DIGIPAPER: A VERSATILE COLOR DOCUMENT IMAGE REPRESENTATION

Daniel Huttenlocher and Pedro Felzenszwalb

Dept. of Computer Science
Cornell University
Ithaca, NY 14853
{dph,pff}@cs.cornell.edu

William Rucklidge

Xerox Palo Alto Research Center
3333 Coyote Hill Rd
Palo Alto, CA 94304
rucklidge@parc.xerox.com

ABSTRACT

We describe a segmentation method and associated file format for storing images of color documents. We separate each page of the document into three layers, containing the background (usually one or more photographic images), the text, and the color of the text. Each of these layers has different properties, making it desirable to use different compression methods to represent the three layers. The background layers are compressed using any method designed for photographic images, the text layers are compressed using a token-based representation, and the text color layers are compressed by augmenting the representation used for the text layers. We also describe an algorithm for segmenting images into these three layers. This representation and algorithm can produce very highly-compressed document files that nonetheless retain excellent image quality.

1. INTRODUCTION

In this paper, we describe the DigiPaper document image representation, with an emphasis on using it to store images of full-color documents. A page from a typical document might have a photograph making up a background image, overlaid by some colored text and other graphical elements such as line-art. Storing such a compound image is complicated by this intermingling of different types of data: a pixel representing the color of a text character can be adjacent to a pixel representing a part of the background photograph. This mixing poses a difficult problem for compression: different types of data are intimately mixed together, and compression methods designed for one type do not work well for other types, expanding the file size or introducing unacceptable loss. For example, using a compression method such as JPEG that is based on the discrete cosine transform will not work well for this sort of image (but see [1]). The sharp transitions between background and foreground generate a lot of high-intensity high-frequency components, which are then severely quantized. On decompression, ringing artifacts around the edges of text characters are clearly visible. Alternately, the quantization can be made more fine, but this greatly increases the file size.

DigiPaper uses three techniques to improve both the image quality and file size for compressed document images. These are

- Mixed Raster Content (MRC),
- Token compression, and
- Color tags.

The following section describes these in more detail.

2. IMAGE REPRESENTATION IN DIGIPAPER

2.1. Mixed Raster Content

DigiPaper uses the Mixed Raster Content imaging model, where a page image is represented as a full-color continuous tone (*contone*) background layer, a full-color or limited-color (palettized) foreground layer, and a binary selector layer. Typically, the background layer represents the contone parts of the page, the foreground layer represents the colors of the text and line-art parts of the page, and the selector layer represents the shapes and positions of the text. The page image is reconstructed by merging the background and foreground layers, using the selector to choose between them.

The background, selector and foreground layers are each compressed separately, using different compression methods. In DigiPaper the background is compressed with JPEG, the selector with token compression, and the foreground with color tags. Also, the three layers can be stored at different resolutions. For example, the background layer might be reduced to 100dpi (dots per inch) before compression. In background color images, this resolution reduction is often not apparent, whereas it would be unacceptable to reduce the selector layer (containing the text) to 100dpi.

Since the text has been removed, the background layer no longer contains foreign step edges, and so it is a much more suitable candidate for standard photographic image compression techniques.

Figure 1 shows an image and its decomposition into these three layers. Wherever the selector layer is black, the

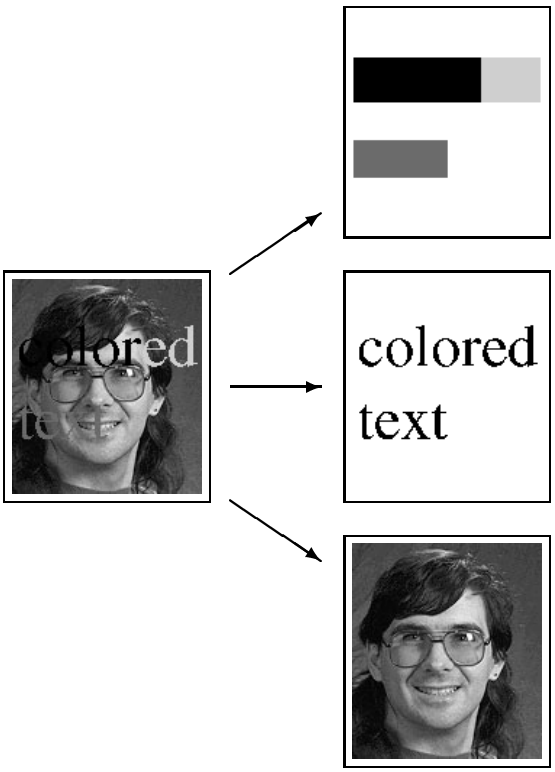


Figure 1: Illustration of MRC

color from the foreground is drawn; wherever the selector layer is white, the color from the background is drawn. One special case of MRC is of interest: if no image is supplied for the background, it defaults to white; if no image is supplied for the foreground, it defaults to black. Thus, a page in a DigiPaper file that has no images supplied for the foreground or background is simply a black and white bi-level image.

The MRC model has recently been adopted for fax as Recommendation T.44 [2] by the International Telecommunications Union (ITU), and for Internet fax as RFC 2301 by the Internet Engineering Task Force (IETF) [3].

2.2. Token compression

A key aspect of the DigiPaper representation is the use of *token compression*, wherein a binary document image is represented using a dictionary of token shapes, together with position information indicating where each token should be drawn. This representation is both compact, storing just a single image of each token, and provides a structured representation for interactive document viewing operations such as cut and paste. Such a scheme was first described by Ascher and Nagy [4] and is the basis of the forthcoming JBIG2 binary image compression standard [5].

Figure 2 shows the effects of performing token compression on the selector layer from Figure 1. The 11 distinct shapes from the selector layer are segmented (using connected components analysis), and then classified into 8 equivalence classes. Exemplars for these equivalence classes are shown, together with the locations of the members of those classes. The exemplars form a *token dictionary*, and this dictionary plus the numerical locations can then be represented in less space than would have been required to represent the original bitmap.

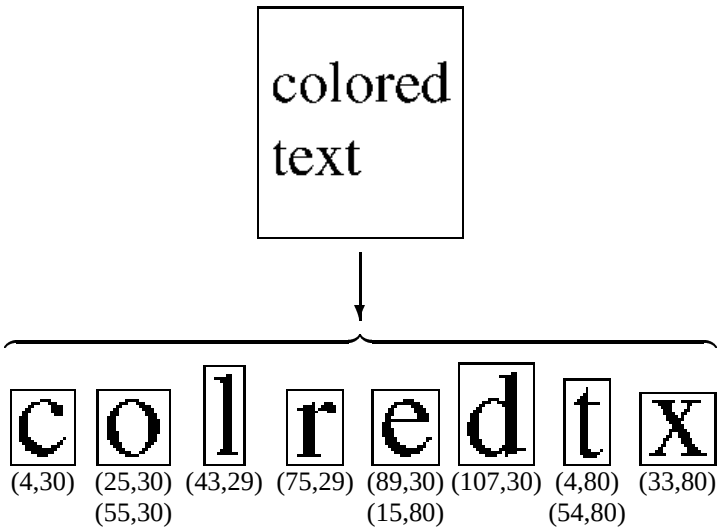


Figure 2: Illustration of token compression

DigiPaper shares the token dictionaries between multiple pages of a document; this greatly improves compression as most tokens occur on many pages. It also allows the foreground layer to share contone images between pages; this can help compression of documents where every page contains a logo, as is common in slide presentations.

Loss can be introduced during token compression, by allowing shapes in the original image that are slightly different to be called the "same" (placed in the same equivalence class). The decompressed image is not identical to the original image, but if the choice of these substitutions is made well, the differences are not noticeable. The shape comparison algorithm we use is described in [6], and is based on a modified Hausdorff distance [7].

2.3. Color tags

DigiPaper uses a novel extension of token compression to represent color images, where each instance of a token specifies not only a location on the page but also color information for drawing that token. The color information may be a single color value, or may specify an image to be masked. We refer to this as a *tagged token* representation, because each instance is tagged with color information. The tagged token representation extends the MRC model because a foreground layer and the corresponding selector layer are rep-

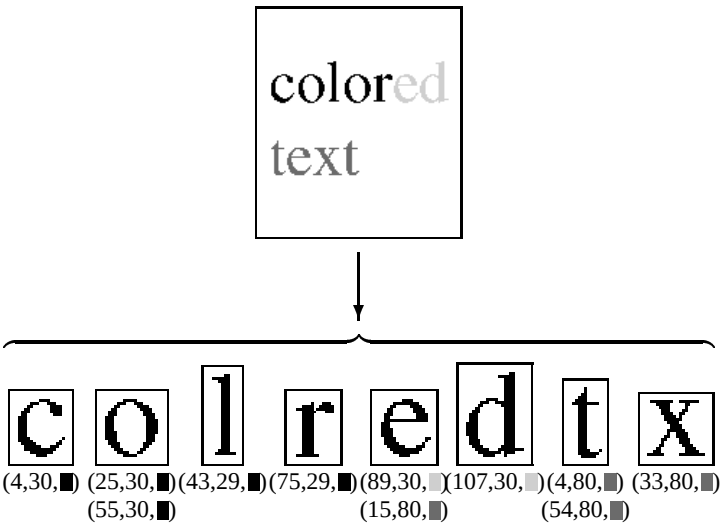


Figure 3: Illustration of color-tagged token compression

resented together, rather than as separate binary and color images. Each position is augmented with information about the color of that instance of that token (the *color tag*).

Figure 3 shows the effect of applying color-tagged token compression to the text in Figure 1. The token compression proceeds as in Subsection 2.2, identifying connected regions of the same color. The only difference is that each of the 11 regions now is identified by four values: its equivalence class, its x and y position, and its color. Note that the selector layer's image can be reconstructed by simply ignoring the color tags. In fact, since characters of the same color often occur near each other, separating the color tags from the rest of the data and run-length compressing them yields very good results.

The tagged token color representation used in DigiPaper allows colored text, such as that used in magazines and slide presentations, to be represented in nearly the same space as is required to represent simple black text. The tagged token approach is also well suited to the drawing models of PostScript and PDF, enabling efficient embedding of DigiPaper data in these formats. This provides a means of creating highly space efficient "print ready" files, that render quickly and always produce the same raster image regardless of the rendering environment (e.g., there are no fonts, layout or other differences).

3. SEGMENTATION OF PAGE IMAGES

Creating DigiPaper representations of scanned color documents is a difficult task: each page must be segmented into the foreground, background and selector layers. Getting the segmentation right improves the compression and visual appearance of the document. Our segmentation algorithm uses the tokenized representation to help with the segmentation. We rely on a number of attributes of text:

- The text shapes have a high contrast with the surrounding area
- Text shapes occur in groups (isolated letters are rare)
- The same shapes occur repeatedly
- Text shapes close to each other tend to have the same color
- The interior color of any given text shape tends to be smooth.

Our method uses these attributes to determine which parts of the scanned document image is text and which is contone background. It proceeds as follows.

1. Convert the page image to a binary image using adaptive thresholding. This uses the high contrast necessary for the text to be readable to segment it from the background. Of course, the thresholding algorithm also picks up a large number of non-text features from high-contrast regions of the background.
2. Find both black and white connected components in the thresholded image. These should include all the text characters, both light-on-dark or dark-on-light.
3. Determine the original color of each component; reject any components with large color variance as these are unlikely to be text.
4. Reject any components with no cohesive shape, as they are probably noise.
5. Find components that are close to other components of about the same size. These are initial guesses for text characters.
6. Feed these initial guesses into the token comparison engine, which groups together things that have the same, or nearly the same, shape.
7. Any component that matches another component is likely to be text, as shapes reoccur frequently within text and infrequently in photographs.
8. Additional components that are aligned with these text components, and are of about the same size and color, are also likely to be text.
9. Finally, select certain components that are close to, but considerably smaller than something marked as text. This picks up textual elements such as periods, commas, "i" dots, and so on.

Once this procedure has finished, the algorithm has a list of shapes that it believes are text. These shapes have already been grouped into equivalence classes. The algorithm then constructs the tokenized selector layer from the list and the equivalence classes. It also constructs the color-tagged foreground layer by annotating each shape on the list with its original color (recovered in step 3).

Figure 4 illustrates the result of applying this algorithm. It shows two regions of the same page. Figure 4(a) shows

the original regions. Figure 4(b) shows the result of adaptive thresholding (step 1). Figure 4(c) shows the selector layer extracted from that thresholded image. Finally, Figure 4(d) shows the results of decompressing the final DigiPaper image. The original image is 2340×3269 pixels, and is approximately 23 megabytes of uncompressed data. The DigiPaper file is approximately 100 kilobytes.

Another example is shown in Figure 5. The original image is 2410×2974 pixels, and is approximately 21.5 megabytes of uncompressed data. The DigiPaper file is approximately 260 kilobytes. This file is larger because the background is more complex, and thus takes more space to encode. In both of these examples, the background is represented at 100dpi.

Note that the pixels in the background and foreground layers that are not selected by the selector layer are "don't care"s, and so their values can be chosen arbitrarily. The values of these unselected "don't care" pixels are usually chosen in such a way as to optimize compression of the pixels that are selected. Methods for filling in these holes are described in [8, 9].

4. DECOMPRESSION

DigiPaper files can be decompressed extremely quickly: simple black and white DigiPaper files can be decompressed at over 30 pages per second, and complex color files can be decompressed at about 1 page per second.

Decompressing a black and white DigiPaper page reduces to decompressing one or more token dictionaries, decompressing the page's selector layer data, and then performing a series of "bit-blit" drawing operations, drawing the page's tokens into their final locations. The format for the compressed dictionary and selector data has been designed so as to make it easy to decompress, so this whole process takes very little time.

Decompressing a color DigiPaper page is not much different: first, the background is decompressed, scaled up to the selector's resolution, and drawn into the output buffer. Next, the selector data (plus any dictionaries it might use) is decompressed, along with the color tag data representing the foreground. Finally, the colored tokens are drawn on top of the background image. Because the images involved here are larger (usually 24 bits per pixel rather than 1 bit per pixel) than in the black and white case, this process is slower than decoding a black and white image. However, the tagged representation of the foreground layer eases the burden considerably: it is much easier to draw a series of colored shapes than it is to produce an entire image for the foreground layer and then mask it through the selector.

Furthermore, DigiPaper files can be printed easily: the imaging operations used to reconstruct the page images map well onto the primitive operations of most common page description languages, such as PostScript. This means that DigiPaper files can be converted to fast, compact PostScript. Again, the tagged representation of the foreground layer is crucial: it allows use of the font-definition and text-drawing PostScript primitives, which are likely to have been heavily optimized by the printer manufacturer.

5. CONCLUSIONS

The DigiPaper image file format uses token compression (embodied in JBIG2) and Mixed Raster Content in order to acheive very compact representation of document images. One of the problems with MRC is the segmentation of input images into the background, foreground, and selector layers. We have presented a method for performing this segmentation, which synergistically uses token compression to improve the quality of the segmentation. We have also presented the color tagging extension to MRC, which again makes synergistic use of token compression to represent the foreground in a manner that improves compression and ease of display. Color tagging also resolves one of the problems confronting MRC by allowing easy mapping of MRC images into printer-friendly formats.

6. REFERENCES

[1] R. de Queiroz. Processing JPEG-compressed images and documents. *IEEE Transactions on Image Processing*, 7(12):1661–1672, December 1999.

[2] International Telecommunication Union — Telecommunication Standardisation Sector. Mixed Raster Content (MRC). Recommendation T.44.

[3] L. McIntyre, S. Zilles, R. Buckley, D. Venable, G. Parsons, and J. Rafferty. File format for internet fax. IETF Request for Comments 2301, March 1998. `http://www.ietf.org/rfc/rfc2301.txt`.

[4] R. N. Ascher and G. Nagy. A means for achieving a high degree of compaction on scan-digitized printed text. *IEEE Transactions on Computers*, C-23:1174–1179, November 1974.

[5] P.G. Howard, F. Kossentini, B. Martins, S. Forchhammer, and W.J. Rucklidge. The emerging JBIG2 standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(7):838–848, November 1998.

[6] W.J. Rucklidge, D.P. Huttenlocher, and E.W. Jaquith. Method and apparatus for comparing symbols extracted from binary images of text using topology preserved dilated representations of the symbols. US patent 5835638, November 1998.

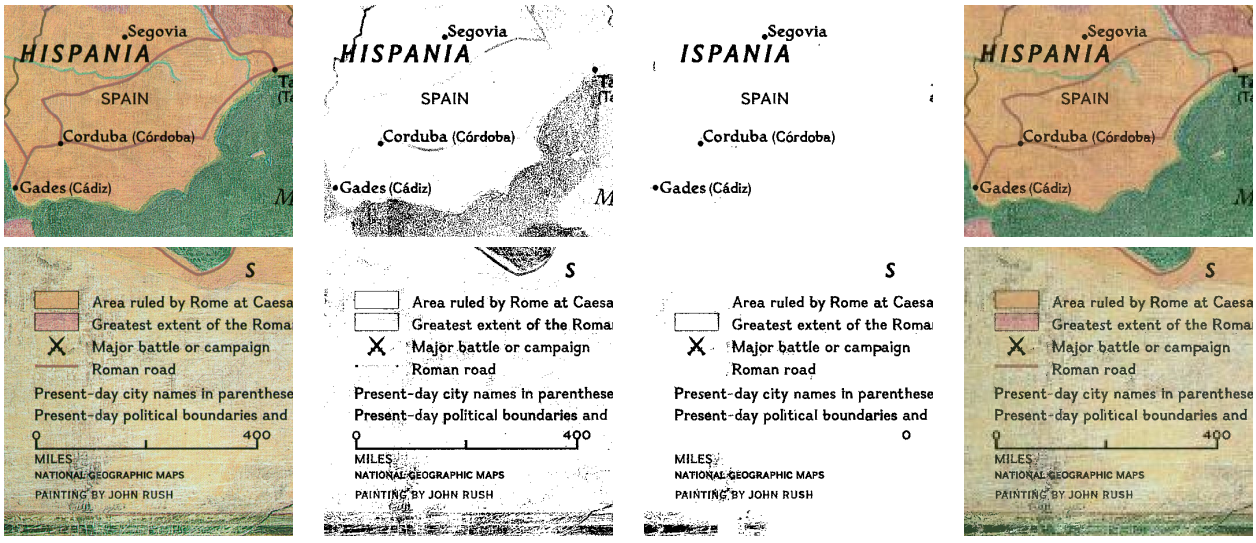
[7] D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, September 1993.

[8] L. Vincent. Morphological algorithms. Technical Report 91-12, Harvard Robotics Laboratory, 1991.

[9] Leon Bottou and Steven Pigeon. Lossy compression of partially masked still images. In J. Storer and M. Cohn, editors, *Proc. Data Compression Conference*, page 528, Snowbird, Utah, 1998.

(a) (b) (c) (d)

Figure 4: Segmentation example 1



(a) (b) (c) (d)

Figure 5: Segmentation example 2