High Quality Document Image Compression with DjVu

Léon Bottou, Patrick Haffner, Paul G. Howard,
Patrice Simard, Yoshua Bengio and Yann LeCun
AT&T Labs
Lincroft, NJ
{leonb,haffner,pgh,patrice,yoshua,yann}@research.att.com

July 13, 1998

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. This enables fast transmission of document images over low-speed connections, while faithfully reproducing the visual aspect of the document, including color, fonts, pictures, and paper texture.

The DjVu compressor separates the text and drawings, which needs a high spatial resolution, from the pictures and backgrounds, which are smoother and can be coded at a lower spatial resolution. Then, several novel techniques are used to maximize the compression ratio: the bi-level foreground image is encoded with AT&T's proposal to the new JBIG2 fax standard, and a new wavelet-based compression method is used for the backgrounds and pictures. Both techniques use a new adaptive binary arithmetic coder called the Z-coder. 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. A real-time, memory efficient version of the decoder was implemented, and is available as a plug-in for popular web browsers.

1 Introduction

Much of the world's cultural, artistic, and scientific production is currently available only in paper form. While the World-Wide Web has been heralded as the platform for the world's universal library, progress has been slowed by the absence of a good way to convert paper documents to a digital form that can be easily distributed over the World-Wide Web.

Many of the world's major libraries are digitizing their collections. Recent studies have shown that it is already less costly to store documents digitally than to provide for buildings and shelves to house them in paper form [Lesk, 1997]. However, the distribution of these collections over the web has proved somewhat impractical due to large size of scanned document images.

Documents can be scanned and simply compressed using JPEG or GIF. Unfortunately, the resulting 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 also only a small portion of it would be visible on the screen at once. GIF is often used on the Web for distributing black and white document images, but it requires 50 to 100KB per page for decent quality (150dpi).

An alternative method is to convert existing documents to computer-friendly formats using Document Understanding techniques and Optical Character Recognition. However, while the accuracy of these systems has been steadily improving over the last decade, they are still far from being able to faithfully translate a scanned document 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.

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. The pleasure of reading an old book or browsing through a glossy catalog is not yet part of the digital realm. The fear of losing some critical information while scanning and translating a scientific, legal, or historical document deters scientists and scholars from using tools that would greatly increase their productivity.

The DjVu document image compression technique presented in this paper is an attempt to solve the problem of distributing high-quality, high-resolution images of scanned document in color over the Internet. With DjVu , scanned pages at 300dpi in full color can be compressed down to 40 to 60 KBytes files from 25 MBytes originals. Black and white pages in DjVu typically occupy 10 to 30 KBytes. This brings down the size of high-quality scanned pages to the same order of magnitude as an average HTML page on the Web today (44 KBytes according to the latest statistics). DjVu pages are displayed within Web browser windows through a plug-in.

The basic idea behind DjVu is that characters and line drawings need high spatial resolution but very little color resolution because they are composed of roughly uniform color areas with sharp edges. On the other hand, pictures and backgrounds are smoother and can be coded at lower spatial resolution with more bits per pixel. The DjVu compressor separates the text and drawings on one hand, from the pictures and backgrounds on the other hand. The text and drawings are compressed with a bi-level compression method at full resolution (typically 300dpi) , while the backgrounds and pictures are compressed with a wavelet-based technique at one-third the resolution (typically 100dpi).

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 [JBIG, 1993]. 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 ZP-coder, also briefly described. Section 6.1 introduces the *plug-in* that allows to browse DjVu documents through standard applications such as Netscape Navigator or Microsoft Explorer. Comparative results on a wide variety of document types are given in Section 7.

2 Image-Based Digital Libraries

The "image-based approach" to digital libraries is to store and to transmit documents as images. OCR can be used, but its role is limited to indexing, which has less stringent accuracy requirement than full transcription.

Several authors have proposed image-based approaches to digital libraries. The most notable example is the RightPages system [Story et al., 1992]. The RightPages system was designed for document image transmission over a local area network. It was used for some years by several research institutions including AT&T Bell Labs, and the University of San Francisco. It was also 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. Other proposals for image-based digital libraries have been made more recently [Phelps and Wilensky, 1996, Witten et al., 1994].

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. Using AT&T's proposals to the JBIG2 standard, images of a typical black and white page at 300dpi occupy 10 to 30KB, which 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.

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. 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 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.

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. A different 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. The text and line drawings for instance are usually highly contrasted from the background with sharp edges. Experiments have shown that the readability of most documents is not degraded by coding the text at 300dpi in bi-level mode. Displaying such images on color screens can be done by low-pass filtering and subsampling. Except for the tiniest font, the readability is preserved if the image is displayed with 100 screen pixels for each inch of paper. With pictures, on the other hand, 100dpi is typically sufficient for an acceptable quality level.

Let us consider a document image scanned at 300dpi with 24 bits per pixel such as the catalog page shown in Figure 1. The main idea of the technique presented in this paper 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 4) can be encoded with a method suitable for continuous-tone images. DjVu uses a progressive, wavelet-based compression algorithm called IW44 for this purpose. The mask image (the upper left image of Figure 4) can be encoded with a bi-level image compression algorithm. DjVu uses a method called JB2 for this purpose.

The foreground/background representation was proposed in the ITU MRC/T.44 recommendation (Mixed Raster Content [MRC, 1997]). The idea is used in Xerox's XIFF image format, which currently uses CCITT-G4 to code the mask layer, and JPEG to code the background and foreground layers. The format is used in Xerox's PagisPro desktop application for document scanning and indexing. Comparison between the various encoding scheme will be given in section Section 7.

4 The DjVu Compression Format

The elements that compose a DjVu encoded image file are:

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 called JB2, which is a variation of AT&T's proposal to the upcoming JBIG2 fax standard (cf. Section 4.1).
2. The *background* is coded at 100dpi using the wavelet-based compression algorithm called IW44 described in Section 4.2.
3. The *foreground* contains the color of the text and line drawings. It generally contains large areas of contiguous pixels with almost identical colors. This is because characters have generally a single color, and neighboring characters have generally identical colors. This image is coded as a low-resolution image using the same IW44 algorithm.

The last step of all compression algorithms is entropy coding. The efficiency of IW44 and JB2 heavily draws on their use of an efficient binary adaptive arithmetic coder called the *ZP-coder* [Bottou et al., 1998], which is described in Section 4.3. The ZP-coder compares favorably with existing approximate arithmetic coders both in terms of compression ratio and speed.

4.1 Coding the Bi-Level 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 standard for fax and bi-level image compression. Although the JBIG1 [JBIG, 1993] 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 JB2. The key to the JB2 compression algorithm is a method for making use of the information found in previously encountered characters without risking the introduction of character substitution errors that is inherent in the use of Optical Character Recognition (OCR) [Ascher and Nagy, 1974].

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" [Howard, 1997].

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.

4.2 Wavelet Compression of Background Images

Multi-resolution wavelet decomposition is one of the most efficient transforms for coding color images [Adelson et al., 1987, Shapiro, 1993], and is the preferred candidate for many ongoing standardizations efforts in color and gray-scale image compression. 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 sparse and sharply concentrated around zero. High compression efficiency is achieved using a quantization and coding scheme that takes advantage of this peaked distribution.

Because of the smoothness assumption, it is natural to use wavelet-based algorithms for encoding the image backgrounds. The digital library applications for which DjVu is designed puts extreme requirements on the technique used. The viewer (decoder) must be able to decode and display a page in at most 2 seconds to decode a page on a low-end PC. It needs to be progressive so that the image quality improves as more bits arrive. It needs to do real-time decompression from a compact data structure so as not to exceed the RAM capacity of an average PC with the fully decoded image. The background image is typically a 100dpi color image containing one to two million pixels. It may contain a nearly uniform background color, or it may contain colorful pictures and illustrations that 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 [Sweldens, 1996]. Then the wavelet coefficients are progressively encoded using arithmetic coding (cf. Section 4.3) and a technique named "Hierarchical Set Difference" that bears some resemblance to zero-trees [Shapiro, 1993] or set-partitioning [Said and Pearlman, 1996] algorithms.

During decompression, the wavelet coefficients are represented in a compact sparse array that uses almost no memory for zero coefficients. Using this technique, the complete background image can be represented using only a quarter of the memory required by the image pixels. The piece of the image that is deisplayed on the screen can generated on-the-fly from this sparse array at the required zoom factor. This greatly reduces the memory requirements of the viewer.

If the foreground color is also represented as a low-resolution image, the same algorithm is used to code this image. In the rest of the paper, this new wavelet compression format will be referred to as IW44 (as we use 4x4 wavelets).

4.3 Arithmetic Coding

Arithmetic coding [Witten et al., 1987, Howard and Vitter, 1994] is a well-known method for encoding a string of symbols with compression ratios that can approach the information theory limit. Its mechanism is to interpret the bitstream to be encoded (or decoded) as a number between 0 and 1 and 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. The computation 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 [JBIG, 1993] and in the lossless mode of JPEG standard is an example of approximate binary arithmetic coder. Other such coders are the Q-Coder [Pennebaker et al., 1988] and the ELS-Coder [Withers, 1996].

4.3.2 The Z-Coder and the ZP-Coder

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

The Z-Coder was developed as a generalization of the Golomb run-length coder [Golomb, 1966], and has inherited its qualities of speed and simplicity [Bottou et al., 1998]. 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.

Golomb coders essentially code the length of contiguous strings of identical symbols as a binary number. A Golomb coder is defined by the parameter M, which is the number of bits in that binary number. Large values of M are preferrable when long strings of repeating symbols are likely, while small values of M are preferrable when those long strings are unlikely. The Z-coder generalizes Golomb coders by automatically adapting M to its optimal value, including non-integer values [Bottou et al., 1998].

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. However, the differences in performance between the coders were all small.

5 Optimization of the Compression Rate

DjVu achieves high compression ratios by using new compression algorithms for the mask layer as well as for the background and foreground layers. Here are some of the novel techniques used by DjVu: the *soft pattern matching* algorithm (cf. Section 5.2), used in the JB2 bi-level image compression algorithm for the mask layer; the sparse set representation of wavelet coefficients used by the IW44 wavelet-based encoder; a *multi-scale successive projections* algorithm (cf. Section 5.3), which avoids spending bits to code the parts of the background image that are covered by foreground objects.

The previous section described the compression format but did not describe the compression algorithms. This section describes the new algorithms used in DjVu such as the foreground/background/mask separation algorithm, the soft pattern matching algorithm, and the wavelet masking technique.

5.1 The Foreground/Background/Mask Separation

The first step of the DjVu compressor is the separation of the text and line drawings from the pictures and backgrounds. Some algorithms proposed in the literature address this problem in a top-down fashion, segmenting the document into visually distinct regions, and classifying those regions into text, line drawings, images or halftones. This approach works well for bi-level documents [Witten et al., 1994] or color documents with uniform backgrounds where the images are clearly separated from the text.

Other algorithms, following a bottom-up scheme, try to find text based on local features. Typically, they take advantage of the distinctive characteristics of the text that make it stand out from the background. For instance, in [Wu et al., 1997], the *textual* foreground is identified by its specific frequency and orientations, and its spatial cohesion (characters of the same string have similar height and orientation).

This section describes a multi-scale bicolor clustering algorithm that does not view the task as "segmenting" or "finding" text, but rather as a much more general foreground/background classification. The image is partitioned into square blocks of pixels. A clustering algorithm finds the two dominant colors within each block. Then, a relaxation algorithm ensures that neighboring blocks assign similar colors to the foreground and the background. After this phase, each pixel is assigned to the foreground if its color is closer to the foreground cluster prototype than to the background cluster prototype. A subsequent phase cleans up and filters foreground components using a variety of criteria.

Despite its simplicity, this algorithm finds text (and many other objects) in complex color images with a very good accuracy. This algorithm is our first candidate to be used as the foreground/background separator in the DjVu encoder. The rest of the section gives a more detailed description of the algorithm.

5.1.1 Color Clustering with the K-means algorithm

Consider the color histogram of a bicolor document image (i.e. an image with a quasi-uniform background and foreground color). Both the foreground and background color is represented by peaks in the histogram. There may be a small ridge between the peaks representing the intermediate colors of the anti-aliased pixels located near the character boundaries.

Extracting the foreground and background color is then easily achieved by running a clustering algorithm on the colors of all image pixels. The following algorithm, based on the well known k-Means algorithm [MacQueen, 1967] achieves this task.

- Initialize the background color to white and the foreground color to black.
- Iterate over the pixels of the image and decide whether this is a foreground pixel or a background pixel by comparing the distances between the pixel color and the current foreground and background colors.
- Update the current foreground (resp. background) color by computing the average color of all the foreground (resp. background) pixels.
- Repeat steps 2 and 3 until convergence of the foreground and background colors. Although this convergence occurs very quickly, a careful use of stochastic approximations can make this algorithm even faster [Bottou and Bengio, 1995].

We have initialized the above algorithm using white as the background color and black as the foreground color. This initialization ensures that the color histogram peak with the highest (resp. lowest) luminosity will be returned as the background (resp. foreground) color.

There are of course more reliable heuristics for choosing which peak represents the foreground color. A simple and reliable method selects the highest peak as the background color. The surface of a typical bicolor document mostly consists of the background color. More sophisticated methods rely on the topological properties of the two sets of pixels identified by the algorithm (e.g. number of connected components, average number of holes in the connected components). These methods however are beyond the scope of this paper.

5.1.2 Block Bicolor Clustering

In reality, typical document images are seldom limited to two colors. The document design and the lighting conditions induce changes in both the background and foreground colors over the image regions.

An obvious extension of the above algorithm consists in dividing the document image using a regular grid delimiting small rectangular blocks of pixels. Running the clustering algorithm within each block produces a pair of colors for each block. We can therefore build two low-resolution images whose pixels correspond to the cells of our grid. The pixels of the first image (resp. the second image) are painted with the foreground (resp. background) color of the corresponding block.

This block bicolor clustering algorithm is affected by several problems involving the choice of a block size and the selection of which peak of each block color histogram represents the background (resp. foreground) color.

- Blocks should be small enough to capture the foreground color change, for instance, of a red word in a line of otherwise black text. The size of the smallest characters therefore is a maximal size for our blocks.
- Such a small block size however increases the number of blocks located entirely outside the text area. Such blocks contain only background pixels. Blocks may also be located entirely inside the ink of a big character. Such blocks contain only foreground pixels. The clustering algorithm fails in both cases to determine a pair of well contrasted foreground and background colors.
- A small block size also ruins the reliable heuristic algorithms for deciding which color histogram peak represents the actual background (resp. foreground) color. We must revert to choosing the most luminous color as the background color. This simple choice produces the foreground and background with quite continuous tones.

5.1.3 Multi-Scale Bicolor Clustering

Instead of considering a single block size, we will now consider several grids of increasing resolution. Each successive grid delimits blocks whose size is a fraction of the size of the blocks of the previous grid.

Applying the bicolor clustering algorithm on the blocks of the first grid (the grid with the largest block size) we obtain a foreground and background color for each block in this grid. The blocks of the next grids are then processed with a slightly modified color clustering algorithm.

This modification introduces a bias in the clustering algorithm by attracting the clusters towards the foreground and background colors found for the corresponding block of the previous grid. The following algorithm is a modification of the k-means algorithm described above for that task:

- Locate the corresponding block of the previous grid (i.e. the block of the previous grid containing the center of the current block). The background and foreground colors of this larger block can be used for initializing the current background and foreground colors.
- Iterate over the pixels of the image and decide whether this is a foreground pixel or a background pixel by comparing the distances between the pixel color and the current foreground and background colors.
- Update the current foreground (resp. background) color by computing a weighted average of (a) the average colors of all foreground (resp. background) pixels, and (b) the foreground (resp. background) color found for the corresponding block of the previous grid.
- Repeat steps 2 and 3 until convergence of the foreground and background colors.

Overlooking the effects of the algorithm initialization, this modified algorithm returns the peaks of a color histogram obtained by computing the weighted average of (a) the color histogram of the current block, and (b) a color histogram concentrated on the foreground and background colors found for the corresponding block of the previous grid.

This operation alleviates the small block size problem discussed in the previous section. If the current block contains only background pixels, for instance, a small proportion of the foreground and background colors of the larger block will play a significant role. The resulting background color will be the average color of the pixels of the block. The resulting foreground color will be the foreground color of the larger block. If however the current block contains pixels representing two nicely contrasted colors, this small proportion of the colors identified for the larger block will have a negligible impact on the resulting clusters.

We have obtained good results on a wide range of document images by mixing 80% of the actual color of the foreground (resp. background) block pixels and 20% of the foreground (resp. background) color of the larger block.

5.1.4 Filtering and Inverting the Foreground

Our foreground/background/mask separation algorithm attempts to retain as much information as possible about the original image while quantizing the colors on two levels only: background for the light colors and foreground for the dark colors. This is not exactly our objective, where the foreground is the part of the image where a high resolution is necessary for visual understanding.

For example, the separation aglorithm may erroneously put highly-contrasted pieces of pictures in the foreground. A variety of filters are applied to the resulting foreground image so as to eliminate the most obvious mistakes. Those filters are based on various heuristics such as the color content of the mark, its size, or its position relative to other marks. Occasionally, the text segmented by the separation algorithm will appear inverted in the foreground image (as holes of a large connected component). Another filter detects those occurences and corrects them.

5.2 Soft Pattern Matching

The JBIG2 standard (cf. Section 4.1) does not specify how the marks are clustered and filtered to achieve an optimal compression rate. After a brief description of related algorithms, we describe the 'Soft Pattern Matching' algorithm [Howard, 1997] that we use in the JBIG2 encoder, which comes in two flavors: lossless and lossy.

5.2.1 Lossy compression based on pattern matching and substitution

Some research has been done on lossy techniques for compressing text images using pattern matching and substitution [Ascher and Nagy, 1974, Mohiuddin et al., 1984, Holt and Xydeas, 1986, Witten et al., 1992, Witten et al., 1994]. Conceptually, the idea is to segment the image into individual characters and to partition the set of characters into equivalence classes, each class ideally containing all the occurrences of a single letter, digit, or punctuation symbol, and nothing else. Coding the image consists in coding the bitmap of a representative instance of each equivalence class, and coding the position of each character instance along with the index of its equivalence class. The equivalence classes and representatives can, of course, be constructed adaptively. The following is a typical practical organization of the encoder for such a system. The various pattern matching and substitution (PM&S) methods differ primarily in the matching technique employed.

Pattern Matching and Substitution Algorithm

```
Segment image into marks
for each mark do
  Find "acceptable match", if any
  if there is a match
    Code index of matching mark
  else
    Code bitmap directly
    Add new mark to library
  end if
  Code mark location as offset
end for
```

A significant drawback to PM&S methods is the likelihood of substitution errors. To obtain good compression ratios, the pattern matcher must be aggressive in declaring matches, but as the matching criteria become less stringent, mismatches inevitably occur. A human reader can often detect and correct such errors (consciously or unconsciously), but sometimes this is not possible. Part numbers in images inherently have no context, and the figures '5' and '6' can be problematical even when context is available. We do not want to include typeface-specific or even alphabet-specific information in a general-purpose coder, but in most alphabets there are *confusion pairs* – different alphabet symbols that are similar according to most easily implemented matching criteria.

5.2.2 Lossless Soft Pattern Matching

The system described in this paper combines features of both the non-progressive JBIG1 encoder and the PM&S methods. The image is segmented into connected components of black pixels, which we call *marks*, following [Witten et al., 1994]. Each pixel of each mark is coded using a JBIG-like coder. We use pattern matching, but instead of directly substituting the matched character as in the PM&S methods, we simply use its bitmap to improve the context used by the coder, a technique we call *soft pattern matching*.

Our system can be used in lossless or lossy mode. Lossless mode is simpler, and is very similar to that of Mohiuddin, Rissanen, and Arps [Mohiuddin et al., 1984]. It is illustrated in the following procedure, which is similar to the generic lossy PM&S method shown above; the only difference (shown in italics) is that lossy direct substitution of the matched character is replaced by a lossless encoding that uses the matched character in the coding context.

Soft pattern matching algorithm

```
Segment image into marks
for each mark do
  Find "acceptable match", if any
  if there is a match
    Code index of matching mark
    Code bitmap using matching mark
    Conditionally add new mark to library
  else
    Code bitmap directly
    Add new mark to library
  end if
  Code mark location as offset
end for
```

5.2.3 Lossy Soft Pattern Matching

Lossy mode follows the same outline, but requires the implementer to exercise some judgment in deciding on acceptable levels of distortion. The idea is to allow some pixels to be changed when doing so reduces the bit count without introducing any "significant" changes to the image.

There are several modifications that can be made to the method described in Section 5.2.2 to increase the compression ratio by making the compression lossy. All the variations involve preprocessing the original image followed by lossless coding of the modified image.

The first three modifications are standard techniques that enhance the image while reducing the bit count as a side effect, while the other two involve preprocessing the marks in a way designed specifically to reduce the bit count while not degrading their appearance appreciably.

We can obtain some improvement in compression ratio by quantizing the offsets. Unfortunately, the increase in compression ratio is small, on the order of one percent, and the restored images do not look as good, so this is not a useful procedure.

We get some improvement by eliminating very small marks that represent noise on the page. In our implementation we ignore only marks consisting of single black pixels surrounded by eight white pixels.

We get more improvement by smoothing each mark before compressing it and thus before entering it into the list of potential matching marks. One simple smoothing that can be done is to remove single protruding pixels (white or black) along mark edges. Smoothing has the effect of standardizing local edge shapes, which improves prediction accuracy and increases the number of matching pixels between the current mark and potential matching marks. The increase in compression ratio is about ten percent.

We can get a considerable improvement in compression by taking advantage of the best matching mark described in Section 5.2.2 and applying *selective pixel reversal.* For every pixel, we obtain a code length less than one bit if we can predict its color well (i.e., with probability greater than 1/2), and greater than one bit if we cannot. One mechanism to reduce the code length is to reverse the color of poorly predicted bits before coding them. To avoid seriously distorting the mark as a whole, we should limit the reversal to places where it will be imperceptible. We can generally reverse isolated poorly predicted pixels, and in fact reversing groups of two isolated pixels causes only a minor perceived difference. In our implementation we use 4-connectivity to decide whether a pixel or group of pixels is isolated. Unfortunately, the exact color probabilities are not readily available to the encoder during the preprocessing phase, before the actual coding begins. First, the Z-Coder changes the probabilities adaptively during coding. Second, there is a cascade effect: changing the color of one pixel changes the context (and thus the color probability) of up to 4 other pixels in the current mark. The pixel's color prediction is approximated by the color of the aligned pixel in the best matching mark. After reversing the color of the isolated mismatched pixels, we code the mark as in the lossless case.

Finally, we can also apply selective pixel reversal to improve the compression of marks with no acceptable matching mark. In this case there is no convenient corresponding pixel to serve as an approximation for the prediction of the current pixel. Instead, we choose as candidates (poorly predicted pixels) those pixels that differ from both of their two nearest causal neighbors, the one above and the one to the left. Other pixels are not necessarily well predicted; we just do not know that they are predicted badly enough that flipping them would be likely to reduce the bit count. Again, candidates are actually reversed only if they are isolated. After reversing the color of the isolated poorly predicted pixels, we code the mark as in the lossless case, using the 10-pixel template of Figure 2.

5.3 Wavelet Compression of Partially Masked Images

Section 4.2 briefly introduced our wavelet decomposition algorithm and described how it enables progressive coding and fast, memory-efficient viewing through a browser. In this section, the wavelet coder is modified to take advantage of the fact that, in our DjVu format, background (resp. foreground) pixels can be masked by foreground (resp. background) pixels. When we are interested in coding the background, background pixels are said to be *visible* whereas foreground pixels are *masked.*

The visible pixels (i.e. pixels that are not masked) are never affected by the coefficients of wavelets whose support is *entirely located below the mask.* Therefore, a simple idea for solving our problem consists of either (a) skipping these coefficients while coding, or (b) setting them to zero, which is the most code-efficient value. The first solution saves a few bits, but requires knowing the mask when decoding the compressed image file. The second solution does not suffer from this constraint: the compressed image file can be decoded with the usual algorithms, regardless of the mask.

This approach is not practical when the mask (i.e. the set of masked pixels) is composed of small connected components. Most of the information about masked background pixels is carried by wavelets whose support is *partially masked only.* Canceling the coefficient of a partially masked wavelet obviously changes pixels located outside the mask. The coefficient of other wavelets must be adjusted to compensate for this effect. We have found empirically that good results are achieved by canceling wavelet coefficients as soon as half the energy of the wavelet is located below the mask.

The solutions of our problem belong to the intersection of the following sets of images:

- $\mathcal{P}$: images that match the input image on the visible pixels.
- $\mathcal{Q}$: images whose masked wavelet coefficients are zero.

The initial image x_0 is already an element of set $\mathcal{P}$. As shown in Figure 3, we successively project this image on sets $\mathcal{Q}$ and $\mathcal{P}$.

This sequence is known [Youla and Webb, 1982] to converge toward a point in the intersection of convex sets $\mathcal{P}$ and $\mathcal{Q}$, provided that this intersection is non-empty. The simplest version of the *Successive Projections Algorithm* therefore consists of the following steps:

i) Initialize a buffer with the pixel values of the initial image.

ii) Perform the wavelet decomposition.
iii) Set the coefficients of all masked wavelets to zero.
iv) Perform the image reconstruction.
v) Reset all visible pixels to their value in the initial image.
vi) Loop to step *(ii)* until convergence is reached.

Convergence is easily monitored by measuring the distance between the visible pixels of the initial image and the corresponding pixels of the image reconstructed in step *(iv)*.

It is possible to prove that the successive projection method reaches a solution with predetermined accuracy after a number of iterations proportional to the logarithm of the number of masked pixels. Moreover, two modification to the successive projection method result in additional speedups:

- Instead of being applied to all the wavelets at once, it can be applied to each scale separately, proceeding from the fine scales to the coarser ones. This *multi-scale successive projection algorithm* has been found to run one order of magnitude faster on realistic images.
- Another speedup can be obtained by applying an *overshooting* technique widely applied for successive projections onto convex sets [Youla and Webb, 1982, Sezan and Stark, 1982].

In summary, we have developed a technique for coding the background (or the foreground) 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.

This section introduced three algorithms designed to optimize the compression rate. The next section explores more qualitative ways to improve the user experience when accessing DjVu images.

6 Implementation

This section reviews additional features attached to the DjVu format intended to satisfy requirements from potentials users and content publishers. A key requirement is to be able to browse DjVu documents through standard applications such as Netscape Communicator or Microsoft Explorer (cf. Section 6.1).

The possibility to retrieve documents using keywords or phrases must be as extensive as possible. Ideally, we would like scanned documents to be as easy to access as HTML documents. Section 6.2 describes our research effort in Optical Character Recognition (OCR) for that purpose.

6.1 Browsing DjVu Documents

The digital library user experience 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, good color reproduction and sharp pictures.

These requirements put stringent constraints on the browsing software. The full resolution color image of a page requires about 25 MBytes 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 pan around in real time.

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 MBytes 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 and Internet Explorer. Unlike with many document browsers, each page of a DjVu document is associated with a single 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, JavaScript, or Java. This provides more flexibility than other document browsing systems where whole documents are treated as a single entity, and the viewer handles the navigation between the pages. Figure 4 shows how a document is displayed while it is downloaded through a 56K modem.

6.2 Indexing

Building an index for a collection of textual images requires performing Document Layout Analysis and Optical Character Recognition (OCR). However, since in our system the OCR result is used solely for indexing, and has no effect on the appearance of the document on the user's screen, the requirements on the error rate are much less stringent than if the OCR were used to produce an ASCII transcription. Before the OCR can be performed, the document image must be analyzed and broken down into paragraphs, lines, and characters.

First, a black and white image is created using the *foreground/background/mask separation* algorithms described in Section 5.1 Then a connected component analysis is performed on this image. The result is a list of blobs (connected black pixels) associated with their coordinates and bounding box sizes.

From this list, large blobs that are obviously not characters are eliminated from the image. Then, words and lines are found by iteratively grouping together connected components that are close to each other and approximately horizontally aligned. The procedure iteratively builds a tree data structure where the leaves contain single connected components, and where the nodes contain clusters of components grouped into words, lines, and paragraphs.

The lines are then segmented into individual characters using heuristic image-processing techniques. Multiple segmentation hypotheses are generated and represented as a Directed Acyclic Graph (DAG). Each candidate segment is sent to a *Convolutional Neural Network* recognizer[LeCun et al., 1995]. The recognizer produces one label with an associated likelihood. The range of possible interpretations for all the possible segmentations is again represented by a DAG. Each path in the DAG corresponds to one interpretation of one particular segmentation. The accumulated penalty for an interpretation is the sum of the penalties along the corresponding path. A shortest-path algorithm, such as the Viterbi algorithm, can be used to extract the best interpretations. The interpretation DAG can be combined with a language model (represented by a directed graph) to extract legal interpretations. More details on this DAG-based OCR system can be found in [LeCun et al., 1997].

7 Results and Comparisons with Other Methods.

This section provides experimental compression results on a variety of image. First, comparisons with JPEG or with a wavelet-only coder are provided on 7 images, either at the same level of readability or the same file size. Then, we attempt to analyze the improvements in compression quality and performance brought by each one of the main ideas and algorithms presented in this paper. It is possible to distinguish three major innovations in DjVu, the foreground/background/mask separation, the use of JB2 for the mask, and the use of the IW44 wavelet encoder for the foreground and background images (note that we use for these experiments a version of DjVu where the foreground is coded as an IW44 image rather than a color attached to each connected component).

7.1 Comparisons with JPEG at the Same Level of Readability or the Same File Size

When it comes to lossy compression of color document images, the most widely used standard is JPEG. This is the reason why we used it as our comparisons baseline.

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. Figure 5 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 22:1 to 73:1. When compared to the original 300dpi raw image, DjVu achieves compression rates ranging from 324:1 to 579:1. Details about the DjVu encoder are reported in the next 2 subsections.

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. The JPEG encoder we used is CJPEG version 6a, available on SGI workstations. Slightly better compression ratio could be obtained with more recent, but proprietary, JPEG encoders. For instance, if we use the "jpeg-wizard" provided by Pegasus Imaging (http://www.pegasusimaging.com), the compressed JPEG 300dpi documents become, on average, 10% smaller for exactly the same quality.

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. It may be argued that JPEG was designed for compression rates of 40:1 rather than 400:1, and the fact that we had to downsample the image to achieve a very high compression rate with JPEG concurs with that remark. In fact, research on wavelet compression algorithms is in part motivated by this need to have a working standard for higher compression rates. The performance of IW44 is typical of the current state-of-the-art in wavelet compression. Unlike JPEG, IW44 was able to compress the *non-subsampled* image to a file size which is comparable to DjVu. The "IW44, 300dpi" column shows that IW44 compressed image quality is better than JPEG, but still far from DjVu. This proves that to represent the textual components of a color document as a bi-level image is critical for DjVu performance.

Figure 6 shows a comparison between DjVu and "JPEG, 100dpi" on a larger segment of an image. It is interesting to note that the rendering of the photographs is also better with DjVu that with JPEG, even though DjVu uses the same 100dpi for the background image. Beside the fact that IW44 is better than JPEG, DjVu can allocate more bits than JPEG to code these photographs (as no bits are wasted on coding the textual part of the document with a smooth decomposition).

7.2 Subjective Evaluation of the Foreground/Background/Mask Separation

The foreground/background/mask separation algorithm produces 3 *sub-images*: (i) the 300dpi bi-level mask image (ii) the 100dpi background color image and (iii) the 25dpi foreground color image. We can combine these *raw sub-images* into a *raw multi-layer image* However, there is no readily available measure of quality for this raw multi-layer image. A distortion measure that would weight equally each pixel would miss the whole purpose of the separation of the pixels into two categories. In fact, an evaluation of the foreground/background separation algorithm is, by essence, subjective. Two different persons may disagree about what shall be part of the foreground and what shall be part of the background.

The proof of concept is obtained by running the complete DjVu algorithm: Figure 6 compares DjVu (which combines the foreground/background separation with the IW44 wavelet encoder) with the IW44 encoder alone (column "IW44, 300dpi"). It clearly demonstrates the benefits of such a separation on the textual components of the image.

Moreover, the experimental digital library which is available online at `http://djvu.research.att.com` shows the performance of the algorithm on a large variety of document types (it is important to stress that no hand-correction was applied to any of these images). From the feedback of users (extremely positive so far), we are in the process of gathering typical segmentation errors that result in unpleasing artifacts. For instance, in the photograph of a person, the eyes or the eyebrows may be classified as

Name	Description	Number Docs	Total Size (in MBytes)
Brattain	handwritten notes	6	55
Cuisine	XVII Century book	8	96
Curry	book with figures	3	40
Flammarion	XIX Century book	2	37
Forbes	magazine ads	15	331
Graham	magazine article	2	43
Hobbylobby	mail-order catalog	8	192
WashPost	newspaper	3	51
Maps	large map	1	50
Microphone	book with figures	9	65
Music	sheet music	8	116
Usa	US Constitution	5	155

Table 1: *Test document images. The third column gives the number of documents per category, the fourth column reports the total file size used by the raw RGB images*

Name	description	dpi	Subsamp.	Bits/Pix
Mask	bi-level mask	300	1	1
Bg	color background	100	3	24
Fg	color foreground	25	12	24

Table 2: *Description of the 3 DjVu sub-images*

foreground. It is fair to say that, on the data we have collected so far, these segmentation errors are infrequent.

7.3 Evaluation of JB2 and IW44 on Complex Document Images

Assuming a satisfactory foreground/background/mask separation, we can focus on the separate compression of the three sub-images. Our test sample, as described in Table 1, is composed of 70 document images (grouped in 12 categories) that would be problematic to transfer to a "symbolic" form (for instance PDF). They contain highly textured background (Cuisine, WashPost, Maps, Usa), handwriting (Brattain, Usa), a mixture of text and images (Forbes, Hobbylobby, Maps), mathematical symbols (Flammarion), hand drawings (Curry, Microphone. Usa) or music scores (Music).

For each image, our foreground/background separation algorithm produces 3 sub-images, as seen in Table 2.

Background (resp. foreground) images are generated by taking the page image and using an *mask interpolation algorithm.* Masked pixels are replaced by the average color of the neighboring visible pixels. This averaging is achieved by dividing the page into small blocks of two by two pixels. If a block contains both masked and visible pixels, the masked pixels are set to the average value of the visible pixels in the block. The process is then repeated with bigger blocks until all masked pixels have been set with a suitable interpolation of the visible pixel colors.

The raw mask with 1 bit/pixel is 24 times smaller than the original . The raw background image is 3x3 smaller. The raw foreground image is 12x12 smaller. The *raw multi-layer image* yields a compression rate of 6.25:1, as shown in the first step of Figure 7. This number (which depends on the foreground and background subsampling factors only) is the "raw" compression rate obtained by using the separation algorithm alone.

On these sub-images, an objective comparison between DjVu and traditional approaches is possible. We measure, for each image, the gain in compression rate brought by the algorithms used in DjVu when compared to the best widely available compression standard for this type of image (CCITT-G4 for the mask, JPEG for the foreground and background).

Name	CCITT-G4	Lossless JB2	Lossy JB2
Brattain	16.2	22.0	22.2
Cuisine	14.9	22.9	29.5
Curry	11.6	19.6	31.0
Flammarion	14.5	28.7	47.6
Forbes	19.5	38.5	53.3
Graham	8.8	23.9	42.4
Hobbylobby	15.2	32.6	47.5
WashPost	9.7	15.6	20.4
Maps	11.5	20.0	24.6
Microphone	10.5	19.9	29.9
Music	18.6	24.3	24.5
Usa	10.4	16.3	17.0

Table 3: *Ratios between the total raw file size and the total compressed file size for each document category. Compression formats are CCITT-G4, lossless and lossy JB2 (cf. Section 5.2)*

The mask is compressed with JB2, which, as we have seen, comes in 2 flavors, lossless or lossy compression. Table 3 reports comparative results: compression rates with lossy JB2 are between 1.5 and 5 times higher than with G4. Figure 7 reports the compression rates for the 70 images grouped together: 14:1 with G4 and 30:1 with JB2. This result is all the more interesting as these single-page documents contain, on average, very few repetitions of similar shapes that JB2 can code efficiently (except Graham, where the gain with JB2 is the most noticeable). This is also why the gain from lossless JB2 to lossy JB2 is lower than reported in [Howard, 1997].

Note that compared images must have the same quality. This is always true if the compression is lossless. The change in quality caused by lossy JB2 is only noticeable when zooming on the image.

Comparison between our IW44 wavelet compression and JPEG must be done on images with comparable distortion with respect to the original. As we are only interested in compressing the background (resp. the foreground), this distortion should be measured on the background (resp. foreground) pixels only: we use the average mean square error on these pixels (Masked MSE). We impose that the background JPEG-compressed image yields the same masked MSE as the IW44-compressed image.

Figure 7 shows that the overall compression ratio for the background sub-images moves from 59:1 if we use JPEG up to 103:1 if we use IW44, with the *successive projection algorithm* for partially masked images (cf. Section 5.3). The *mask interpolation algorithm* is somehow redundant with this *successive projection algorithm*, but ensures a fair comparison with JPEG (which would otherwise spend too many bits coding for masked pixels)

The *successive projection algorithm* is indispensable for the compression of the foreground sub-images, where most of pixels are masked. In this case, the *mask interpolation algorithm* outputs a very non-smooth representation of the image (made out of 2x2 squares of various sizes). The compression ratio obtained with our smooth wavelets is only 31:1 (less that JPEG, which yields 35:1), but goes up to 52:1 with the *successive projection algorithm*.

Overall, Figure 7 shows that the novel compression techniques used in DjVu to compress the mask, foreground and background sub-images compress our typical 23MBytes image into a 59 KBytes DjVu file. This is about half what we would have obtained by using the foreground/background algorithm with standard techniques only.

7.4 Evaluation of DjVu on Bi-Level Document Images

We downloaded a standard benchmark for bi-level document images, the OCR Lab 10 sample images (available at `http://imagebiz.com/PaperWeb/ocrlab.htm`). This benchmark has been designed to test various methods to digitize bi-level images, mostly with the help of OCR. On bi-level images, DjVu has only one layer: the mask coded with JB2. DjVu requires 268 KBytes to code the 10 images: this is one fourth of what G4 requires (1056 KBytes) and half of what PDF with Acrobat Capture 2.0 requires (498 KBytes).

Other experiments show that, on clean bi-level images (obtained, for instance, by converting a PostScript document into a bitmap), the compression rate with DjVu is between 7 and 8 times better than with G4.

In summary, on bi-level and color images, the DjVu compression rate is 4 to 10 times higher than traditional techniques such as CCITT-G4 or JPEG, at the same level of readability. Every algorithm used in DjVu, foreground/background separation, JB2 compression for the mask, IW44 compression for the background, contributes to this gain in performance in a significant way. Wavelet-based techniques, which are likely to be used in the future JPEG2000 standard, are still very far from DjVu in terms of readability for a given compression rate, and any progress in these techniques will also benefit DjVu , as they can be used to encode a background image which accounts for nearly 50% of the DjVu file size.

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 ratios 5 to 10 times higher than JPEG.

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* algorithms, our lossy JBIG2 compression ratios are on average twice those of lossless JBIG1, the best existing standard for bi-level images. The *ZP-coder* is faster than other approximate arithmetic coders, and yields, on average, better compression. The *multi-scale successive projections* algorithm for wavelet decomposition of partially masked images significantly reduces the compressed file sizes and can handle arbitrarily complex masks with reasonable computational requirements. All these algorithms have been integrated into a standalone DjVu encoder.

A 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 documents.

References

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

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

[Bottou and Bengio, 1995] Bottou, L. and Bengio, Y. (1995). Convergence properties of the Kmeans algorithm. In *Advances in Neural Information Processing Systems*, volume 7, Denver. MIT Press.

[Bottou et al., 1998] Bottou, L., Howard, P. G., and Bengio, Y. (1998). The Z-coder adaptive binary coder. In *Proceedings of IEEE Data Compression Conference*, pages 13–22, Snowbird, UT.

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

[Holt and Xydeas, 1986] Holt, M. J. and Xydeas, C. S. (1986). Recent developments in image data compression for digital facsimile. *ICL Technical Journal*.

[Howard, 1997] Howard, P. G. (1997). Text image compression using soft pattern matching. *Computer Journal*, 40(2/3):146–156.

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

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

[LeCun et al., 1997] LeCun, Y., Bottou, L., , and Bengio, Y. (1997). Reading checks with multilayer graph transformer networks. In *Proceedings of IEEE Int. Conference on Acoustics, Speech and Signal Processing ICASSP'97*.

[LeCun et al., 1995] LeCun, Y., Jackel, L. D., Bottou, L., Brunot, A., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Muller, U. A., Sackinger, E., Simard, P., and Vapnik, V. (1995). Comparison of learning algorithms for handwritten digit recognition. In Fogelman, F. and Gallinari, P., editors, *International Conference on Artificial Neural Networks*, pages 53–60, Paris.

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

[MacQueen, 1967] MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In LeCam, L. M. and Neyman, J., editors, *Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics, and Probabilities*, volume 1, pages 281–297, Berkeley and Los Angeles, (Calif). University of California Press.

[Mohiuddin et al., 1984] Mohiuddin, K., Rissanen, J. J., and Arps, R. (1984). Lossless binary image compression based on pattern matching. In *Proc. Intl. Conf. on Computers, Systems, and Signal Processing*, Bangalore.

[MRC, 1997] MRC (1997). Mixed rater content (MRC) mode. ITU Recommendation T.44.

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

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

[Said and Pearlman, 1996] Said, A. and Pearlman, W. A. (1996). 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.

[Sezan and Stark, 1982] Sezan, M. I. and Stark, H. (1982). Image restoration by the method of convex projections: Part 2 – applications and numerical results. *IEEE Transactions on Medical Imaging*, MI-1(2):95–101.

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

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

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

[Withers, 1996] Withers, W. D. (1996). A rapid entropy-coding algorithm. Technical report, Pegasus Imaging Corporation. URL ftp://www.pegasusimaging.com/pub/ELSCODER.PDF.

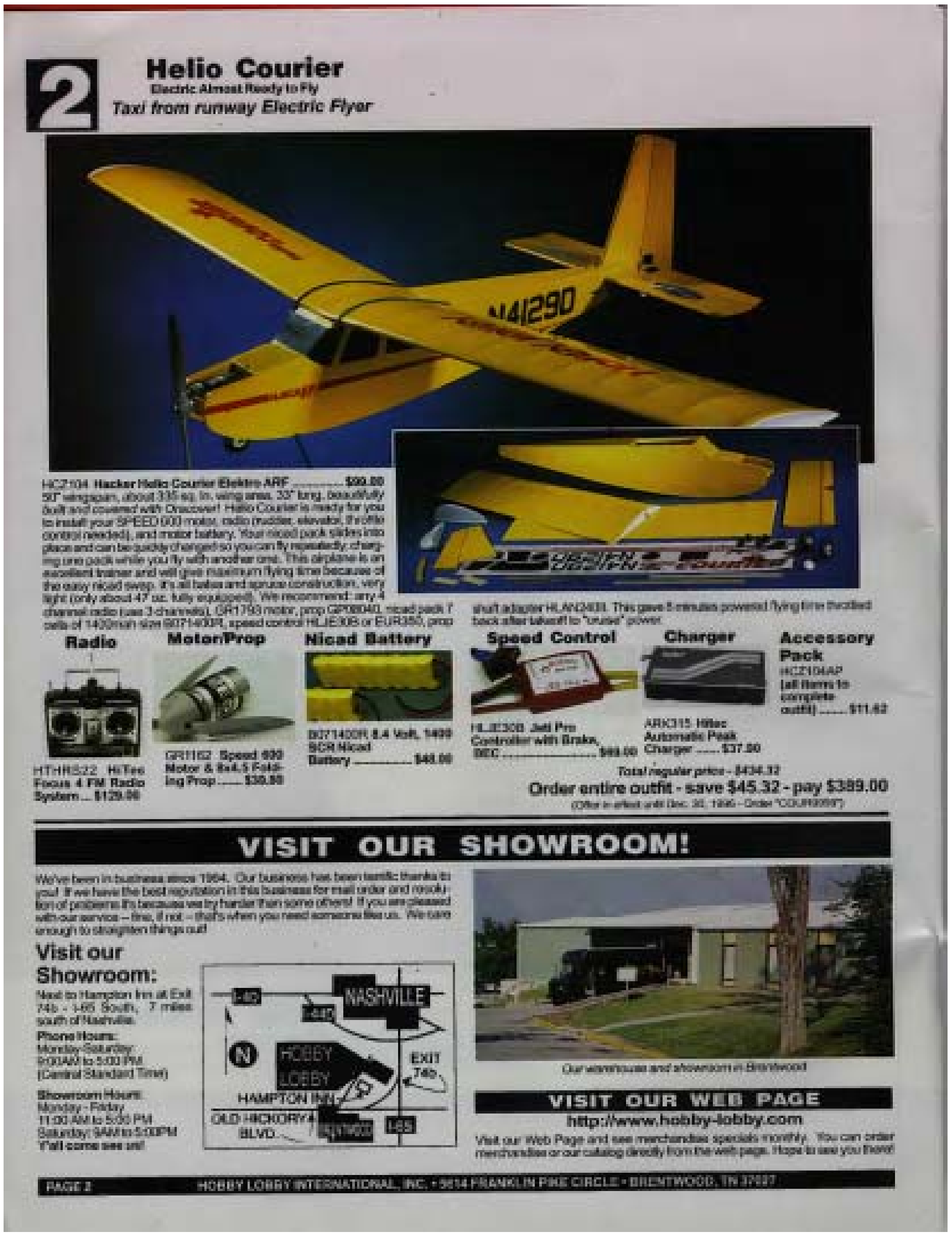
[Witten et al., 1992] Witten, I. H., Bell, T. C., Harrison, M. E., James, M. L., and Moffat, A. (1992). Textual image compression. In Storer, J. A. and Cohn, M., editors, *Proc. Data Compression Conference*, pages 42–51, Snowbird, Utah.

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

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

[Wu et al., 1997] Wu, V., Manmatha, R., and Riseman, E. (1997). Finding text in images. In *Proceedings of the 2nd ACM International Conference on Digital Libraries (DL'97)*, Lausanne.

[Youla and Webb, 1982] Youla, D. C. and Webb, H. (1982). Image restoration by the method of convex projections: Part 1 – theory. *IEEE Transactions on Medical Imaging*, MI-1(2):81–84.



2 **Helio Courier**

Electric Almost Ready to Fly

Taxi from runway Electric Flyer

HCZ104 **Hacker Helio Courier Elektro ARF** **$99.00**
50" wingspan, about 335 sq. in. wing area, 33" long, beautifully built and covered with Oracover! Helio Courier is ready for you to install your SPEED 600 motor, radio (rudder, elevator, throttle control needed), and motor battery. Your nicad pack slides into place and can be quickly changed so you can fly repeatedly, charging one pack while you fly with another one. This airplane is an excellent trainer and will give maximum flying time because of the easy nicad swap. It's all balsa and spruce construction, very light (only about 47 oz. fully equipped). We recommend: any 4 channel radio (uses 3 channels), GR1793 motor, prop GP08040, nicad pack 7 cells of 1400mah size 8071400R, speed control HLJE30B or EUR350, prop shaft adapter HLAN2400. This gave 8 minutes powered flying time throttled back after takeoff to "cruise" power.

Radio

HTHRS22 **HiTec Focus 4 FM Radio System** ... **$129.00**

Motor/Prop

GR1162 **Speed 600 Motor & 8x4.5 Folding Prop** **$30.00**

Nicad Battery

8071400R **8.4 Volt, 1400 SCR Nicad Battery** **$48.00**

Speed Control

HLJE30B **Jeti Pro Controller with Brake, BEC** **$89.00**

Charger

ARK315 **Hitec Automatic Peak Charger** **$37.00**

Accessory Pack

HCZ104AP **(all items to complete outfit)** **$11.62**

Total regular price - $434.32

Order entire outfit - save $45.32 - pay $389.00

(Offer in effect until Dec. 30, 1996 - Order "COUR[illegible]")

VISIT OUR SHOWROOM!

We've been in business since 1964. Our business has been terrific thanks to you! If we have the best reputation in this business for mail order and resolution of problems it's because we try harder than some others! If you are pleased with our service – fine, if not – that's when you need someone like us. We care enough to straighten things out!

Visit our Showroom:

Next to Hampton Inn at Exit 74b - I-65 South, 7 miles south of Nashville.

Phone Hours:
Monday-Saturday:
9:00AM to 5:00 PM
(Central Standard Time)

Showroom Hours:
Monday - Friday
11:00 AM to 5:00 PM
Saturday: 9AM to 5:00PM
Y'all come see us!

Our warehouse and showroom in Brentwood

VISIT OUR WEB PAGE

http://www.hobby-lobby.com

Visit our Web Page and see merchandise specials monthly. You can order merchandise or our catalog directly from the web page. Hope to see you there!

PAGE 2 HOBBY LOBBY INTERNATIONAL, INC. • 5614 FRANKLIN PIKE CIRCLE • BRENTWOOD, TN 37027

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

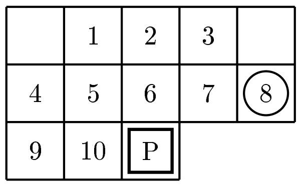


Figure 2: JBIG1 template. The numbered pixels are used as the context for coding of the pixel marked 'P'.

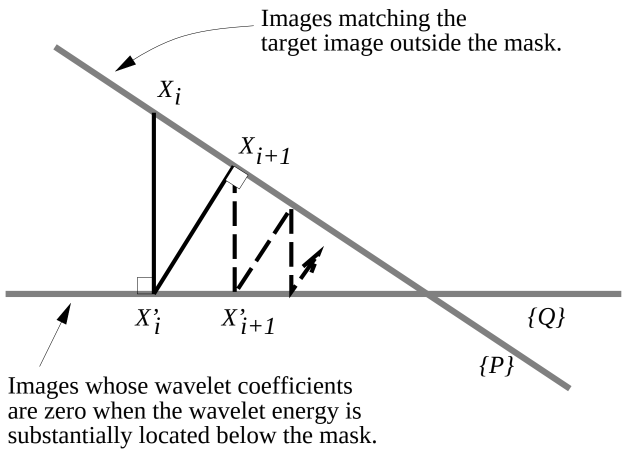
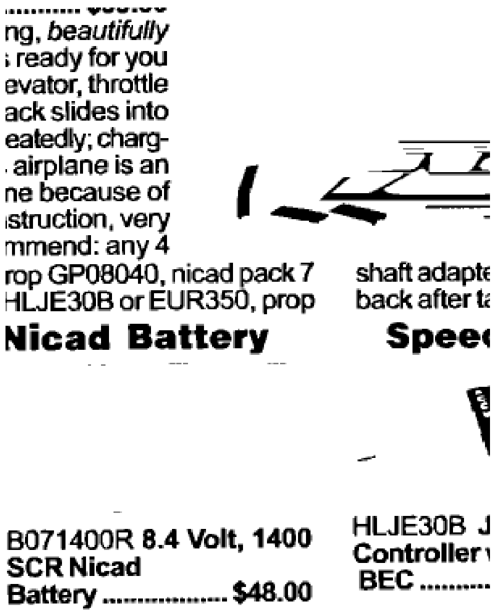
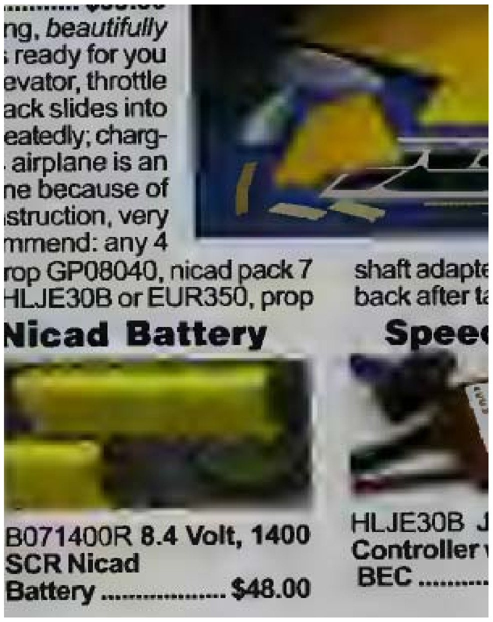

Images matching the
target image outside the mask.
X_i
X_{i+1}
X'_i
X'_{i+1}
{Q}
{P}
Images whose wavelet coefficients
are zero when the wavelet energy is
substantially located below the mask.

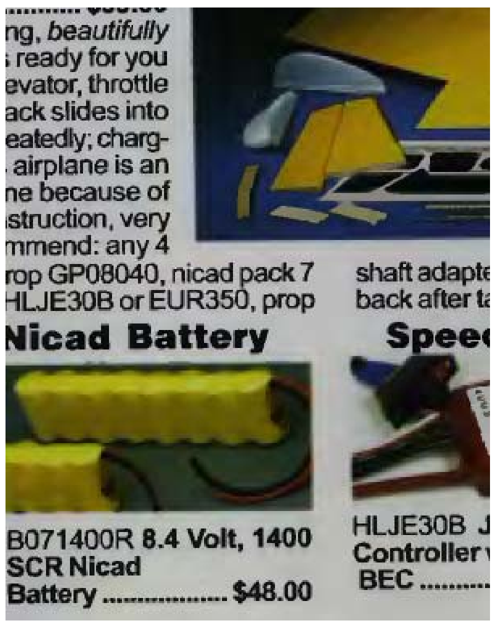
Figure 3: *Successive projections converge towards a point in the intersection of two convex subspaces representing (a) images matching the initial image outside the mask, and (b) images whose masked wavelet coefficients are zero.*



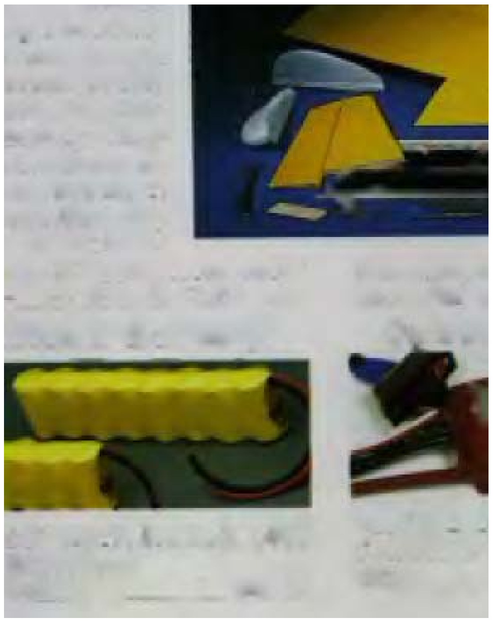
3.1sec/23k: the *mask* (text, 23K) is loaded.



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



9.4sec/67K: Loading is finished.



35K are necessary for this background image.

Figure 4: *Downloading through a 56K modem: progressive decompression of text first, followed by the background at increasing quality(detail of Figure 1)*

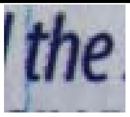
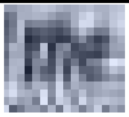
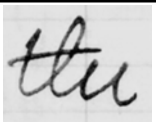
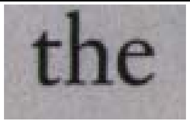
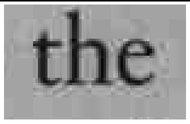
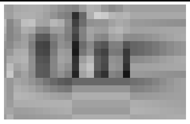
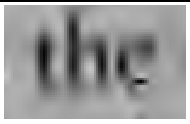
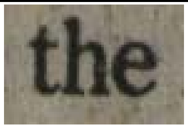
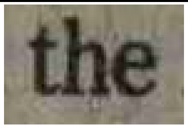
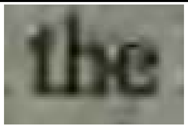
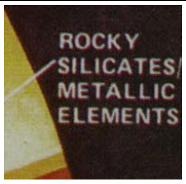
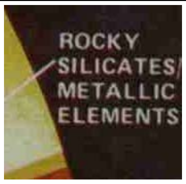
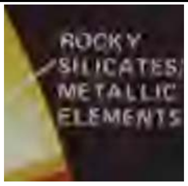
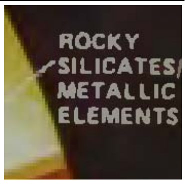
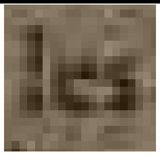
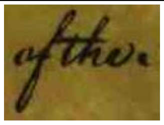
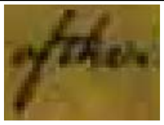
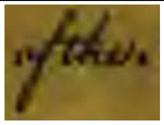
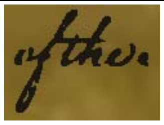
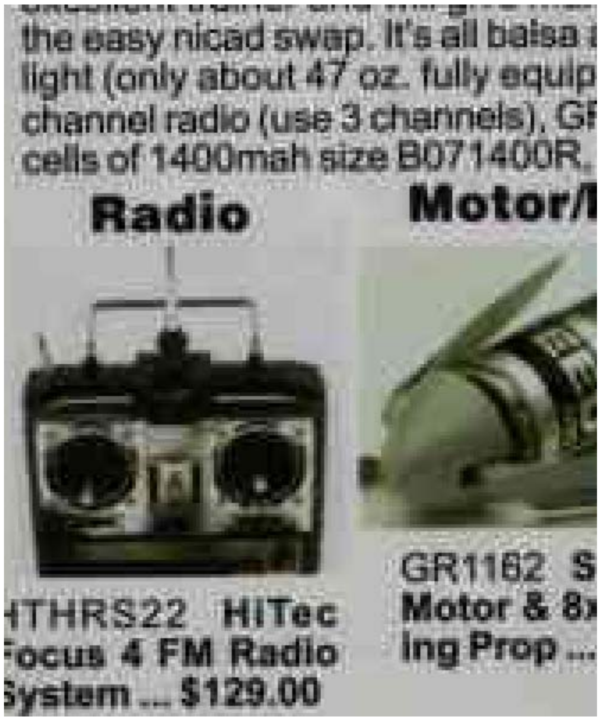
Image Description	Raw image detail	JPEG, 300dpi, quality 20	JPEG, 100dpi, size=DjVu	IW44, 300dpi, size=DjVu	**DjVu compressed**
Magazine Add % image= 56					
`ads-freehand-300`	20640K	292K 70:1	50K 412:1	61K 338:1	52K 396:1
Brattain Notebook % image= 22					
`brattain-0001`	9534K	116K 82:1	17K 560:1	20K 476:1	19K 501:1
Scientific Article % image= 46					
`graham-001`	22013K	383K 57:1	41K 536:1	43K 511:1	38K 579:1
Newspaper Article % image= 50					
`lrr-wpost-1`	12990K	250K 51:1	38K 341:1	42K 309:1	40K 324:1
Cross-Section of Jupiter % image= 73					
`planets-jupiter`	24405K	284K 85:1	47K 519:1	52K 469:1	47K 519:1
XVIIIth Century book % image= 45					
`cuisine-p006`	12128K	206K 58:1	35K 346:1	39K 310:1	37K 327:1
US First Amendment % image= 30					
`usa-amend1`	31059K	388K 80:1	77K 403:1	78K 398:1	73K 425:1

Figure 5: *Compression results for seven selected images with 4 compression formats.* `Raw` *applies no compression. JPEG-100dpi: the image is low-pass filtered and subsampled to 100dpi, and then compressed with JPEG while adjusting the quality parameter so as to obtain a file approximately the same size as the DjVu . 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).*



the easy nicad swap. It's all balsa
light (only about 47 oz. fully equip
channel radio (use 3 channels), GF
cells of 1400mah size B071400R,

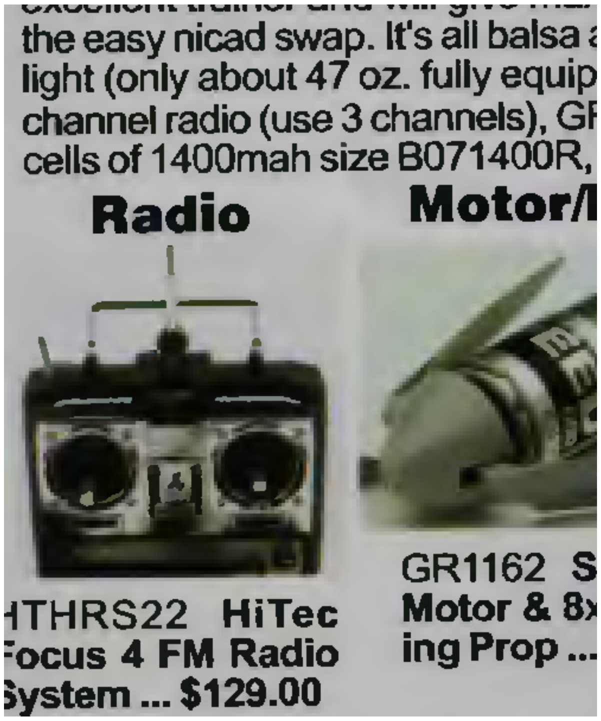
Radio

Motor/

HTHRS22 HiTec
Focus 4 FM Radio
System ... $129.00

GR1162 S
Motor & 8x
ing Prop ...

JPEG at 100dpi only.



the easy nicad swap. It's all balsa a
light (only about 47 oz. fully equip
channel radio (use 3 channels), GF
cells of 1400mah size B071400R,

Radio

Motor/

HTHRS22 HiTec
Focus 4 FM Radio
System ... $129.00

GR1162 S
Motor & 8x
ing Prop ...

DjVu.

Figure 6: *Comparison of JPEG at 100dpi (left) with quality factor 30% and DjVu (right). The images are cropped from* **hobby002**. *The file sizes are 82 KBytes for JPEG and 67 KBytes for DjVu*

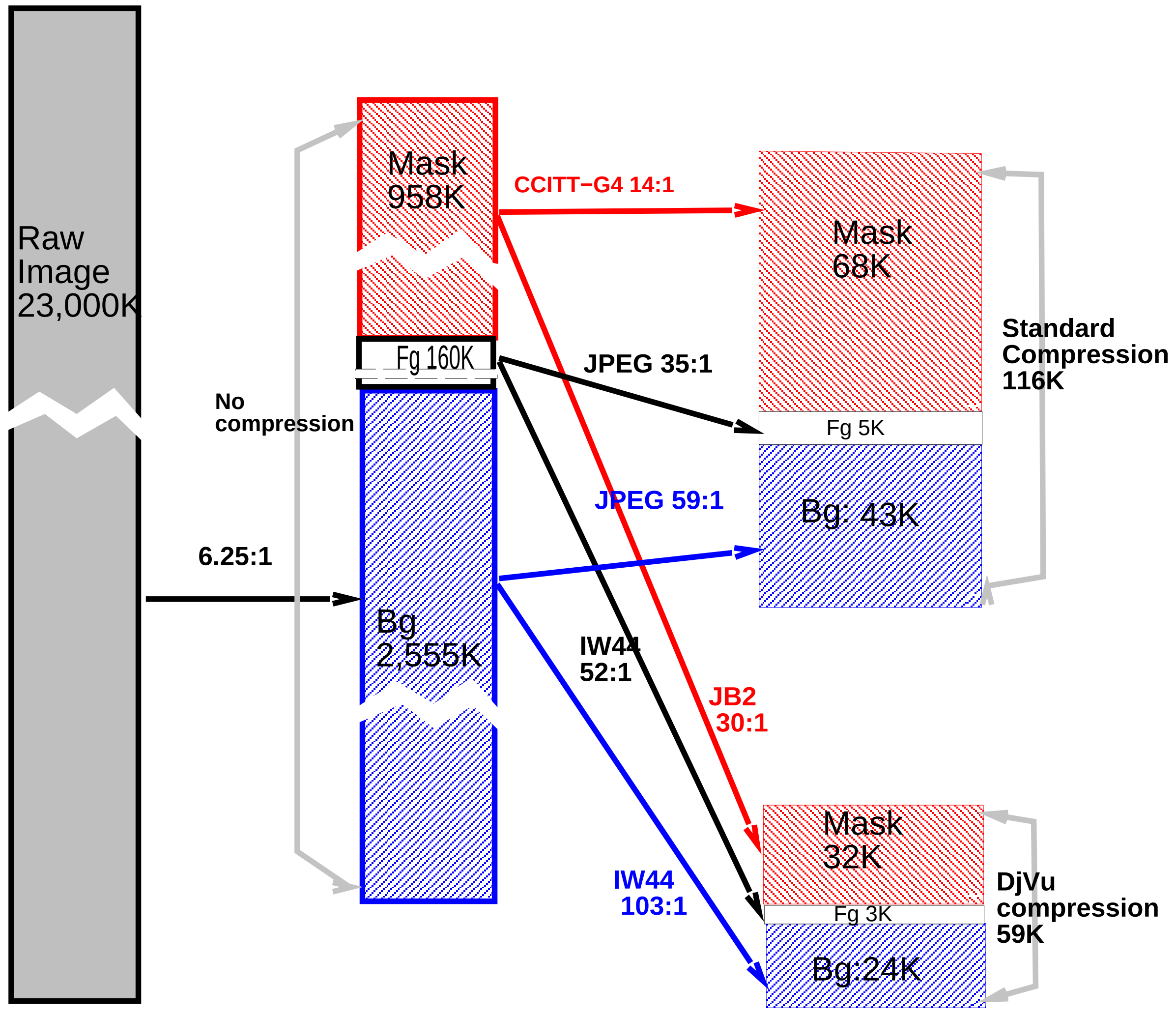


Figure 7: *For a given foreground/background separation, this figure compares, for the three sub-images, the following configurations: (i) no compression, (ii) standard compression techniques such as JPEG and CCITT-G4 and (iii) compression techniques used in DjVu. Assuming we start from a 23MBytes raw document image, the average size we can expect for each sub-image is reported in the corresponding block. The arrows show the techniques used and the compression rates obtained.*