memory space consumes 60% of the processing time.

The playback speed exceeds the generation speed, especially for the monochrome format. This is because the decoder need not handle such a large volume of data as the encoder nor does it need to perform dithering.

4.3 Comparison of Different Dithering Methods

We compare the dithering methods discussed in this paper to another well-known algorithm, error diffusion⁴. Error diffusion is seen by many as a method that yields excellent quality images^{12, 15}. We compare cost by how many arithmetic operations it takes to dither one pixel. Three image quality metrics often used in comparing image quality are low frequency rendition, high frequency rendition, and processing artifacts¹⁵. Low frequency rendition refers to how well an algorithm can reproduce a gray scale. High frequency rendition refers to how well an algorithm reproduces fine details. Processing artifacts measures how many extraneous details result as a side effect of the algorithm.

Figure 6 is the result of dithering by error diffusion along 2 dimensions, using the values of 8 neighboring pixels in computing each pixel value. This implementation yields good quality, but it is costly.

Figure 6: The results of a two-dimensional error diffusion dithering algorithm. The image quality is better than the image quality from random and ordered dithers, but this algorithm is much more expensive. It requires 17 addition operations and a division operation to evaluate each pixel.

Table 2 summarizes the cost and performance of the techniques discussed in this paper. Note that random and ordered dither algorithms with contrast enhancement yield comparatively good quality at low cost.

Table 2: Comparison of different dithering algorithms

Algorithm	Low-frequency	High-frequency	Artifacts	Cost
Random dither without contrast enhancement	Full scale	Details are obscured by random noise	Many artifacts from random noise.	1 comparison operation
Random dither with contrast enhancement	Some loss at extreme ends	Less sensitive to random noise	Minimal artifacts from noise	1 comparison operation
Ordered dither without contrast enhancement	Full scale	Details are obscured by dither patterns	Noticeable regular patterns	1 comparison operation
Ordered dither with contrast enhancement	Some loss at extreme ends	Less sensitive to dither patterns	Minimal regular patterns	1 comparison operation
Two-dimensional error diffusion	Full scale	Good detail	Noticeable artifacts in areas of uniform color	17 addition operations and 1 division operation

5.0 CONCLUSIONS

Real-time networked digital video systems intended for inexpensive computers can benefit by the incorporation of the source-based dithering techniques described in this paper. Dithering combined with contrast enhancement provides rapid color depth reduction with good image quality. Furthermore, because dithering is done entirely by the source, network bandwidth and decoder processing requirements are sharply reduced.

6.0 ACKNOWLEDGEMENTS

This work was supported in part by grants from the Sequoia 2000 Project, DEC, IBM, NCR, NSF, TRW, and UC MICRO. We are grateful to John Morse for providing the video frame grabbers and related software. We are also grateful to the referees for their helpful suggestions.

7.0 References

- [1] Bayer, B. E., "An Optimum Method for Two-level Rendition of Continuous Tone Picture," Proceedings of the IEEE International Conference on Communications, Conference Record, p. (26-11) - (26-15), 1973.
- [2] Berc, L., Private discussion, November 1992.
- [3] Chen M.-S., "Multimedia Multiparty Teleconference (MMT) System," lecture, U.C. San Diego, November 1992.
- [4] Floyd, R., Steinberg, L., "An adaptive Algorithm for Spatial Gray Scale," Society for Information Display Symposium Digest of Technical Papers, p. 36-37, 1975.
- [5] Foley, J., A. van Dam, S. Feiner, J. Hughes, **Computer Graphics: Principles and Practice**, Second Ed., Addison-Wesley, Mass., 1990.
- [6] Frederick, R., "Some info on the NV 2.x packet format," rem-conf mailing list, December 1992.
- [7] Le Galle, D., "MPEG: A Video Compression Standard for Multimedia Applications," Comm. of the ACM, Vol. 34, No. 4, p. 46-58, April, 1991.
- [8] Jarvis, J. F., C. N. Judice, W. H. Ninke, "A Survey of Techniques for the Display of Continuous Tone Pictures on Bilevel Displays," Computer Graphics and Image Processing, Vol. 5, p. 13-40, 1976.
- [9] Judith, J. N., J. F. Jarvis, W. Ninke, "Using Ordered Dither to Display Continuous Tone Pictures on an AC Plasma Panel," Proceedings of the Society for Information Display, Q4, p. 161-169, 1976.
- [10] Liou, M., "Overview of the px64 kbit/s Video Coding Standard," Comm. of the ACM, Vol. 34, No. 4, p. 59-63, 1991.
- [11] Heckbert, P., "Color Image Quantization for Frame Buffer Display," Computer Graphics, Vol. 16, No. 3, p. 297-307, July, 1982.
- [12] Netravali, A., B. G. Haskell, **Digital Pictures: Representation and Compression**, Plenum Press, NY, 1988.
- [13] Roberts, L. G., "Picture Coding Using Pseudo-Random Noise," IRE Trans. on Information Theory, Vol. 8, p. 145-154, 1962.
- [14] Rowe, L., K. Patel, and B. Smith, MPEG Video Software Decoder distribution, toe.cs.berkeley.edu, University of California at Berkeley, Nov 1992.
- [15] Stoffel, J. C., J. F. Moreland, "Survey of Electronic Techniques for Pictorial Image Reproduction," IEEE Trans. on Communications, Vol. 29, No. 12, p. 1898-1925, 1981.
- [16] Turlletti, T., INRIA Videoconferencing System, INRIA, France, 1992.
- [17] Ulichney, R., **Digital Halftoning**, MIT Press, Mass., 1987.
- [18] Ulichney, R., "Video Rendering," Digital Technical Journal, Vol. 5, No. 2, p. 9-18, Spring, 1993.
- [19] Wallace, G., "The JPEG Still Picture Compression Standard," Communications of the ACM, Vol. 34, No. 4, p. 30-44, April 1991.
- [20] Wilson, A., "What Color is Color?" Electronic System Design Magazine, p. 38-44, January, 1988.