Browsing through High Quality Document Images with DjVu

Patrick Haffner, Léon Bottou, Paul G. Howard,
Patrice Simard, Yoshua Bengio and Yann Le Cun

AT&T Labs-Research
100 Schultz Drive
Red Bank, NJ 07701-7033
{haffner,leonb,pgh,patrice,yoshua,yann}@research.att.com

Abstract

We present a new image compression technique called "DjVu " that is specifically geared towards the compression of high-resolution, high-quality images of scanned documents in color. With DjVu , any screen connected to the Internet can access and display images of scanned pages while faithfully reproducing the font, color, drawing, pictures, and paper texture. A typical magazine page in color at 300dpi can be compressed down to between 40 to 60 KB, approximately 5 to 10 times better than JPEG for a similar level of subjective quality. B&W documents are typically 15 to 30 KBytes at 300dpi, or 4 to 8 times better than CCITT-G4. A real-time, memory efficient version of the decoder was implemented, and is available as a plug-in for popular web browsers.

Keywords: digital libraries, image compression, image segmentation, arithmetic coding, wavelet coding, JBIG2

1 Introduction

As electronic storage, retrieval, and distribution of documents becomes faster and cheaper, libraries are becoming increasingly digital. Recent studies have shown that it is already less costly to store documents digitally than to provide for buildings and shelves to house them [1]. Unfortunately, the difficulty of converting existing documents to electronic form is a major obstacle to the development of digital libraries.

Existing documents are usually re-typed and converted to HTML or Adobe's PDF format, a tedious and expensive task sometimes facilitated by the use of Optical Character Recognition (OCR). While the accuracy of OCR systems has been steadily improving over the last decade, they are still far from being able to translate faithfully a scanned document into a computer-readable format without extensive manual correction.

Even if pictures and drawings are scanned and integrated into the web page, much of the visual aspect of the original document is likely to be lost. Visual details, including font irregularities, paper color, and paper texture, are particularly important for historical documents, and may also be crucial in documents with tables, mathematical or chemical formulae, and handwritten text.

A simple alternative would be to scan the original page and simply compress the image as a JPEG or GIF file. Unfortunately, those files tend to be quite large if one wants to preserve the readability of the text. Compressed with JPEG, a color image of a typical magazine page scanned at 100dpi (dots per inch) would be around 100 KBytes to 200 KBytes, and would be barely readable. The same page at 300dpi would be of acceptable quality, but would occupy around 500 KBytes. Even worse, not only would the decompressed image fill up the entire memory of an average PC, but only a small portion of it would be visible on the screen at once. A just-readable black and white page in GIF would be around 50 to 100 KBytes.

To summarize the current situation, it is clear that the complete digitization of the world's major library collections is only a matter of time. In this context, it seems paradoxical that there exist no universal standard for efficient storage, retrieval, and transmission of high-quality document images in color.

To make remote access to digital libraries a pleasant experience, pages must appear on the screen after only a few seconds delay. Assuming a 56 kilobits per second (kbps) connection, this means that the most relevant parts of the document (the text) must be compressed down to about 20 to 30 KBytes. With a progressive compression technique, the text would be transmitted and displayed first. Then the pictures, drawings, and backgrounds would be transmitted and displayed, improving the quality of the image as more bits arrive. The overall size of the file should be on the order of 50 to 100 KBytes to keep the overall transmission time and storage requirements within reasonable bounds.

Another peculiarity of document images, their large size, makes current image compression techniques inappropriate. A magazine-size page at 300 dots per inch is 3300 pixel high and 2500 pixel wide. Uncompressed, it occupies 25 MBytes of memory, more than what the average PC can handle. A practical document image viewer would need to keep the image in a compressed form in the memory of the machine, and only decompress on-demand the part of the image that is displayed on the screen.

The DjVu document image compression technique described in this paper is an answer to all the above problems. With DjVu , scanned pages at 300dpi in full color can be compressed down to 30 to 60 KBytes files from 25 MBytes originals with excellent quality. Black and white pages typically occupy 10 to 30 KBytes once compressed. This puts the size of high-quality scanned pages in the reach of an average HTML page (44 KBytes according to the latest statistics). DjVu pages are displayed within the browser window through a plug-in. The DjVu plug-in allows easy panning and zooming of very large images. This is made possible by an on-the-fly decompression method which allows images that would normally require 25 MBytes of RAM once decompressed to require only 2 MBytes of RAM.

The basic idea behind DjVu is to separate the text from the backgrounds and picture and to use different techniques to compress each of those components with the most appropriate method. Traditional methods are either designed to compress natural images with few edges (JPEG), or to compress black and white document images almost entirely composed of sharp edges (CCITT G3, G4, and JBIG-1). The DjVu technique improves on both, and combines the best of both approaches.

Section 2 reviews the current available compression and displaying technologies and states the requirements for document image compression. Section 3 describes the DjVu method of separately coding the text and drawings on one hand, and the pictures and backgrounds on the other hand. Section 4 turns this idea into an actual image compression format. It starts with a description of the method used by DjVu to encode the text and the drawings. This method is a variation of AT&T's proposal to the new JBIG2 fax standard [2]. It also includes a description of the IW44 wavelet-based compression method for pictures and background. The performance of both of these methods heavily relies on a new adaptive binary arithmetic coding technique called the Z-coder, also briefly described. Section 5 introduces the *plug-in* that allows to browse DjVu documents through standard applications such as Netscape Navigator or Microsoft Explorer with a DjVu plug-in Section 6 gives a very brief overview of the DjVu encoder. Comparative results on a wide variety of document types are given in Section 7.

2 Image-Based Digital Libraries

An obvious approach is to store and to transmit documents as images. When the document is remotely accessed, it is transmitted as an image. To achieve that, we need to devise a method for compressing document images that makes it possible to transfer a high-quality page over low-speed links (modem or ISDN) in a few seconds. OCR would only be used for indexing, with less stringent constraints on accuracy.

Several authors have proposed image-based approaches to digital libraries. The most notable example is the RightPages system [3]. The RightPages system was designed for document image transmission over a local area network. It was used for some years by the AT&T Bell Labs technical community, and distributed commercially for customized applications in several countries. The absence of a universal and open platform for networking and browsing, such as today's Internet, limited the dissemination of the RightPages system. Similar proposals have been made more recently [4, 5].

All of the above image-based approaches, and most commercially available document image management systems, are restricted to black and white (bi-level) images. This is adequate for technical and business documents, but insufficient for other types of documents such as magazines, catalogs, or historical documents. Many formats exist for coding bi-level document images, notably the CCITT G3 and G4 fax standards, the recent JBIG1 standard, and the upcoming JBIG2 standard. Using AT&T's proposals to the JBIG2 standard, images of a typical black and white page at 300dpi (dots per inch) can be transmitted over a modem link in a few seconds. Work on bi-level image compression standards is motivated by the fact that, until recently, document images were primarily destined to be printed on paper. Most low-cost printer technologies excel at printing bi-level images, but they must rely on dithering and half-toning to print grey-level or color images, thus reducing their effective resolution.

2.1 Displaying Color Documents

The low cost and availability of high-resolution color displays is causing more and more users to rely on their screen rather than on their printer to display document images. Even modern low-end PCs can display 1024x768 pixel images with 16 bits per pixel (5 bits per RGB component), while high-end PCs and workstations can display 1280x1024 at 24 bits per pixel.

Most documents displayed in bi-level mode are readable at 200dpi, but are not pleasant to read. At 300dpi the quality is quite acceptable in bi-level mode. Displaying an entire 8.5x11 letter-size page at such high resolution requires a screen resolution of 3300 pixels vertically and 2500 pixels horizontally, which is beyond traditional display technology. Fortunately, using color or gray levels when displaying document images at lower resolutions drastically improves readability and subjective quality. Most documents are readable when displayed at 100dpi on a color or grey-scale display. Only documents with particularly small fonts require 150dpi for effortless readability. At 100dpi, a typical page occupies 1100 pixels vertically, and 850 pixels horizontally. This is within the range of today's high-end displays. Low-end PC displays and high-end portable computer displays have enough pixels, but in the landscape mode rather that the desired portrait mode.

Figure 1: *Example of color document (*`hobby002`*)*

3 Document Image Compression with DjVu

As we stated earlier, the digital library experience cannot be complete without a way of transmitting and displaying document images in color. Traditional color image compression standards such as JPEG are inappropriate for document images. JPEG's use of local cosine transforms relies on the assumption that the high spatial frequency components in images can essentially be removed (or heavily quantized) without too much degradation of quality. While this assumption holds for most pictures of natural scenes, it does not for document images. Another technique is required to code accurately and efficiently the sharp edges of character images so as to maximize their clarity.

It is clear that different elements in the color image of a typical page have different perceptual characteristics. First, the text is usually highly contrasted from the background with sharp edges. The text must be rendered at high resolution, 300dpi in bi-level, or 100dpi in color, if reading the page is to be a pleasant experience. The second element in a document image is the pictures. Rendering pictures at 50dpi to 100dpi is typically sufficient for acceptable quality. The third element is the background color and paper texture. The background colors may not require more than 25dpi resolution.

Let us consider a document image scanned at 300dpi with 24 bits per pixel such the catalog page shown in Figure 1. The main idea of our document image compression technique is to generate and encode separately three images from which the original image can be reconstructed: the background image, the foreground image and the mask image. The first two are low-resolution color images, and the latter is a high-resolution bi-level image (300dpi). A pixel in the decoded image is constructed as follows: if the corresponding pixel in the mask image is 0, the output pixel takes the value of the corresponding pixel in the appropriately upsampled background image. If the mask pixel is 1, the pixel color is chosen as the color of the connected component (or taken from the foreground image). The background image (see for instance the lower right image of Figure 2) can be encoded with any suitable means, such as JPEG, or, in our case, a wavelet compression algorithm. The mask image (see for instance the upper left image of Figure 2) can be encoded using a bi-level compression algorithm.

Note that the foreground/background representation make it possible to re-use the existing state-of-the-art in image compression algorithms, and integrate them into a readily available coder.

The foreground/background representation is also instrumental in the MRC/T.44 recommendation proposal (Mixed Raster Content [6]). It is used in Xerox's PagisPro application, which demonstrates the XIFF format, an extension of TIFF which is used here to handle document representations that combine G4-encoded text with JPEG-encoded images

4 The DjVu Compression Format

In the previous section, we have seen that the Foreground/Background representation leads us to a great flexibility and modularity in the way we can use existing compression standards to code for the different planes.

In this section, we show how we designed the DjVu compression format, as a way to provide, as soon as possible, a virtual presence in a library. We make some necessary choices in the way we code the different levels in order to be able to provide a readily available, fully integrated compression scheme.

1. The text and drawings, also called the *Mask*, are represented by a single bitmap whose bits indicate whether the corresponding pixel in the document image has been classified as a foreground or a background pixel. This bitmap typically contains all the text and the high-contrast components of the drawings. It is coded at 300dpi using an algorithm which comes from our JBIG2 proposal (cf. Section 4.1).

2. The color of the text, namely the *Foreground*, contain a large number of neighboring pixels with almost identical colors. It can be considered as uniform for a given *mark* (i.e. a connected component of foreground pixels).

3. The *Background* is coded at 100dpi using the wavelet decomposition algorithm described in Section 4.2.

4.1 Coding the Bilevel Mask Using JB2

The bi-level image compression algorithm used by DjVu to encode the mask is dubbed JB2. It is a variation on AT&T's proposal to the upcoming JBIG2 fax standard. Although the JBIG1 [2] bi-level image compression algorithm works quite well, it has become clear over the past few years that there is a need to provide better compression capabilities for both lossless and lossy compression of arbitrary scanned images (containing both text and half-tone images) with scanning resolutions from 100 to 800 dots per inch. This need was the basis for JBIG2, which is being developped as a standard for bi-level document coding. The key to the compression method is a method for making use of the information in previously encountered characters without risking the introduction of character substitution errors that is inherent in the use of Optical Character Recognition (OCR) methods[7].

The basic ideas behind JB2 are as follows:

- The basic image is first segmented into individual marks (connected components of black pixels).

- The marks are clustered hierarchically based on similarity using an appropriate distance measure.

- Some marks are compressed and coded directly using a statistical model and arithmetic coding.

- Other marks are compressed and coded indirectly based on previously coded marks, also using a statistical model and arithmetic coding. The previously coded mark used to help in coding a given mark may have been coded directly or indirectly.

- The image is coded by specifying, for each mark, the identifying index of the mark and its position relative to that of the previous mark.

There are many ways to achieve the clustering and the conditional encoding of marks, the algorithm that we currently use is called "soft pattern matching" [8]. This algorithm does not yet attempt to optimize the clustering step.

The key novelty with JB2 coding is the solution to the problem of substitution errors in which an imperfectly scanned symbol (due to noise, irregularities in scanning, etc.) is improperly matched and treated as a totally different symbol. Typical examples of this type occur frequently in OCR representations of scanned documents where symbols like 'o' are often represented as 'c' when a complete loop is not obtained in the scanned document, or a 't' is changed to an 'l' when the upper cross in the 't' is not detected properly. By coding the bitmap of each mark, rather than simply sending the matched class index, the JB2 method is robust to small errors in the matching of the marks to class tokens. Furthermore, in the case when a good match is not found for the current mark, that mark becomes a token for a new class. This new token is then coded using JBIG1 with a fixed template of previous pixels around the current mark. By doing a small amount of preprocessing, such as elimination of very small marks that represent noise introduced during the scanning process, and smoothing of marks before compression, the JB2 method can be made highly robust to small distortions of the scanning process used to create the bi-level input image.

The JB2 method has proven itself to be about 20% more efficient that the JBIG1 standard for lossless compression of bi-level images. By running the algorithm in a controlled lossy mode (by preprocessing and decreasing the threshold for an acceptable match to an existing mark), the JB2 method provides compression ratios about 2 to 4 times that of the JBIG1 method for a wide range of documents with various combinations of text and continuous tone images. In lossy mode, JB2 is 4 to 8 times better than CCITT-G4 (which is lossless). It is also 4 to 8 times better than GIF.

4.2 Wavelet Compression of Background Images

Multi-resolution wavelet decomposition is one of the most efficient algorithms for coding color images [9, 10], and is the most likely candidate for future multi-level image compression standards. The image is first represented as a linear combination of locally supported wavelets. The image local smoothness ensures that the distribution of the wavelet coefficients is sharply concentrated around zero. High compression efficiency is achieved using a quantization and coding scheme that takes advantage of of this peaked distribution.

Because of the smoothness assumption, it is natural to use wavelet-based algorithms for encoding the image backgrounds. However, the requirements of the DjVu project set extreme constraints on the speed and memory requirements of the wavelet encoding scheme. The background image is typically a 100 dpi color image containing one to two million pixels. It may only represent a nearly uniform background color. It may also contain colorful pictures and illustrations which should be displayed incrementally while the DjVu data is coming.

Our wavelet compression algorithm uses an intermediate representation based on a very fast five stage lifting decomposition using Deslauriers-Dubuc interpolating wavelets with four analyzing moments and four vanishing moments [11]. Then the wavelet coefficients are progressively encoded using arithmetic coding (cf. Section 4.3) and a technique named "Hierarchical Set Difference" comparable to zero-trees [10] or set-partitioning [12] algorithms.

Finally we have developed a technique for coding the background image without wasting bits on background pixels that will be masked by foreground text. This simple and direct numerical method sets a large number of wavelet coefficients to zero, while transforming the remaining wavelet coefficients in order to preserve the visible pixels of the background only. The null coefficients do not use memory and are coded very efficiently by the arithmetic coder.

During decompression, the wavelet coefficients are represented in a compact sparse array which uses almost no memory for zero coefficients. Using this technique, we can represent the complete background using only a quarter of the memory required by the image pixels, and generate the fully decompressed image on-demand. This greatly reduces the memory requirements of the viewer.

4.3 Arithmetic Coding

Arithmetic coding [13, 14] is a well known algorithm for encoding a string of symbols with compression ratios that can reach the information theory limit. Its mechanism is to partition the interval $[0, 1)$ of the real line into subintervals whose lengths are proportional to the probabilities of the sequences of events they represent. After the subinterval corresponding to the actual sequence of data is known, the coder outputs enough bits to distinguish that subinterval from all others. If probabilities are known for the possible events at a given point in the sequence, an arithmetic coder will use almost exactly $-\log_2 p$ bits to code an event whose probability is p. In other words, the coder achieves *entropic compression*. We can think of the encoder and decoder as black boxes that use the probability information to produce and consume a bitstream.

Arithmetic coders unfortunately are computationally intensive. For each string element, a subroutine must provide the coder/decoder with a table containing estimated probabilities for the occurrence of each possible symbol at this point in the string. The coder/decoder itself must perform a table search and at least one multiplication.

4.3.1 Binary Arithmetic Coding

Binary adaptive arithmetic coders have been developed to overcome this drawback, as computations can be approximated using a small number of shifts and additions instead of a multiplication.

Moreover, Binary Adaptive Arithmetic Coders include an adaptive algorithm for estimating the symbol probabilities. This algorithm updates the integer variable along with the encoding and decoding operations. Complex probability models are easily represented by maintaining multiple indices representing the conditional probabilities of the symbols for each value of the contextual information considered by the model.

The QM-Coder used in the JBIG1 standard [2] and in the lossless mode of JPEG standard is an example of approximate binary arithmetic coder. Other such coders are the Q-Coder [15] and the ELS-Coder [16].

4.3.2 The Z-Coder and the ZP-Coder

We have implemented new approximate binary arithmetic coders, namely the Z-Coder and the ZP-Coder.

The Z-Coder was developed as a generalization of the Golomb run-length coder [17], and it has inherited its qualities of speed and simplicity [18]. Its internal representation leads to faster and more accurate implementations than either the Q-Coder or the QM-Coder. The probability adaptation in the Z-Coder also departs from the Q-Coder and QM-Coder algorithms in a way that simplifies the design of the coder tables.

The ZP-Coder is a variation on the Z-Coder with nearly exactly the same speed and performance characteristics. A rougher approximation in the optimal entropic Z-value costs less than half a percent penalty in code size.

We have compared the ZP-Coder with three other adaptive binary coders, the QM-Coder, the Q15-Coder (a variant of the Q-Coder that uses 15 bit registers instead of 12-bit), and the Augmented ELS-Coder, based on the ELS-Coder.

In the main test, various coders including the ZP-Coder have been incorporated into the JB2 compression system. The ZP-Coder did slightly worse than the ELS-Coder, about the same as the QM-Coder, and better than the Q15-Coder. The differences are all small, in the 1 to 2 percent range. We also performed two artificial tests. In a test of steady state behavior, coding a long sequence of random bits with fixed probabilities, the ZP-Coder performed about as well as the QM-Coder, better than the Q15-Coder, and much better than the ELS-Coder.

In a test of early adaptation, coding a long sequence of random bits with fixed probabilities but reinitializing the encoder index every 50 output bits, the ZP-Coder did better than the QM-Coder, which was better than the Q15-Coder, which in turn was better than the ELS-Coder.

The ZP-Coder's decoding speed is faster than that of the QM-Coder, which is in turn faster than the other two coders.

5 Browsing DjVu Documents

Satisfaction of the digital library user depends critically on the performance of the browsing tools. Much more time is spent viewing documents than formulating queries. As a consequence, browsers must provide very fast response, smooth zooming and scrolling abilities, realistic colors and sharp pictures.

These requirements put stringent requirements on the browsing software. The full resolution color image of a page requires about 25 megabytes of memory. We cannot store and process many of these in the browser without exceeding the memory limits of average desktop computers. However, we want to display such images seamlessly and allow users to drag them in real time on their screen.

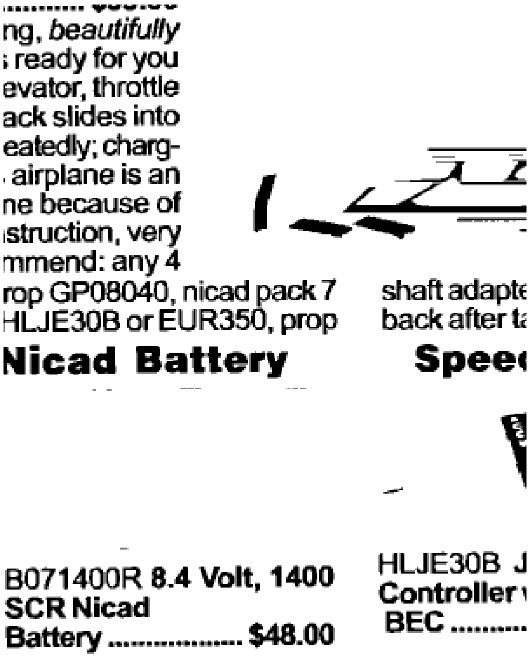
We developed a solution called "Multi-threaded two-stage decoding" consisting of the following:

- A first thread, known as the *decoding thread*, reads bytes on the internet connection and partially decodes the DjVu stream. This first stage of the decoding process asynchronously updates an intermediate representation of the page image. This representation is still highly compressed: our implementation requires less than 2 megabytes of main memory per page.

- A second thread, known as the *interactive thread*, handles user interaction and repaints the screen as needed. This thread uses the intermediate representation to reconstruct the pixels corresponding to the parts of the document that must be redrawn on the screen.

Despite the complexity related to thread management and synchronization, this organization provides many advantages. Since both threads work asynchronously, the browser is able to display images incrementally while data is coming in. Memory requirements are limited because the interactive thread computes image pixels only for regions smaller than the screen size. Scrolling operations are smooth because they involve just the second stage decoding of the few pixels uncovered by the scrolling operations.

We implemented these ideas in a plug-in for Netscape Navigator or Internet Explorer. Each page of a DjVu document is displayed by invoking its URL. Behind the scenes, the plug-in implements information caching and sharing. This design allows the digital library designer to set up a navigation interface using well known Web technologies like HTML or Java. Figure 2 shows how a document is displayed while it is downloaded through a 56K modem.



3.1sec/23k: the *mask* (text, 23K) is loaded.

4.8sec/35K: The background is still blurred.

9.4sec/67K: Loading is finished.

35K are necessary for this background image.

Figure 2: *Downloading through a 56K modem: progressive decompression of text first, followed by the background at increasing quality(detail of Figure 1)*

6 A Brief Overview of the DjVu Encoder

A description of all the algorithms that are used in the DjVu encoder is beyond the scope of this paper. One of the goals was to be at least as good as the state of the art for every layer of the DjVu format (mask, background and foreground layers). To achieve this, novel algorithms that address a wide variety of classical image processing problems have been proposed. *Multi-scale bicolor clustering* performs a foreground/background/mask separation that is more general than the standard text/image segmentation. With the *soft pattern matching* algorithm [8], our lossy JBIG2 compression ratios are on average twice those of lossless JBIG1, the best existing standard for bilevel images. The *multi-scale successive projections* algorithm [19] for wavelet decomposition of partially masked images significantly reduces the compressed file sizes and can handle arbitrarily complex masks with reasonable computational requirements. As seen before, the *Z-coder* [18] is faster than other approximate arithmetic coders, and yields, on average, better compression performance. All these algorithms have real-time implementations, and have been integrated into a standalone DjVu encoder.

7 Results and Comparisons with Other Methods.

We have selected seven images representing typical color documents. These images have been scanned at 300dpi and 24 bits/pixel from a variety of sources. Our compression scheme combines two main chunks (one wavelet compressed chunk for the background color, one JBIG2 chunk for the bitmap) whose combined size is reported in Figure 3.

Figure 3 gives a full comparison between JPEG and DjVu, with details from each image to assess the readability. Those details do not show the significant amount of graphics and texture that all these images contain. However, we give the percentage of bits spent on coding graphics and texture in each image, which ranges from 22 to 73. When compared to the original 300dpi raw image, DjVu achieves compression rates ranging from 324 to 579.

Compressing JPEG documents at 300 dpi with the lowest possible quality setting (20) yields images that are of comparable quality as with DjVu. As shown in the "JPEG, 300dpi" column, file sizes are 5 to 10 times larger than DjVu file sizes.

For the sake of comparison, we subsampled the document images to 100dpi (with local averaging) and applied JPEG compression adjusting the quality parameter to produce files sizes similar to those of DjVu. In the "JPEG, 100dpi" column, the fact that text is hardly readable is not due to the 100dpi subsampling, but to "ringing" artifacts inherent in low JPEG quality settings.

JPEG at 100dpi only. DjVu.

Figure 4: *Comparison of JPEG at 100dpi (left) with quality factor 30% and DjVu (right). The images are cropped from* **hobby002**. *The file sizes are 82K for JPEG and 67K for DjVu*

Figure 4 shows a more global comparison between DjVu and JPEG-100.

8 Conclusion

As digital libraries are increasingly becoming a fact of life, they will require a universal standard for efficient storage, retrieval and transmission of high-quality document images. The work described in this paper is a substantial step towards meeting this need, by proposing a highly efficient compression format (DjVu), together with a browser that enables fast internet access. With the same level of legibility (300 dots per inch), DjVu achieves compression rates 5 to 10 times higher than JPEG.

The DjVu plug-in is freely available for download at `http://djvu.research.att.com`. This site contains an experimental "Digital Library" with documents from various origins. The DjVu encoder is also available for research and evaluation purposes.

It is possible to further optimize the compression rate. A version of DjVu that encodes several pages together will be able to share JBIG2 shape dictionaries between pages. Problems such as foreground/background/mask separation or connected component filtering are being rephrased in terms of compression rate optimization.

The addition of *text layout analysis* and *optical character recognition* (OCR) will make it possible to index and edit text extracted from DJVU-encoded

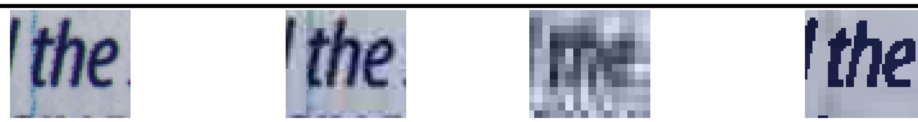
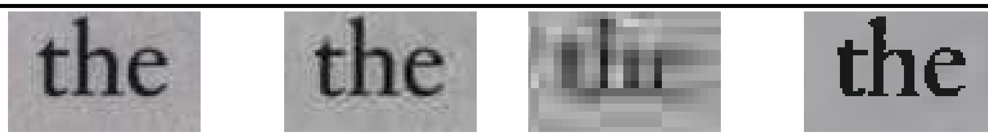
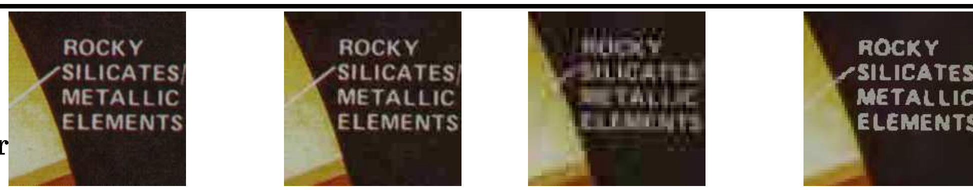
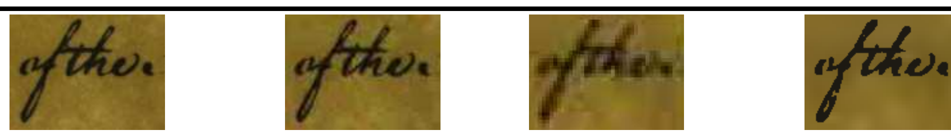
Image Description	Raw image detail	JPEG, 300dpi, quality 20	JPEG, 100dpi, size=DjVu	**DjVu compressed**	
Magazine Add % image= 56 `ads-freehand-300`	20640K	292K 70:1	50K 412:1	52K 396:1	
Brattain Notebook % image= 22 `brattain-0001`	9534K	116K 82:1	17K 560:1	19K 501:1	
Scientific Article % image= 46 `graham-001`	22013K	383K 57:1	41K 536:1	38K 579:1	
Newspaper Article % image= 50 `lrr-wpost-lrr-wpost-1`	12990K	250K 51:1	38K 341:1	40K 324:1	
Cross-Section of Jupiter % image= 73 `planets-jupiter`	24405K	284K 85:1	47K 519:1	47K 519:1	
XVIIIth Century book % image= 45 `cuisine-p006`	12128K	206K 58:1	35K 346:1	37K 327:1	
US First Amendment % image= 30 `usa-amend1`	31059K	388K 80:1	77K 403:1	73K 425:1	

Figure 3: *Compression results for seven selected images with 4 compression format.* `Raw` *applies no compression. JPEG-100 quality ranges from 5 to 50, the value yielding the compression rate which is the closest to DjVu is chosen. The "% image" value corresponds to the percentage of bits required to code for the background. Each column shows the same selected detail of the image. To make selection as objective as possible, when possible, the first occurrence of the word "the" was chosen. The two numbers under each image are the file size in kilobytes and the compression ratio (with respect to the raw file size).*

documents.

References

[1] M. Lesk. *Practical Digital Libraries: Books, Bytes and Bucks.* Morgan Kaufmann, 1997.

[2] JBIG. Progressive bi-level image compression. ITU recommendation T.82, ISO/IEC International Standard 11544, 1993.

[3] G. Story, L. O'Gorman, D. Fox, L. Shaper, and H Jagadish. The RightPages image-based electronic library for alerting and browsing. *IEEE Computer*, 25(9):17–26, 1992.

[4] T. Phelps and R. Wilensky. Towards active, extensible, networked documents: Multivalent architecture and applications. In *Proceedings of the 1st ACM International Conference on Digital Libraries*, pages 100–108, 1996.

[5] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Van Nostrand Reinhold, New York, 1994.

[6] MRC. Mixed rater content (mrc) mode. ITU draft recommendation T.44, ISO/IEC International Standard, 1997.

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

[8] P. G. Howard. Text image compression using soft pattern matching. *Computer Journal*, 1997. to appear.

[9] E. H. Adelson, E. Simoncelli, and R. Hingorani. Orthogonal pyramid transform for image coding. In *Proc. SPIE vol 845: Visual Communication and Image Processing II.*, pages 50–58, Cambridge, MA, October 1987.

[10] J. M. Shapiro. Embedded image coding using zerotrees of wavelets coefficients. *IEEE Transactions on Signal Processing*, 41:3445–3462, December 1993.

[11] Wim Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Journal of Applied Computing and Harmonic Analysis*, 3:186–200, 1996.

[12] Amir Said and William 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.

[13] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, June 1987.

[14] P. G. Howard and J. S. Vitter. Arithmetic coding for data compression. *Proceedings of the IEEE*, 82:857–865, 1994.

[15] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, and R. B. Arps. An overview of the basic principles of the q-coder adaptive arithmetic binary coder. *IBM Journal of Research and Development*, 32(6):717–726, November 1988.

[16] Wm. D Withers. A rapid entropy-coding algorithm. Technical report, Pegasus Imaging Corporation, 1996. Url ftp://www.pegasusimaging.com/pub/elscoder.pdf.

[17] S.W. Golomb. Run-length encodings. *IEEE Trans. Inform Theory*, IT-12:399–401, July 1966.

[18] L. Bottou, P. Howard, and Y. Bengio. The Z-coder adaptive binary coder. In *Proceedings of IEEE Data Compression Conference*, Snowbird, UT, 1998.

[19] L. Bottou and S. Pigeon. Lossy compression of partially masked still images. In *Proceedings of IEEE Data Compression Conference*, Snowbird, UT, 1998.