PROGRESSIVE COLOR BLOCK CODING OF BILEVEL SCANNED DOCUMENTS*

V. R. Algazi, R. R. Estes, G. E. Ford and P. L. Kelly
CIPIC – Center for Image Processing and Integrated Computing
University of California, Davis

Abstract

The efficient coding of scanned documents is a problem of growing importance. The international facsimile transmission standards of 200 dpi, as well as 300 dpi high quality document scanning for computer use, are of interest. We present, and discuss, a progressive coding technique, building upon *color shrinking*, i.e. the gross, data dependent localization of the black portions of an image, that we have presented previously. We propose two intermediate quality standards, 100 dpi binary, and 100 dpi grey scale, which provide high compression, as well as a readable version of documents, specially suited for a computer workstation display.

1 Introduction[1]

The efficient coding of scanned documents is a problem of growing importance. The encoding of facsimile documents for transmission has been extensively studied and a number of techniques have been adopted as international transmission standards by the CCITT [3]. With the widespread use of document scanners interfaced to personal computers, the local storage of such scanned documents has also become common. While in facsimile the international standards of 100 and 200 dots per inch (dpi) allow for marginal quality and legibility, at the local level, 300 dpi is usually required to achieve the desired quality.

In this paper, first we determine the effectiveness of the standard CCITT codes (modified Huffman and READ [3]) as a function of document resolution. Then we examine extensions to the technique of *color shrinking*, i.e. the gross, data dependent localization of the black portions of an image, that we have

*This research was supported by Pacific Bell, Hewlett Packard, and the University of California MICRO program.

[1]There were several erros in the original version of this paper — no attempt has been made to correct them all, so that the paper is close in content to that presented at the ICASSP90. The most significant changes occur in section 6, where most of the numbers have been changed.

presented previously for facsimile, to the case of high resolution scanned documents [4]. For a 200 dpi document, we have shown that the PCSE code, which combines morphological preprocessing, color shrinking and edge point labeling can outperform the CCITT READ code by 30%, on the average [1, 4]. The edge point labeling scheme transforms the original source into a sparse source consisting of the vertical and/or horizontal ends of runs. Thus, the edge labeled source has a high percentage of white that increases with resolution. For higher resolution scanning, e.g. 300 dpi, we propose a pyramid of the information source which allows for a trade–off between the compression achieved and the quality of the representation. A progressive code is then proposed that first generates a color block image that may be useful for identification of the document. Progression in quality corresponds to 100 dpi binary images, 100 dpi grey scale images, 300 dpi binary images, and finally 300 dpi grey scale images. Each stage in the progressive code is based on the data available at the previous stage. We examine the performance possible with such a progressive code.

2 Evaluation of CCITT Standard Codes versus Resolution

We have applied the CCITT standard codes for documents over a range of scanning resolutions. We started with the seven standard CCITT documents and decreased the resolution by subsampling. Thus, for a 200 dpi document, we produce a 100 dpi version by subsampling a 2×2 block to a single pixel. A 2×2 block is mapped to black if any of the 4 pixels is black. The compression ratios obtained using the modified Huffman code (MHC) and the READ code with $k = 4$ are shown in figure 1 for standard CCITT documents 1 and 7, for subsampling factors of 1.0 to 6.0.

A figure of merit (F_m) that can be used to compare performance as the resolution changes is the compression ratio times the subsampling factor. If the coding technique scaled ideally with the resolution, then the number of bits would increase linearly with resolution, since the further resolution is used to improve the boundaries of characters and graphics, which grow linearly in length with resolution. Evaluation of F_m provides two interesting results. The first is that neither the MHC or the READ code are perfectly efficient. Comparing the performance at 100 and 200 dpi, we have $\frac{F_{m,200}}{F_{m,100}}$ of 83.5% for the MHC, 93% for the READ code. Second, a progressive code providing high efficiency at lower resolutions would be very useful in transmitting and displaying lower quality previews.

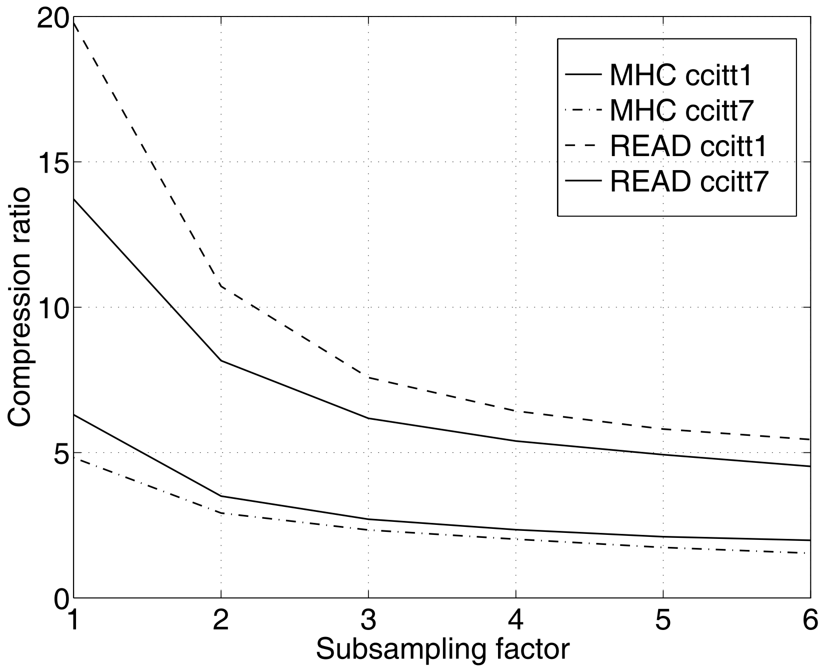


Figure 1: Efficiency of CCITT codes versus document resolution.

3 Progressive Representation

We now consider the progressive representation of documents, either binary or grey scale. Some of the techniques used here are extensions, and specializations, of the methods proposed in [5]. We consider documents, containing text and graphics, which are scanned at 300 dpi. For this scanning resolution, we have 2550×3300, or 8.4 Million, pixels in an $8\frac{1}{2} \times 11$ inch digitized document. This translates into 4.2 Megabytes of data if 4 bits per pixel are used.

Our first objective in the representation and coding of such a scanned document is to develop a set of techniques that trade standards of quality for the total number of bits. A second objective is to provide low resolution, low bit rate, representations of the document, which can be used to determine its content, to read it, to analyze it with some degradation, or to display it on a low resolution display. A final objective is to have a progressive representation in the sense that we begin with the low quality, low bit rate representation and then provide incremental, or progressive, data that updates to improve the representation, Thus, each update complements the previously transmitted data.

We have adopted four resolution and display standards for 300 dpi documents, as shown if table 1, that provide a 36:1 range in the number of representation bytes. As compression techniques are applied to these representations,

Quality Standard		Raw Data Rate (Megabytes)
1	300 dpi, 4 bit grey scale	4.208
2	300 dpi, binary	1.052
3	100 dpi, 10 level grey scale	0.468
4	100 dpi, binary	0.117

Table 1: Raw data requirements of our quality standards.

the number of bytes required will decrease significantly. We illustrate these four quality standards in figure 2.

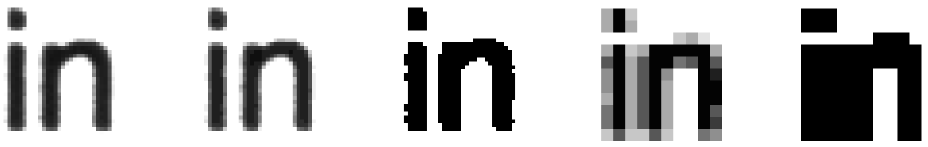


Figure 2: Quality Standards. From left to right, original 300 dpi 4 bit grey scale, 300 dpi with 4 bit grey scale used only on boundaries, 300 dpi binary, 100 dpi 10 level grey, 100 dpi binary

Note that the choice of the 100 dpi, 10 levels of grey standard is quite appropriate for display on computer workstations, where several bits of grey scale are commonly available, but which do not provide the 2550×3300 pixels needed for a 300 dpi scanned document.

4 Alternate Coding Strategies

We consider three different types of coding strategies to achieve high, progressive compression.

4.1 Gross Activity Coders

By localizing the gross activity (color) in a scanned document, the large, contiguous inactive, or white areas can be encoded efficiently. We consider color shrinking, that, for the case of documents, performs better than the classical quad-tree decomposition [1].

4.2 Binary Local Codes

Given that we localize the activity, or color, in a document, the standard CCITT run length and READ codes are less attractive as local codes, because within the color blocks all runs are short and the probability of black and white pixels are much closer to each other than in the original document. For local codes, we have made use, principally, of small block codes, such as a 3×3 block code. When these codes are used in active subareas of a scanned document, they provide, with a fixed code, good and consistent performance, independently of the specific document [1].

4.3 Progressive, or Conditional Codes

We postulate a progressive representation and encoding scheme in which a lower quality image is available and incremental encoding can generate a higher quality representation. For such a strategy, we examine several conditional codes, which are conditioned on the available image. They parallel the designation of quality standards discussed earlier.

- Conditional code $C_{3,4}$
  $C_{3,4}$ is a code for the representation of a 100 dpi, 10 level grey scale image (values 0—9), given that the 100 dpi, binary image is available. Thus, the code only considers the grey scale for the pixels previously labeled as black.

- Conditional code $C_{2,3}$
  $C_{2,3}$ is a code for the representation of a 300 dpi binary image, given a 100 dpi, 10 level grey scale image. Since the grey scale level is the total number of black pixels in a 3×3 array, the conditional code $C_{2,3}$ describes the specific configuration of black pixels, given their number.

- Conditional code $C_{1,2}$
  $C_{1,2}$ is a code for the representation of a 300 dpi, grey scale image given the 300 dpi, binary image. For high resolution document scanning, such as 300 dpi, the grey scale pixels i.e., other than full white or black, will only occur at the boundaries of characters and other black on white contours in the original document. Thus, we study a conditional code, $C_{1,2}$, which detects boundaries and then only generates gray scale codes at these boundaries

5 Performance of Progressive Coders versus Resolution

We consider a progressive coder based on color shrinking, followed by 3×3 block encoding within the color blocks and then conditional codes to update the image to improve resolution. In this section, we discuss the performance of each of the sub-codes, or steps, of a total composite code as a function of resolution.

5.1 Color Shrinking

We denote by color shrinking the simple localization of the activity, or color, of a document within bounding rectangles. We have found that this generalized white skipping technique provides, simply, a substantial reduction in the total area to be encoded, and is specially applicable to scanned documents [1].

Run-length encoding is an efficient method for segmentation of white and black regions in one dimension, but the READ code does not extend this efficiency to two dimensions as well as we would like. Of the many possible ways of representing the blocks in two dimensions, all of which perform the localization of black fairly well, and at low cost, we chose one that would emphasize the natural boundaries in the document. A brief explanation of the color shrinking algorithm used is given below. For a more detailed description, see [1]. Our color shrinking algorithm can be broken into three steps.

Local connectivity. We do not wish to preserve all natural boundaries between characters, symbols, and lines — small gaps can be absorbed into color blocks, and by reducing the number of blocks, we reduce the color blocking overhead. To reduce the number of natural boundaries, we fill all gaps that are within p pixels of each other, either horizontally, or vertically. The parameter, p, depends on the resolution and is taken to be 4 at 100 dpi, and 12 at 300 dpi.

Squaring by projections. To further refine the blocks, we compute horizontal and vertical projections of the black areas within each 64×64 pixel, non-overlapping block. The intersection of these two projections defines rectangular sub-blocks that enclose the color. This process is iterated on each sub-block until tightly bounding rectangles are obtained. See figure 3.

Removing vertices. The last step involves analyzing the image generated by the squaring step , and removing any vertices, from a small set of possibilities, that will improve the final code. Vertices are deleted if their encoding requires more bits than the encoding of the white area they remove (see figure 3). This step typically reduces the number of vertices by 30%.

An example of color shrinking is shown in figure 4.

Color shrinking differs in two ways from a quad-tree decomposition, which also locates the active regions of an image in a progressively finer fashion [2]. Firstly, in color shrinking, the partitioning of the image is driven by the data itself and not by the binary representation of tree nodes, as used in the quad-tree. Secondly, a quad-tree leads to a complete representation of the image to

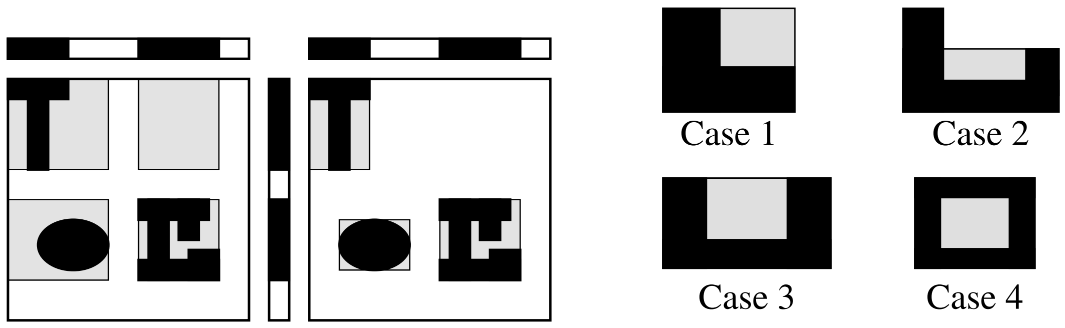


Figure 3: Color shrinking. Squaring by projections (left) and removing vertices (right).

its finest level of resolution. On the other hand, color shrinking is only a partial decomposition, which removes, with a limited overhead to represent the vertices of color block boundaries, a large portion of the white area in a document.

As a first step in a progressive code, color shrinking provides approximately the same compression ratio at all resolutions. That is to say that the same fraction of the total number of pixels will be contained within the color blocks. There are a few second order effects.

1. The number of bits used to describe color block boundaries, n, increases as $log_2 N$, where N is a measure of resolution, in dots per inch. But, n is generally a small fraction of the total number of bits in the code.

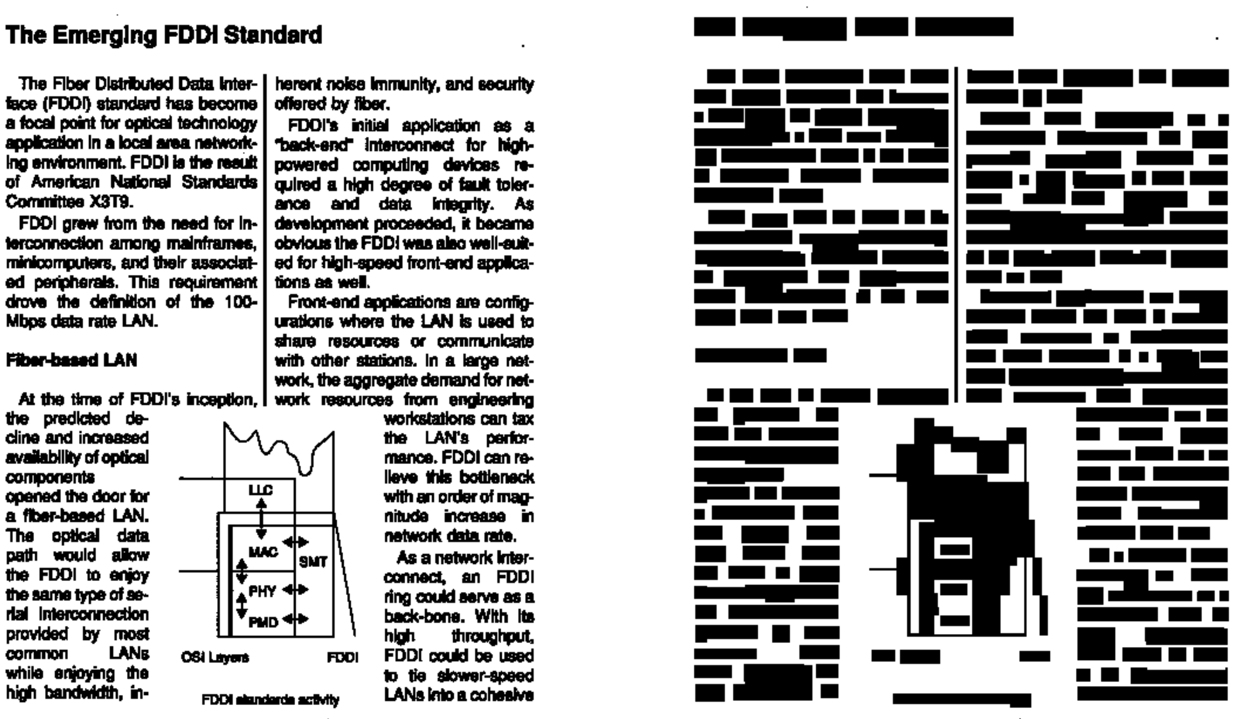
2. The relative area of color blocks decreases slightly with increasing N. This is because subsampling, which preserves all black pixels, will encompass, within color blocks, all 3×3 blocks with at least one black pixel. Thus a white border 2 pixels wide, at the original resolution, may be included in a subsampled area.

3. The color shrinking process can be performed at several levels of resolution, or detail. As resolution decreases, due to subsampling, some details may be merged, and the same level of color shrinking detail cannot be achieved.

5.2 Binary Coding within Color Blocks

Within the color block, the technique we have developed encodes 3×3 blocks of pixels, considered as the information source. We now examine briefly the effect of changing the scanning resolution, N.

With increasing resolution, the number of 3×3 blocks will increase as N^2. The number of all-white, and all-black, 3×3 blocks will also increase as N^2, and because they are on the boundary of solid black areas, the number of 3×3



The Emerging FDDI Standard

The Fiber Distributed Data Interface (FDDI) standard has become a focal point for optical technology application in a local area networking environment. FDDI is the result of American National Standards Committee X3T9.

FDDI grew from the need for interconnection among mainframes, minicomputers, and their associated peripherals. This requirement drove the definition of the 100-Mbps data rate LAN.

Fiber-based LAN

At the time of FDDI's inception, the predicted decline and increased availability of optical components opened the door for a fiber-based LAN. The optical data path would allow the FDDI to enjoy the same type of serial interconnection provided by most common LANs while enjoying the high bandwidth, inherent noise immunity, and security offered by fiber.

FDDI's initial application as a "back-end" interconnect for high-powered computing devices required a high degree of fault tolerance and data integrity. As development proceeded, it became obvious the FDDI was also well-suited for high-speed front-end applications as well.

Front-end applications are configurations where the LAN is used to share resources or communicate with other stations. In a large network, the aggregate demand for network resources from engineering workstations can tax the LAN's performance. FDDI can relieve this bottleneck with an order of magnitude increase in network data rate.

As a network interconnect, an FDDI ring could serve as a back-bone. With its high throughput, FDDI could be used to tie slower-speed LANs into a cohesive

Figure 4: An example of color shrinking. Color blocked version (right) of the original (left).

colored blocks will increase as N. This implies that the probability of a color block decreases as $\frac{1}{N}$.

We now evaluate the entropy of the 3×3 block source as a function of N. Let b, w, b_0 and w_0 designate the all black, and all white, blocks at resolutions N, and N_0, respectively. We have,

$$\begin{aligned} H(s) &= -\sum p_k log_2 p_k \\ &= -p_w log_2 p_w - p_b log_2 p_b - \sum_{k \neq w,b} p_k log_2 p_k. \end{aligned} \tag{1}$$

Expressing H in terms of N_0, b_0 and w_0,

$$H(s) \cong -p_{w_0} log_2 p_{w_0} - p_{b_0} log_2 p_{b_0} - \frac{N_0}{N} \sum_{k_0 \neq w_0, b_0} p_{k_0} log_2 p_{k_0},$$

$$= \frac{N_0}{N}\left[H(s_0) + [(p_{b_0} + p_{w_0}) - 1]\, log_2 \frac{N_0}{N}\right] + \left(\frac{N_0}{N} - 1\right)[p_{b_0} log_2 p_{b_0} + p_{w_0} log_2 p_{w_0}] \qquad (2)$$

Although $H(s)$ depends on the specific document, which determines $H(s_0)$, some numerical evaluations of (2), for $p_{b_0} + p_{w_0} \cong 1$, leads to the conclusion that, to the first order approximation, $H(s)$ scales as $\frac{N_0}{N}$.

5.3 Composite Code

If we use, successively, color shrinking, followed by 3×3 block encoding within the color blocks, then the total compression will depend on the partial performance. If C_s is the compression factor for color shrinking, then $\frac{M}{C_s}$ is the total number of pixels within the color blocks, where M is the total number of pixels in the document. The total number of 3×3 blocks within color blocks is $\frac{M}{9C_s}$. The total number of bits to encode these blocks is then $\frac{M}{9C_s}H(s)$, and the total compression is then $C_t = \frac{M}{codebits} \leq \frac{9C_s}{H(s)}$. The compression of this code, denoted the PCSE code, will thus scale approximately as N, since C_s is independent of N and $H(s)$ varies as $\frac{1}{N}$.

5.4 Conditional Codes

5.4.1 The $C_{3,4}$ Code

For a $C_{3,4}$ code, the resolution of 100 dpi is not enough to confine grey scale pixels to boundaries of black areas. This is due to the fact that gaps between characters can be less than 3 dots. Thus, there appears to be no merit to determining boundaries as a preliminary processing step. We use a simple conditional code. Since we have the binary 100 dpi document, we need to determine whether the black pixels were grey instead of black, and then encode the grey scale value. To indicate that a pixel is black, or grey, will require 1 bit. For grey pixels, we have only 8 possible values, since black and white are ruled out; therefore, they require 3 bits each. This code is essentially an optimum memoryless code if there is a preponderance of black pixels *among the binary black pixels*, and the grey values are equally likely.

5.4.2 The $C_{2,3}$ Code

Given that we have decoded a 100 dpi image with grey scale, we would like to update this image by the information needed to generate a 300 dpi binary image. Thus, the process is to convert all 100 dpi single pixels into a 3×3 pixel array. For all black, or all white, pixels nothing else needs to be done. For grey scale pixels, we need to devise the conditional code $C_{2,3}$, which specifies the configuration of black pixels within the 3×3 array, given their number. We enumerate the possible configurations which requires $\lceil log_2(C_k^9) \rceil$ bits, where $\lceil \bullet \rceil$ stands for the smallest integer greater than $\bullet$. Assuming that all ten grey scales values are equally likely, we compute the average number of bits $N_{2,3}$,

$$N_{2,3} = \frac{1}{10} \sum_{k=1}^{8} \lceil log_2(C_k^9) \rceil = 4.8. \tag{3}$$

This is an upper bound on the average number of bits, which is actually much smaller for the actual data. Without the grey level information, a similar simple representation of the 3×3 binary array would require 9 bits.

5.4.3 The $C_{1,2}$ Code

We now assume that a 300 dpi binary image has been decoded. We would like to provide grey scale information for such an image. For a scanned black and white document, the grey scale pixels are localized on the boundary of black areas. Therefore, we first determine the boundary pixels using the standard morphological, 8–connected boundary operation. Since the boundary locations can be found from the original image, no locational information must be stored.

6 An Illustrative Example

We consider, as an illustrative example, the simple document shown in figure 4 which has been scanned at 300 dpi, with 4 bits of grey scale. This document has 2.87 Million pixels.

6.1 Performance of CCITT codes and the PCSE code at 100 and 300 dpi (binary).

We evaluated the performance of the modified Huffman code (MHC), the READ code (k=4) and the PCSE code for both 100 and 300 dpi. The resulting compression ratios are shown in table 2.

Resolution	MHC	READ	PCSE
100 dpi	2.21	2.81	2.95
300 dpi	4.40	7.40	7.49

Table 2: Relative performance of binary codes.

We note that the performance is rather low for all three codes. In this case the PCSE code does only slightly better than the READ code, but for sparser images it is approximately 30% better, as noted for the CCITT documents [1]. The scaling of compression ratio with resolution is slightly less than N for all codes, with the worst case for the MHC, as expected.

6.2 Performance of the Progressive Code

We evaluate the performance of the progressive code described in this paper. We thus consider the four standards of quality discussed earlier. For each, we determined the incremental and absolute performance, shown in table 3.

100 dpi binary For this readable, lower resolution document, the compression ratio of the PCSE code of 2.95 is with respect to the 100 dpi original. This code thus requires 107,879 bits to code the entire document. With respect to the original 300 dpi binary, the compression ratio is 26.6:1. Note that the color blocks themselves, which allow recognition of the document, require approximately one fifth the number of bits of this code.

100 dpi grey scale This standard provides a very good quality display on a CRT, with 10 levels of grey. Given the 100 dpi binary image, we incrementally provide grey scale information for all the black pixels. As discussed in section 5, we use a simple conditional code. We find here, that for the 71,445 black pixels of the 100 dpi binary document, we need 200,118 bits to specify the grey scale, or 2.80 bits/pixel. The compression ratio from the 300 dpi binary to 100 dpi grey scale is now 9.31:1.

300 dpi binary the 300 dpi binary image is obtained from the 100 dpi, grey scale image by specification of the 3×3 binary pattern for each of the grey scale pixels. The analysis of the conditional code $C_{2,3}$, of section 5, gives an upper bound of 4.8 bits/3×3 block. We have found that the conditional pattern of binary pixels within a 3×3 block, given their total number, are not equally likely. In fact, the favored patterns are for groupings of black pixels, or white pixels, around the borders of the 3×3 blocks. We have evaluated the total number of additional bits needed at 121,517. The composite compression ratio from the original 300 dpi, binary image is now 6.67. This is lower than what can be achieved by a simple one step coding, as shown in table 2, but allows for intermediate and useful standards for images.

300 dpi grey scale We also considered providing grey scale for the 300 dpi binary image. As discussed, we confined the grey scale to boundary pixels determined by morphological operations. We found that the 300 dpi image image has 190,768 boundary pixels. Thus, we require an additional $190,768 \times L$ bits, where L is the number of grey level bits. For $L = 4$, and 2,866,500 pixels in the original image represented at 4 bits, we find a compression ratio of 3.04, if grey scale pixels are only specified at the boundaries of black areas in the original scanned document, and 9.61 if all pixels in the original allow for 4 bits of grey scale.

		Original document			
		100 dpi		300 dpi	
		binary	grey†	binary	grey
Encoded	100 dpi, binary	2.95	5.60	26.6	33.6
	100 dpi, grey	—	1.96	9.31	11.8
	300 dpi, binary	—	—	6.67	8.45
	300 dpi, grey	—	—	—	3.04

Table 3: Progressive coder results. †We assume that only black, or grey, pixels are represented with 4 bits. The total number of grey scale bits is then 604,280.

7 Discussion

We have presented some preliminary results on a simple progressive code for scanned 300 dpi documents. The 100 dpi binary, and grey scale, are useful intermediate quality standards which can be obtained with a low number of bits and are thus useful for previews of documents and good quality display on CRT's. The method of color shrinking, which localizes activity, is a promising method which is related to the quad-tree decomposition. We are currently studying a more general strategy for color shrinking, closer to the formulation of the quad-tree representation, but better suited to the structure and encoding of scanned documents, which have natural horizontal and vertical boundaries, as well as diagonal structures within graphics.

References

[1] V. Ralph Algazi, Phillip L. Kelly, and Robert R. Estes. Compression of binary facsimile images by preprocessing and color shrinking. *IEEE Transactions on Communications*, 38(9):1592–8, September 1990.

[2] Y. Cohen, M. S. Landy, and M. M. Pavel. Hierarchal coding of binary images. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 7:284–98, 1985.

[3] R. Hunter and A. H. Robinson. International digital facsimile coding standards. *Proceedings of the IEEE*, 68(7):854–67, July 1980.

[4] P. Kelly and V. R. Algazi. Two step color shrinking algorithm for the encoding of graphics. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 48.3.1–48.3.4, December 1984.

[5] K. Knowlton. Progressive transmission of grey–scale and binary pictures by simple, efficient, and lossless encoding schemes. *Proceedings of the IEEE*, 68(7):885–96, July 1980.