Image Compression for Memory Constrained Printers

Pamela Cosman
UCSD Dept. of ECE
La Jolla, CA 92093-0407
email: cosman@code.ucsd.edu

Tamás Frajka
UCSD Dept. of ECE
La Jolla, CA 92093-0407
email: frajka@code.ucsd.edu

Kenneth Zeger
UCSD Dept. of ECE
La Jolla, CA 92093-0407
email: zeger@ucsd.edu

Abstract

We study memory efficient image compression techniques for color digital printers. Particular constraints imposed by printers are incorporated into the algorithms presented.

1. Introduction

Limited data transmission rates between a computer and a color printer can be the bottleneck in producing rapid prints of digitized images. In some low-cost personal computer systems today, an inexpensive transmission line between the computer and a printer, supporting transmission rates of about 100 kbits/sec, can impose about 4 minutes of delay time to transmit a 1024×1024 color image at 24 bpp. If such images are sent over a 10 Mbit/sec Ethernet network to a network printer, each image would require about 2.5 seconds to transmit. This can substantially increase congestion on a network where images are transmitted for printing every few seconds, which can occur at peak congestion periods in office environments.

Image compression techniques can greatly speed up the printing process, but there are two difficult constraints that must be met that are not typically both found in other image compression problems: (1) *the amount of memory on-board the printer is limited, and (2) the decompression algorithm must progress from the top of the paper to the bottom of the paper, in the order that the paper emerges from the printer.* Typically, there is also a complexity constraint for the decoder. These constraints are derived from the practical issues of designing competitive low-cost color printers for home use and for businesses.

Wavelet-based compression algorithms such as EZW [1] and SPIHT [2] have excellent distortion-rate performance, but do not meet the memory constraints and sequential processing needs of printers. Their progressive property is not useful here; unlike a video monitor, paper printers cannot return to and change earlier rows, once the paper has begun exiting the printer. The full quality decoding for the top row of an image must be completed before any printing can begin. An important problem is therefore to find good image compression algorithms which can be sequentially decompressed, from the top of an image to the bottom, i.e., as the paper exits the printer. This constraint is already satisfied with baseline sequential JPEG, in which the image is processed in blocks of 8×8 pixels. However, many wavelet-based algorithms have much better distortion vs. rate performances than the JPEG technique. EZW and SPIHT are among the many wavelet algorithms which provide excellent compression performance, but which require a complete image to be reconstructed before any particular part of the image can be printed by a printer.

We present a method for ordering the wavelet coefficient information in a compressed bit stream to allow a decoded image to be sequentially decompressed, from the top of an image to the bottom, as the paper exits the printer. In addition, we use a hybrid filtering scheme that uses different horizontal and vertical filters, each with different depths of wavelet decomposition. This reduces the decoder memory requirements by reducing the instantaneous number of coefficients needed for inverse filtering. We also incorporate a modified version of the quadtree-guided wavelet encoder introduced by Teng and Neuhoff [5] in order to reduce the memory requirement imposed by the zerotree structure.

2. Memory Requirements

Our choice of wavelet based coders is motivated by their superior performance compared to spatial domain or DCT based coders. Certain properties of the wavelet transform and the coder determine the memory requirement at the decoder. To determine the memory requirement notice that only a small subset of the wavelet coefficients is required in the inverse transform to obtain any given output row, without further quantization loss. This is illustrated in Figure 1.

This process can be described in terms of line-by-line printing in the spatial domain of the image. We seek the *minimum set* of wavelet coefficients that must be received by the decoder in order to print out a given single horizontal row in the reconstructed image. After the printer has received that minimal set, and used it to print out one particular row in the image, we then seek the *minimum set of additional wavelet coefficients* that must be transmitted by the encoder so that the following row can be printed. We also determine how much of the currently stored set of coefficients can be purged from memory. Coefficients that can be expunged are those which are not needed to print out a future row which has not yet been printed. This process is iterated for each row until the entire image is printed.

In the case of wavelet based coders three major factors determine this minimum set of wavelet coefficients: the length of the wavelet filters, the number of decomposition levels and the coding algorithm.

The filter length becomes an important factor in the vertical inverse transform. To be able to recover any single line one needs to receive as many lines in the wavelet domain as there are taps on the filter. Some of these lines are in the vertically lowpass filtered band, and some are in the highpass filtered band. Thus shorter filters would help reduce the memory requirements. Unfortunately the use of shorter filters often results in a decrease in image quality.

As noted above, some of the lines that feed the filter taps come from the LL band. In the case where more levels of wavelet decomposition are done, one has to do several levels of inverse wavelet transforms to obtain the lines used at the last level of the inverse filtering. Generally at each level the same filters are used, thus contributing the same number of rows necessary at each level (although the rows get shorter horizontally in the lower frequency subbands of the decomposition).

In addition to the memory requirements induced by the nature of the wavelet transform, the order in which the wavelet coefficients are coded also influences the memory needs. Because zerotree-based coders tend to send quantization information that pertains to large groups of coefficients at one time, the decoder receives and must store information about many coefficients that are not needed to reproduce the first line of the original image. Methods that use context modeling to encode wavelet coefficients would require that the decoder store the coefficients that form the context.

In the sections to follow we propose methods that aim to reduce the memory requirements with these factors in mind.

3. Bit-stream re-ordering and hybrid filtering

One approach to conserving memory is to partition the image into horizontal strips, each containing a small number of horizontal rows, and then to compress each strip independently using an EZW-type algorithm on the rectangular sub-images, with optimal bit allocation among them. The storage requirement for the wavelet coefficients is upper bounded by the size of the horizontal strips. However, this technique yields much poorer compression performance compared to using the original full image as the input [4].

A better approach is to maintain the full-frame wavelet transform, but to rearrange the order of the transmitted bit stream from EZW or SPIHT, so that a reduced amount of memory is needed at any given time [6, 7]. As shown in Figure 1, one could send only the minimal set of coefficients needed to print out the next row. If one is using a zerotree-style quantization, the zerotree dependencies will entail more coefficients being transmitted by the encoder beyond those actually needed by the decoder to print out row k. In EZW, each low-band coefficient has three children; each of those in turn has a 2×2 block of children in the next lower directional subband. This additional quantization information is not needed in the inverse transform to reconstruct the top row of pixels. However, with an EZW-style quantization, one cannot avoid the transmission and storage of this additional information when the top row is reconstructed.

This whole process can be viewed as a pure reordering of the SPIHT or EZW output bit stream as in [8]. In [8], one or more regions of importance are designated and the encoder transmits information about those important regions on an accelerated schedule. In our low-memory printer application, the region of importance can be thought of as the actual row(s) to be printed. Bits are sent in the order such that the decoder receives a succession of "minimal sets" of coefficients corresponding to successive printing rows. In order for the decoder to correctly determine where the information concerning the region of interest ends in the bitstream, a header (usually small) is required; the encoder describes the threshold T_n at which the encoding of the original algorithm (not re-ordered) terminates for the region. The decoder then evaluates, as it decodes each row in the re-ordered bit stream, whether the received bits correspond to a threshold of T_n. Once it passes that threshold, the decoder de-

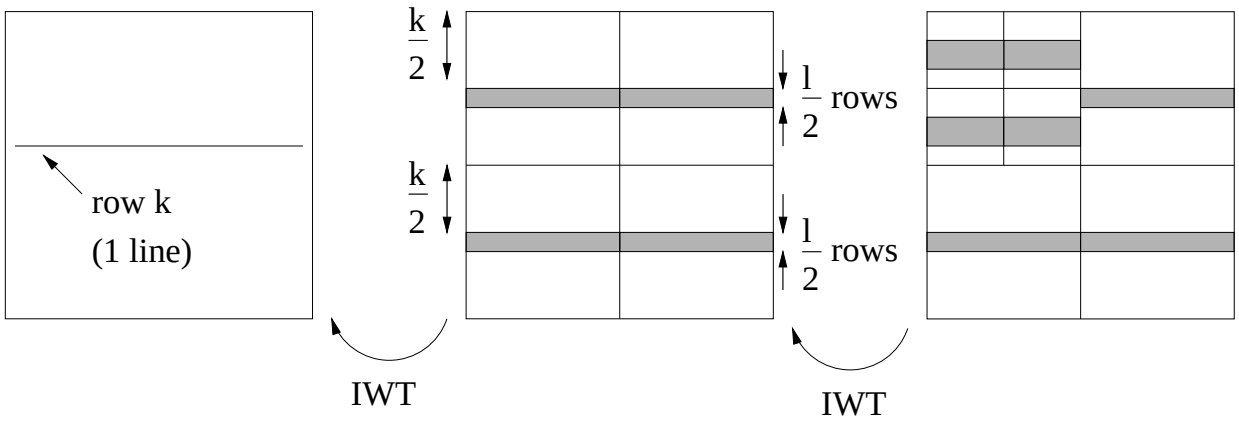

Figure 1: On the left, the printer is ready to print out row k of the output image. The middle picture shows strips of wavelet coefficient rows that are required at the first level of decomposition. On the right are strips required at the second level of decomposition.

duces that the data corresponds to the next row.

3..1 Hybrid Filtering

More wavelet coefficients are required to produce a single output row when either the filter length increases, or when the number of decomposition levels increases. For a 512×512 image, using the filtering and zerotree structure from the SPIHT encoder, 100% of the wavelet coefficient array is involved in producing a single output row. With a single level of filtering with a 2-tap filter, only 0.4% of the wavelet coefficient array is required to produce an output row (512 pixels). With a full 6 levels of decomposition using the same short filter, 64 rows or 12.5% of the array is involved in producing a single output row. However, the decrease in SPIHT performance is very significant when Haar filters are substituted for the 9-7 biorthogonal filters. With a 4-tap filter and 6 levels of decomposition, 25% of the array is involved in the inverse transform to produce a single output row, and with an 8-tap filter, 50% of the array is required. The decrease in SPIHT performance is also very significant when the number of decomposition levels drops below 4.

Filtering in the horizontal direction has no effect on the memory requirements, and thus one can maintain in the horizontal direction the full 6 levels of decomposition using the 9-7 filters. In the vertical direction, we can, for example, perform 3 levels of decomposition with the 9-7 filters, and 3 levels of filtering with the Haar filters of length 2. The total number of levels of decomposition does not have to be the same in the two directions. For example, the vertical direction might consist of 2 levels of decomposition with the 9-7 filters, and 3 levels of filtering with the Haar filters of length 2, for a total of only 5. There exist many different possibilities, each presenting its own trade-off between distortion, rate, and buffering requirements.

4. Quadtree-Guided Encoding

The quadtree-guided wavelet compression technique of Teng and Neuhoff [5] can achieve a substantial reduction in the memory requirements because it uses only a single level of wavelet decomposition.

The encoding of the LL band and the outer bands are different, and in fact the LL band encoding strongly influences the outer band encoding. The LL band is divided into $k \times k$ blocks (typically k is 8 or 16) which are processed in raster scan order. For each block, the lower right corner (called the foot) is predicted from the pixel just above and to the left of the block in the same line or row, as shown in Figure 2. The prediction error is quantized and added to the prediction. Using this foot value, as well as the reconstructed values of neighboring pixels from previous adjacent blocks, the rest of the pixels are predicted using two and four point linear interpolation. A quality test then checks how well the interpolation approximates the real pixel values. If the block passes the test, the quantized prediction error for the foot is transmitted using a variable length code. If not, the block is split into four sub blocks, and the same procedure is repeated for those.

The coding of the outer band relies on the assump-

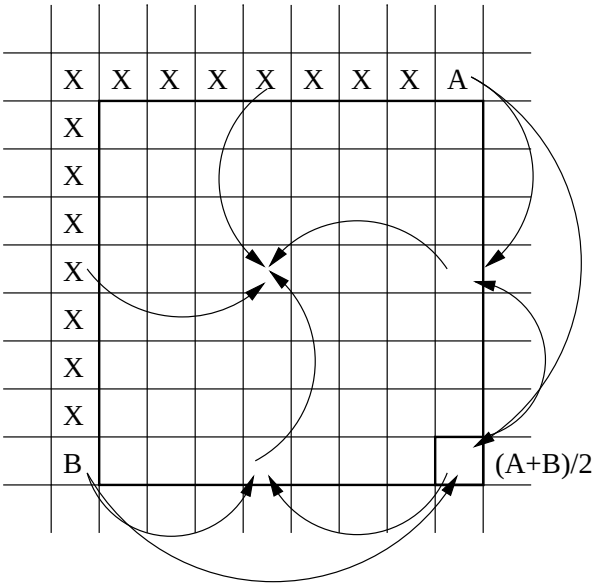

Figure 2: Quadtree encoding: the lower right corner pixel (the foot) is first predicted from pixels A and B, then it is used in the linear prediction of other coefficients, until the whole block is interpolated.

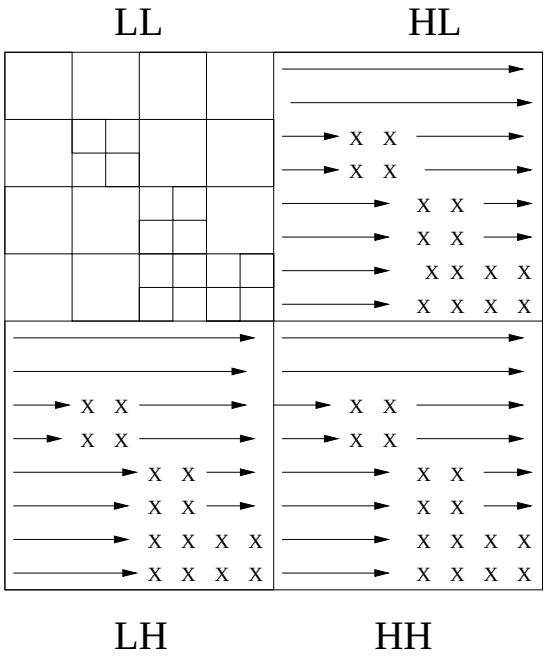

Figure 3: Quadtree encoding: showing the block division in the LL band, the significant coefficients in the outer band (X) and the scanning order (which applies to the significant coefficients only)

tion that if a block is difficult to predict linearly in the LL band, then the corresponding coefficients in the higher frequency bands are significant. A block is declared difficult to predict if the interpolation fails the quality test on a 2×2 block. The high frequency band coefficients corresponding to those highly subdivided LL blocks are quantized with a symmetric scalar quantizer; for all the blocks where the interpolation passed the quality test, the corresponding high frequency coefficients are not encoded (therefore quantized to zero). This is depicted in Figure 3. Finally, those quantized coefficients are runlength encoded. The original algorithm of Teng and Neuhoff used a zigzag ordering similar to JPEG. To make it more suitable for the printer problem, we instead use strictly horizontal ordering for the runlength coding in the outer bands. (For the images tested, this resulted in a 3-8% decrease in outer band rate compared to the zig-zag ordering.) Using this type of encoding one only needs to buffer $2k$ lines to be able to decode one line, where k is the initial block size of the quadtree coding. In the case where $k = 8$, this results in 16 rows of wavelet coefficients being buffered, only 3% of the total array.

5. Results and Conclusions

PSNR results are obtained for the 24-bit color Lena image compressed to 0.24 bpp, for both quadtree-based and zerotree-based systems. For zerotree-based systems the horizontal filtering is maintained at 6 levels of decomposition using the 9-7 filters, as in the SPIHT algorithm. The vertical filtering also uses 6 levels of decomposition, but anywhere from 0 to all 6 of those filtering rounds can be done by 9-7 filters; the remainder use the Haar filter. In the case of quadtree coding, only 1 level of filtering is done using 9-7 filters. The original SPIHT algorithm produces a PSNR of 30.77 dB and requires 100% of the coefficient array in decoder memory. With 6 levels of vertical Haar filtering and the bit-stream re-ordering, a 1.1 dB drop down to 29.60 can be obtained with buffering only 64 rows of wavelet coefficients in decoder memory. With higher memory requirements, a system with four levels of decomposition vertically with 9-7 filters, followed by 2 additional Haar levels, imposes only a 0.3 dB drop compared to SPIHT performance. The quadtree-based scheme, on the other hand, uses only 16 rows of buffering and results in a 1.2 dB drop down to 29.52 dB. It is thus comparable

in PSNR to the zerotree method with all Haar filters vertically, but it uses much less memory and is more visually pleasing. In independent research, Chrysafis and Ortega [3] have recently introduced a related line-by-line wavelet coding scheme; their method involves 5 levels of wavelet decomposition with 9-7 filters and requires 87 rows of wavelet coefficients to be buffered.

We have presented two modifications, bit stream reordering and hybrid filtering, of a well-known class of wavelet-based image compression algorithms and we have presented a modification of an encoding scheme that departs from typical zerotree encoding techniques. Our modifications reduce the decoder memory requirements in a way that is particularly useful for printers. We note that these modifications can allow low-memory encoding as well. These modifications could be used with most wavelet quantization methods currently described in the literature.

Acknowledgments: We thank Stefan Tjärnlund and Magnus Pettersson for their help in running some numerical experiments. We thank Christos Chrysafis and Antonio Ortega for providing a preprint of reference [3]. This work was supported in part by the National Science Foundation and by Hewlett-Packard Co.

References

[1] J. M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, December 1993.

[2] A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, June 1996.

[3] C. Chrysafis and A. Ortega, Line Based, Reduced Memory, Wavelet Image Compression. *Proceedings of the 1998 Data Compression Conference*, pp. 398–407, Snowbird, Utah, March 1998.

[4] M. Pettersson and S. Tjärnlund. Color image compression for color printers using wavelet transform. Master's thesis, Linköping University, Dept. of Electrical Engineering, S-581 83 Linköping, Sweden, November 1997.

[5] C. Teng and D. L. Neuhoff, Quadtree-Guided Wavelet Image Coding. *Proceedings of the Data Compression Conference*, pp 406-415, Mar 1996.

[6] P.C. Cosman and K. Zeger. Memory constrained wavelet-based image coding. In *Proceedings of the First Annual UCSD Conference on Wireless Communications*, pages 54–60, San Diego, CA, March 1998.

[7] P.C. Cosman and K. Zeger. Memory constrained wavelet-based image coding. *IEEE Signal Processing Letters*, to appear.

[8] T. Frajka, P.G. Sherwood, and K. Zeger, Progressive Image Coding with Spatially Variable Resolution. *Proceedings of the International Conference on Image Processing*, pp 53-56, Vol 1., Oct 97.