DjVu Document Browsing with On-Demand loading and rendering of image components

Yann Le Cun, Léon Bottou, Andrei Erofeev, Patrick Haffner, Bill Riemers[a]

AT&T Labs - Research, Middletown, NJ

ABSTRACT

Image-based digital documents are composed of multiple pages, each of which may be composed of multiple components such as the text, pictures, background, and annotations. We describe the image structure and software architecture that allows the DjVu system to load and render the required components on demand while minimizing the bandwidth requirements, and the memory requirements in the client. DjVu document files are merely a list of enriched URLs that point to individual files (or file elements) that contain image components. Image components include: text images, background images, shape dictionaries shared by multiple pages, OCRed text, and several types of annotations. A multithreaded software architecture with smart caching allows individual components to be loaded and pre-decoded and rendered on-demand. Pages are pre-fetched or loaded on demand, allowing users to randomly access pages without downloading the entire document, and without the help of a byte server.

Components that are shared accross pages (e.g. shape dictionnaries, or background layers) are loaded as required and cached. This greatly reduces the overall bandwidth requirements. Shared dictionnaries allow 40% typical file size reduction for scanned bitonal documents at 300dpi. Compression ratios on scanned US patents at 300dpi are 5.2 to 10.2 times higher than GroupIV with shared dictionnaries and 3.6 to 8.5 times higher than GroupIV without shared dictionnaries.

Keywords: DjVu, document imaging, scanning, wavelets, segmentation, digital libraries

1. INTRODUCTION

As the world transitions from the universe of paper to the universe of instantly accessible digital content, document imaging, document image compression, and document image distribution have become prominent research topics.

Although, the Internet has given us a worldwide infrastructure on which to build the universal library, much of the world knowledge, history, and literature is still trapped on paper in the basements of the world's traditional libraries. Many libraries and content owners are in the process of digitizing their collections. While many such efforts involve the painstaking process of converting paper documents to computer-friendly form, such as SGML-based formats, the high cost of such conversions limits their extent. Scanning documents, and distributing the resulting images electronically is not only considerably cheaper, but also more faithful to the original document because it preserves its visual aspect.

Despite the quickly improving speed of network connections and computers, the number of scanned document images accessible on the Web today is relatively small. There are several reasons for this.

The first reason is the relatively high cost of scanning anything else but unbound sheets in black and white. This problem is slowly going away with the appearance of fast and low-cost color scanners with sheet feeders. In addition, the emergence of look-down book scanners based on ultra high resolution digital cameras (above 8 megapixels) will soon reduce the cost of non-destructive book scanning dramatically.

The second reason is that long-established image compression standards and file formats have proved inadequate for distributing scanned documents at high resolution, particularly color documents. Not only are the file sizes and download times impractical, the decoding and rendering times are also prohibitive. A typical magazine page scanned in color at 100 DPI in JPEG would typically occupy 100 KB to 200 KB, but the text would be hardly readable: insufficient for screen viewing and totally unacceptable for printing. The same page at 300 DPI would have sufficient

E-mail: {yann,leonb,haffner}@research.att.comA.)

[a]B. R. is now with Lizardtech, Seattle, WA, Email: bcr@lizardtech.com

quality for viewing and printing, but the file size would be 300 KB to 1000 KB at best, which is impractical for remote access. Another major problem is that a fully decoded 300 DPI color images of a letter-size page occupies 24 MB of RAM (2500x3200 pixels), which is more than what a typical PC can handle without heavy disk swapping.

The third reason is that scanned documents are more than just a collection of individual page images. Pages in a scanned documents have a natural serial order. Special provision must be made to ensure that flipping pages be instantaneous and effortless so as to maintain a good user experience. Even more important, most existing document formats force users to download the entire document first before displaying a chosen page. However, users often want to jump to individual pages of the document without waiting for the entire document to download. Efficient browsing requires efficient random page access, fast sequential page flipping, and quick rendering. This can be achieved with a combination of advanced compression, pre-fetching, pre-decoding, caching, and progressive rendering. Lastly, advanced document image compression systems, such as the one described in this paper, decompose each page into multiple components (text, backgrounds, images, libraries of common shapes...) that may be shared by several pages and downloaded on demand. All these requirements call for a very sophisticated but parsimonious control mechanism to handle on-demand downloading, pre-fetching, decoding, caching, and progressive rendering of the page images. What is being considered here is not just a document image compression technique, but a whole platform for document delivery.

DjVu is an image compression technique, a document format, and a software platform for delivering documents images over the Internet that fulfills the above requirements. The DjVu image compression is based on three techniques:

- DjVuBitonal, also known as JB2, is a bitonal image compression that takes advantage of repetitions of nearly identical shapes on the page (such as characters) to efficiently compress text images. A typical 300dpi page in DjVuBitonal occupies 5 to 25KB (3 to 8 times better than TIFF-GroupIV or PDF).
- DjVuPhoto, also known as IW44, is a wavelet-based continuous-tone image compression technique with progressive decoding/rendering. Images are typically half the size as JPEG for the same distortion.
- DjVuDocument, or simply DjVu, a compression technique specifically designed for scanned documents in color. DjVuDocument segments images into separately compressed layers, one of which is compressed with DjVu Bitonal and contains the text and drawings, the other one which is compressed with DjVuPhoto and contains the backgrounds and pictures at lower resolution.

Earlier versions of the compression techniques were described in several papers.[1,2] The present article briefly reports results obtained with the latest incarnation of the compressor (v3.0), particularly the improvements to textual image compression brought about by extracting shape libraries that are shared by multiple pages. The rest of the paper concentrates on the design and implementation choices that were made to optimize the user experience when accessing DjVu documents over the Web. Much of the design choices were driven by an attempt to minimize the delay between the user's request for a page and the moment it appears on the display.

2. DJVU OVERVIEW

2.1. DjVuBitonal/JB2

The bitonal compression method employed in DjVu is a variation of AT&T's original proposal to the JBIG-2 standard committee. The algorithm consists in isolating individual components in a page (generally the connected components) and clustering them into groups of similar shapes.[3] Prototype shapes for each cluster are compressed as bitmaps with a technique similar to the JBIG standard. Each pixel in the shape is coded using a fast adaptive arithmetic coder called the ZP-coder.[4] The arithmetic coder uses a *context* formed by previously transmitted neighboring pixels to predict the value of the pixel under consideration, and to code its value using a nearly optimal amount of information (within a few percent of the Shannon limit). Other shapes in a cluster are coded using a context that includes neighboring pixels as well as corresponding pixels in the prototype shape of the cluster.[1] This strategy minimizes the overall bit count because most pixels in the shape and in the prototype are identical. Since perfectly lossless compression is rarely useful, compression ratios can be considerably improved by substituting a shape by its cluster prototype whenever the difference between the two is small enough to be imperceptible. Positions at which the shapes are drawn on the page are also coded using the ZP-coder.

In cases where the document has multiple pages, it is advantageous to build a library of shapes that are common to all the pages, in addition to the library of shapes specific to each page. DjVuBitonal uses a fast incremental technique to extract common shape libraries across pages.

In the following, we report benchmark results of DjVuBitonal v2.0 and v3.0 compared with CCITT-GroupIV (also known as MMR). The main differences between DjVuBitonal v3.0 and v2.0 include the common shape library extraction, better handling of half-tones, much faster compression, and better pattern matching.

An independent test for lossless compression[5] on a large collection of documents gives compression ratios of 26.5 for DjVuBitonal v2.0. This can be compared with 13.5 obtained with MMR/Group 4 and 19.4 obtained with JBIG.

Results in lossy mode are more interesting. Table 1 shows results with DjVuBitonal compared with MMR/CCITT–GroupIV. Compression ratios obtained with DjVu v3.0 in multi-page mode with shared shape library are 4.5 to 10 times better than MMR/G4. Average page sizes are 5 to 11 KB, depending on the document. A comparison with PDF is in order. PDF applied to scanned bitonal documents merely encapsulates MMR/G4 versions of the pages into a PDF container. Therefore, PDF files sizes are similar to MMR/G4.

Document Nips10 is particularly interesting because it is a typical scientific publication. With it 1090 pages, the TIFF/G4 (or PDF) file occupies 76MB. In DjVu, it reduces to about 13MB. The average page size is about 10KB, which suggests that about 600,000 such pages could be stored on a CD-ROM (50 volumes like that one).

Document "Snowbird" has not been scanned but directly converted from PostScript. The original PostScript requires about 10.7 MB. Compressed with "gzip", it requires 3.1 MB. Converted to TIFF/G4 at 300 dpi with GhostScript, it requires 7.4 MB. Compressed with DjVu with shared shape library, it occupies a mere 728KB. It is a factor of 10 over TIFF/G4, 14 compared to PostScript, and 4.25 compared to gzipped PostScript.

2.2. DjVuPhoto/IW44

For continuous-tone images, DjVu uses a wavelet-based compression algorithm called DjVuPhoto or IW44 that has several advantages over other existing wavelet coders. First, IW44/DjVuPhoto uses the fast "lifting" method to perform the wavelet transform .[6] Second, this transform is implemented so as not to require multiplies, only shifts and adds, in order to accelerate the computation. Third, the intermediate image representation in memory is designed to allow progressive refinement of the wavelet coefficients while occupying a memory footprint proportional to the number of non-zero coefficients, and not to the number of pixels. Fourth, this intermediate representation can be used to render any sub-image at any zoom factor efficiently. Fifth, the multiple-projection masking technique allows to spend bits only on parts of the image that are actually visible, and spend no bits on parts that are covered by other elements. Lastly, IW44 uses the ZP-coder to code its wavelet coefficients.

The performance of IW44 is typical of the latest wavelet-based compression methods, but it has been heavily optimized to minimize the decoding time and memory footprint rather than maximize the compression ratio.

The size of an IW44 file is generally 30% to 50% less than a JPEG with the same signal to noise ratio, but the biggest advantage over JPEG is the progressive refinement. Table 2 gives SNR obtained with IW44 on two standard test images (Lena et Goldhill), and compares with JPEG (using "cjpeg" from the Independent JPEG Group), and several other wavelet-based methods described in the literature: EZW,[7] SPIHT,[8] and EBCOT.[9] Comparisons of DjVuPhoto and JPEG can be seen at `http://www.djvuzone.org/djvu/photos/jpgvsdjvu01`.

On Lena, IW44 dominates JPEG by about 2dB. IW44 beats also EZW, but is not quite as good as SPIHT and EBCOT. This is because IW44 was optimized for fast decoding, not compression ratio (c.f. section 3). SPIHT et EBCOT use the 9-7 filter of Antonini et al.[10] computed in floating point. IW44 uses 4/4 Deslauriers-Dubuc filter computed in 16 bit fixed point with no multiplies (only additions subtractions and shifts. Using a more advanced filter would no doubt increase the compression ratios, but would slow down the decoding.

2.3. DjVuDocument

The basic idea behind DjVuDocument is to separate the text from the backgrounds and pictures and to use different techniques to compress each of those components. DjVuDocument segments images into separately compressed layers:

- The **background layer** is encoded with DjVuPhoto and contains a low resolution (e.g. 100 dpi.) image representing details that are best encoded using a continuous tone technique. This usually includes the document background and the pictures.

Total file sizes
(average size per page)

Document	pages	raw (KB)	G4 (bytes)	JB2 v2.0 (bytes)	JB2-3(1) (bytes)	JB2-3(10) (bytes)	JB2-3(inf) (bytes)
p5960411	19	18338	880372	154923	129912	114610	110148
		(965)	(46335)	(8153)	(6837)	(6032)	(5797)
p5960412	167	161181	3819436	1129964	1056946	842448	730805
		(965)	(22870)	(6766)	(6329)	(5044)	(4376)
p5960414	12	11582	769390	109700	91278	79581	77855
		(965)	(64115)	(9141)	(7606)	(6631)	(6487)
p5960413	17	16408	1585926	238924	201166	182141	180435
		(965)	(93289)	(14054)	(11833)	(10714)	(10613)
p5960415	18	17373	955482	147030	112629	92793	90028
		(965)	(53082)	(8168)	(6257)	(5155)	(5001)
p5960416	53	51153	2750910	559900	469527	410680	393792
		(965)	(51903)	(10564)	(8859)	(7748)	(7430)
Snowbird	133	137067	7406482	1099934	988321	869625	728196
		(1030)	(55687)	(8270)	(7430)	(6538)	(5475)
Nips10	1090	1276971	76443591	n/a	15838908	12692602	11013261
		(1171)	(70131)	(n(a)	(14531)	(11644)	(10103)

Compression Ratios (and ratios relative to CCITT-GroupIV)

Document	pages	raw (KB)	G4 (ratio)	JB2 v2.0 (ratio)	JB2-3(1) (ratio)	JB2-3(10) (ratio)	JB2-3(inf) (ratio)
p5960411	19	18338	20.8	118.4 (5.7)	141.2 (6.8)	160.0 (7.7)	166.5 (8.0)
p5960412	167	161181	42.2	142.6 (3.4)	152.5 (3.6)	191.3 (4.5)	220.6 (5.2)
p5960414	12	11582	15.1	105.6 (7.0)	126.9 (8.4)	145.5 (9.7)	148.8 (9.9)
p5960413	17	16408	10.3	68.7 (6.6)	81.6 (7.9)	90.1 (8.7)	90.9 (8.8)
p5960415	18	17373	18.2	118.2 (6.5)	154.2 (8.5)	187.2 (10.3)	193.0 (10.6)
p5960416	53	51153	18.6	91.4 (4.9)	108.9 (5.9)	124.6 (6.7)	129.9 (7.0)
Snowbird	133	137067	18.5	124.6 (6.7)	138.7 (7.5)	157.6 (8.5)	188.2 (10.2)
Nips10	1090	1276971	16.7	n/a (n/a)	80.6 (4.8)	100.6 (6.0)	115.9 (6.9)

Table 1. Comparison between MMR/G4, and JB2 (DjVu bitonal). The first six documents are patents from the U.S. patent office web site scanned at 300 dpi, Nips10 is volume 10 of the NIPS proceedings scanned at 400dpi, Snowbird is a collection of conference abstracts in PostScript (no scanning). The columns give the following information: "page": number of pages; "raw": size of the original file at 1 bit per pixel. "MMR/G4": MMR/CCITT-GroupIV compression; "JB2 v2.0": previous generation DjVuBitonal; "JB2 v3.0(1)": new generation DjVuBitonal with shared library extraction turned off. "JB2-3(10)": new DjVuBitonal with shared shape library across groups of 10 page; "JB2-3(inf)": latest DjVuBitonal with a single shared shape library for the whole document.

- The **foreground layer** contains a high resolution bitonal mask (e.g. 300 dpi) encoded with DjVuBitonal. The mask accurately defines the shape of details with sharp edges such as text and line-art. The color information is either encoded as a solid color per connected component in the mask, or as a very low resolution image (e.g. 25 dpi) whose colors are applied using the mask as a stencil.

This separation is achieved using a segmentation algorithm that decides which pixels represent components of the foreground or background layer.

The first step of the segmentation algorithm relies on a bi-dimensional causal Markov Field whose states represent the foreground and background layers. Each state locally describes the pixel color distribution in the corresponding layer as a Gaussian distribution parametrized by the color mean and variance. The transition probabilities are tuned to produce an *over-segmentation* in order to locate all objects that might be better encoded into the foreground layer. High contrast details in pictures might be at this stage classified as foreground objects. A variety of filters are then applied to the resulting foreground image so as to eliminate such mistakes.

Image	bpp	IW44	JPEG	EZW	SPIHT	EBCOT
Lena	0.25	33.62	31.67	33.17	34.11	34.28
Lena	0.50	36.61	34.84	36.28	37.21	37.43
Lena	1.00	39.67	37.94	39.55	40.46	40.61
Goldhill	0.25	30.25	29.23	n/a	30.56	n/a
Goldhill	0.50	32.56	31.03	n/a	33.12	n/a

Table 2. SNR (in dB) obtained with standard test images using IW44, JPEG, a several wavelet-based methods: EZW, SPIHT, et EBCOT. The wavelet "filter" used for IW44 is very fast but sub-optimal for the compression ratio. The wavelets used are as follows: IW44: 5 levels, 4/4 Deslauriers-Dubuc filter; JPEG: IJG code with optimized Huffman tables; EZW: 6 levels, order 9 QMF, "coding" "Embedded Zero Tree"; SPIHT: 5 levels, 9-7 filter, with arithmetic coder; EBCOT: 5 levels, 9-7 filter.

Compression	none	GIF	JPEG	DjVu
Hobby p15	24715	1562	469	58
dict. medical	16411	1395	536	110
time zone	9174	576	249	36
cookbook	12128	1000	280	52
Hobby p17	23923	1595	482	52
U.S. Constit.	31288	2538	604	134
Hobby p2	23923	1213	383	68
ATT Olympic	23946	955	285	41

Table 3. File sizes (in KB) obtained from eight scanned documents at 300dpi with the following algorithms: no compression, GIF at 150 dpi, JPEG at 300 dpi (IJG-JPEG, quality 20%), and DjVu with a 300dpi mask and a 100dpi background. The visual quality of these documents can be observed at `http://www.djvuzone.org/djvu`

The main filter is designed to be as general as possible and avoids heuristics that would have to be tuned on hundreds of different kinds of documents. Since the goal is compression, the problem is to decide, for each foreground blob found by the previous algorithm, whether it is preferable to *actually* code it as foreground or as background. Two competing strategies are associated with data-generating models. Using a Minimum Description Length (MDL) approach,[11,12] the preferred strategy is the one that yields the lowest overall coding cost, which is the sum of the cost of coding the model parameters and the cost of coding the error with respect to this ideal model. Like most MDL approaches used for segmentation,[12] the motivation is to obtain a system with very few parameters to hand-tune. However, the MDL principle is used here to make only *one assymetrical* decision, thus avoiding the time consuming minimization of a complex objective function.

To code the blob as part of the smooth background only requires a background model. To code the blob as a piece of foreground that sticks out of the background requires a background model and a foreground mask model. The background model is a Gaussian distribution centered on the the average of the colors of the closest background pixels that can be found up and to the left. In regions which are masked by the foreground, the background color must be interpolated in a way to minimize the background reconstruction error. The foreground model assumes that the color of each connected component is uniform and that the shape of the connected components is distributed according to the length of its horizontal and vertical boundaries.

The foreground/background decision is taken by comparing the ratio of the foreground and background MDL costs with an adjustable threshold. A high value of the threshold over-segments the document image. The mask then contains noise and dithering that increase the size of the DjVuBitonal encoded foreground layer. A small value of the threshold under-segments the document image. Text might then be encoded in the background and increase the size of the DjVuPhoto encoded background layer.

The DjVuDocument encoding scheme provides compression ratios ranging from 300:1 to 1000:1. Table 3 provide DjVuDocument sizes for a few documents scanned in color at 300 dpi. Standard documents (magazines, mail order catalogs) require 30KB to 80 KB. Certain documents require between 80KB and 140KB because their physical size is larger or because they contain high quality pictures.

Action	Real-world equivalent	Acceptable delay
Zooming/Panning	Moving the eyes	Immediate
Next/Previous Page	Turning a page	<1 second
Random Page access	Finding a page	<3 seconds

Table 4. Acceptable delays associated with various types of user interaction. These delays must compare with the delays associated with equivalent actions when reading a paper document.

These sizes are 5 to 10 times smaller than those obtained by adjusting the JPEG quality factor in order to obtain a similarly legible image. The visual quality of a large number of scanned document images can be observed at `http://www.djvuzone.org/djvu`. Links to other sources of DjVu encoded documents on the Internet can be found under `http://www.djvuzone.org/links`.

3. BROWSING DJVU DOCUMENTS

The single most important aspect of the user experience is the delay between the user's request for a page and the moment it appears on the screen. The design of the DjVu document delivery platform was systematically optimized to minimize this delay. The overall delay is a combination of several factors: the time it takes a user to pick the page he or she wants to see using the user interface, the transmission time (directly related to file size), the decoding time, and the rendering time. This section describes some of the design choices.

3.1. Progressive decoding

Progressive decoding is a familiar feature of modern web browsers. The design of DjVu compression system provides considerable flexibility in that respect. Document images are represented by successive *chunks* encapsulated into an IFF85 container file format.[13] The foreground mask and the foreground colors are represented by different chunks. The background layer is composed of an arbitrary number of chunks representing successive refinements of the background image.

The DjVu viewer attempts to update the display immediately after receiving each such chunk. When only the mask chunk is available, a black and white image, generally containing the text and line art is displayed. When a foreground color chunk or one or more background chunks have arrived, the displayed image is upgraded to a full color image. The background is refined as more background chunks arrive.

Another level of flexibility is provided by the IW44 wavelet codec which is the current default encoder for the background layer. A background image can partitioned and repartitioned into chunks after compression without introducing additional loss. Similar capabilities are now offered by the forthcoming JPEG-2000 standard.[14]

3.2. User interaction

Reading a scanned document entails a number of frequently performed actions, such as zooming, panning, and page flipping, that are facilitated by the DjVu plug-in user interface. Many document viewing software use multiple "modes", e.g. panning mode, zooming mode, selection mode. We advocate instead a "modeless" interface where all the commonly used functions (panning, zooming, flipping pages) are directly accessible with a single mouse action or a single keyboard shortcut. This minimizes the delay between the user's decision to access a page and the moment the page appears on the screen.

The time required for performing these operations should match the time required to achieve the corresponding actions on a paper document. Table 4 summarizes the acceptable delays from the moment the user initiates the action to the moment the viewer completes the operation.

The first step consists in quickly initiating the operation, regardless of the status of the network connection or the image decoding machinery. Thanks to its multi-threaded architecture (section 4), the DjVu viewer starts reacting to user actions within milliseconds. The most challenging part still consists in completing the requested operation quickly enough. This is made possible by using several techniques described in the next subsections. The DjVu viewer meets the stringent requirements of table 4 when running on an average PC with a 300 Kbps Internet connection, but the performance is excellent even on a five years old PC with a 56 Kbps modem.

Image Representation	Memory	Panning speed	Progressivity
Raw Image Data	25 Mb	$-$	$--$
Compressed Data	50 Kb	$-$	$+$
Intermediate Data	1.5 Mb	$++$	$+$

Table 5. Qualitative comparison of various image representation schemes for the viewer. The "Memory" column indicates the typical memory size required for a letter sized page at 300 d.p.i.. The "Panning Speed" and "Progressivity" columns indicate the relative computational cost of implementing these features.

3.3. Internal Image Representation

How the image data is represented in memory by the viewer directly impacts the the zooming and panning speed and also affects the efficiency of the progressive decoder. Most image viewers simply decode the data and keep the fully decompressed image data in memory. This simple approach fails in two respects:

- A fully decompressed 300dpi document image in color occupies around 24MB of RAM, which causes heavy disk swapping on most PC configurations. Panning is impracticably slow if newly exposed parts of the image must be swapped-in from disk.
- Each pass of a progressive refinement scheme would have to update the entire decompressed image data. This complete decoding of the full image at each refinement pass would cause unbearably long delays in the display, even on high performance machines.

The other extreme solution is to keep the untouched compressed data in RAM, but the rendering time would be prohibitive and would prevent fast panning. Computational requirements might be reduced by splitting the image into separately encoded tiles and only decoding the few tiles that contain newly exposed pixels. Small tiles negatively impacts the compression ratio because they prevent taking advantage inter-tile redundancies. Large tiles negatively impact the panning speed since complete tiles must be decoded after each mouse movement.

The solution implemented in DjVu keeps the foreground and background layers in RAM in partially decompressed and compact intermediate forms. This intermediate representation is created or updated by partially decoding the compressed data arriving from the network connection. The delay induced by this operation is hardly noticeable since it overlaps the download time. Image pixels are rendered on the fly from this representation whenever the user pans around the image or change the zoom factor, or whenever a enough new data has arrived to warrant a display update. This process is efficient because the rendering process only computes the pixels actually being displayed. The typical size of this partially decompressed intermediate structure is around 1.5 to 2MB for a 300dpi color page.

In the intermediate representation, the foreground mask is a composed of a list of run-length encoded marks and a list of records specifying how to reconstruct the foreground image by blitting specified marks at specified locations with specified colors. These lists match the structure of the JB2 codec and are easily derived from the foreground mask chunk.

The intermediate representation of the background layer was in fact the major reason behind the design the IW44 wavelet codec. There are three collections of quantized wavelet coefficients (one for the luminance, two for the chrominances). Each collection is organized as a grid of 32x32 blocks of wavelet coefficients. Blocks themselves are stored as a sparse array containing only the non-zero coefficients. Memory requirements are small because typical images contain much fewer non-zero quantized wavelet coefficients than pixels.

The sparse arrays are structured as trees. The top level node contains four pointers to second level nodes containing sixteen pointers to leaf nodes containing sixteen wavelet coefficients. This tree structure mimics the dependency structure of the IW44 block-based bitstream. All toplevel pointers are initially null. Whenever new background data is available, the decoder simply allocates new nodes in the sparse array and/or updates the coefficients in the leaf nodes. Therefore the sparse array can be updated "in place" with minimal overhead.

The rendering of a sub-image involves the following steps: marking the 32x32 blocks and the coefficients therein whose corresponding wavelet overlaps the sub-image to be displayed; performing an inverse wavelet transform with these coefficients; and finally convert the YCrCb pixels thereby obtained into RGB, suitable for screen display. The inverse wavelet transform is performed by taking into account progressively higher frequency coefficients, stopping early when the coefficients code details that are invisible at the current zoom settings.

Since the readability of the text on computer displays is paramount to the usefulness of DjVu, special care was devoted to designing a fast and accurate image scaler with good anti-aliasing that preserves text readability at low zoom factors.

3.4. On-Demand Loading, Pre-Fetching and Caching

DjVu provides two formats called "bundled" and "indirect" under which a document can be stored. In the bundled format, the whole document is stored in a single file. Since the file is loaded sequentially, users cannot jump to the last pages until the whole document has been downloaded. This format is convenient for accessing relatively short documents (up to a few 100KB), and for manipulating them as a single entity (e.g. for sending them by email). In the indirect format, each page is in a separate file, and the document master file merely contains pointers to those page files in the form of relative URLs or filenames. This format is more appropriate for large documents with many pages because it allows user to jump randomly to any page without waiting for all the pages to load and without requiring a special web server. Most component files of an indirect DjVu document are individual pages, but some may be chunks shared by multiple pages, such as JB2 shape libraries, or page thumbnails. When a page is first displayed, all the component files needed for that page are downloaded on-demand and cached. To minimize the delay in flipping pages, the viewer always pre-fetches and pre-decodes the next page. This ensures nearly-instantaneous page flipping.

To minimize delays, the DjVu plug-in uses three caches. The first cache is the web browser cache, which avoids multiple loading of often-accessed components. The second cache contains the partially decoded image of recently displayed pages and the next page. This allows fast page flipping back and forth. The last cache contains recently displayed zones of the current page in a fully rendered form. This minimizes the computing load when panning around a large image.

We briefly considered the option of breaking down images into individually loadable "tiles", but decided against it because each TCP connection and HTTP query involves multiple round-trips to the web server, with associated delays that would have slowed down panning operations. Minimizing the number of server round-trips does much to accelerate the overall process.

3.5. Integration of DjVu content with Web Content

Documents are rarely provided in isolation. A Document image format would be difficult to use without built-in features that allow interaction with other Web content (HTML, JavaScript, etc). Two prominent examples of such features are hyper-linking, and indexing/searching. DjVu allows hyper-links and other annotation to be embedded within pages through the "annotation" chunk. Hyper-links are rectangular or polygonal areas described by a series of page coordinates associated with a URL.

The behavior of the DjVu plug-in can be controlled by the Web designer in three different ways: directives included in the annotation chunk within the page, CGI-style URL arguments appended to the URL of the document, and argument passed to an OBJECT (or EMBED) HTML tag if the document is embedded in an HTML page. A full discussion of this is beyond the scope of this paper, but the following example illustrates how CGI-style arguments can conveniently control the DjVu plug-in: A hyper-link to a particular page (say page 10) of a DjVu document (say `djvu/antics/cuisine/index.djvu` on `djvuzone.org`) can be created using the following URL: `http://www.djvuzone.org/djvu/antics/cuisine/index.djvu?djvopts&page=10`

The argument after the question mark is ignored by the web server, but is interpreted by the DjVu plug-in receiving the document to mean "jump to page 10". Zoom factors, hyper-links, or highlighted areas on the document can be defined similarly. This allows the creation of dynamic content that is perfectly integrated with the more traditional web content.

Indexing and searching DjVu documents can be done with the help of OCR. DjVu pages can contain a "hidden text" chunk which includes the recognized text as well as the coordinates of each word on the page in a compressed form. When this chunk is present, words or phrases in a DjVu document can be searched from within the DjVu plug-in. The text chunk is efficiently compressed using BZZ, a Burrows-Wheeler based general purpose compression engine included as part of the DjVu software.

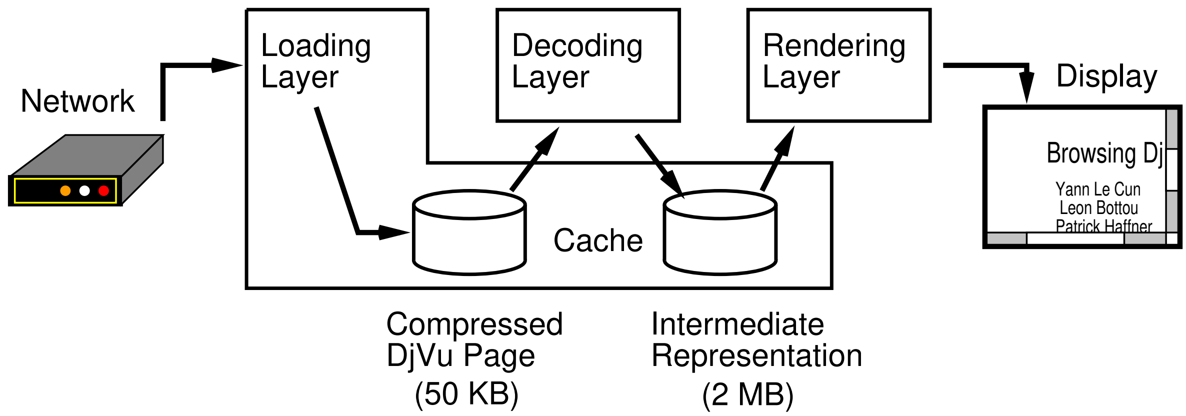

Figure 1. The three layers of the DjVu viewer: the loading layer loads and caches document components, the decoding layer constructs the intermediate representations, and the rendering layer computes the image pixels.

Server-based indexing of document collections are easily performed when the OCR'd text is available. An example of a searchable DjVu digital library is available at `http:///www.djvuzone.org/djvu`.

4. IMPLEMENTATION

This section provides a brief overview of the DjVu viewer implementations. The preferred mode of access for DjVu document is through a web browser. The DjVu viewer is implemented as a plug-in, compatible with the major web browsers on Windows, Mac and Unix (including Linux). To ensure portability, the Unix version is implemented as a stand-alone viewer that integrates itself in the browser window and communicates with it through pipes and through a small plug-in module. In all cases the plug-in provides a seamless interface between DjVu content and regular Web content.

This overview focuses on the decoding and displaying functions of the DjVu viewer. The viewer also implements a number of mundane features such as displaying thumbnails, processing hyper-links, saving or printing entire documents, or searching text in annotated DjVu images. These functions are implemented using the same basic structure as the decoding and displaying functions.

4.1. Multi-thread architecture

The DjVu viewer architecture revolves around the intermediate image representations introduced in section 3.3. The first stage of the decoding process consists in processing incoming data and creating or updating the intermediate image representations. The second stage consists in rendering the pixels on-the-fly in response to user interaction. Both stages are driven by asynchronous event sources, either network events or user interaction events. These stages are naturally implemented as separate threads.

The DjVu viewer is composed of three major components as illustrated in figure 1:

- **Loading Layer** — The loading layer loads and caches document components on-demand.
- **Decoding Layer** — The decoding layer is composed of multiple threads that construct the intermediate (partially decoded) representation from compressed data provided the loading layer.
- **Rendering Layer** — The rendering layer computes the pixels of the image being displayed on the user's screen from the intermediate representation. In order to implement progressive decoding, the rendering layer receives notifications from the decoding layer threads indicating that sufficient data has been processed to justify rendering an updated version of the image.

Because the partially decoded image structures are manipulated by a large number of threads, a strict synchronization policy would cause severe locking contention and eventually reduce the overall performance. For this reason, the renderer is designed to operate on partially decoded image structure even while they are being updated by a decoding thread, thus reducing the need for synchronization. As it turns out, one of the most difficult software issues in this system was to properly shut down all the threads and deallocate the appropriate structures, and delete partially loaded data chunks when the user hits the browser's "stop" or "back" buttons.

The DjVu viewer contains a large number of threads that allocate objects and simultaneously operate on them. Given the complexity of the situation, the memory management is handled by a thread-safe garbage collector. This garbage collector is based on reference counts and is relatively transparent to the programmer. Objects communicate with each other through notification channels. The garbage collector and the notification system cooperate in order to ensure that routes are atomically destroyed when one of the objects they connect is destroyed.

5. CONCLUSION

DjVu is a new compression technique for color document images that fills the gap between the world of paper and the world of bits. It provides high compression ratios (5 to 10 times higher than either G4 or JPEG)and allows scanned document to be easily published on the Internet.

DjVu can also be considered as an enabling technology for document analysis techniques, imaging and rendering techniques, and efficient use of the network bandwidth. High compression ratios are achieved using complex image segmentation and pattern matching algorithms. Smooth browsing of large document images is achieved using sophisticated intermediate image representations. Fast and efficient access to the pages of a document are achieved using multiple caching and prefetching strategies.

REFERENCES

1. P. G. Howard, "Text image compression using soft pattern matching," *Computer Journal* **40(2/3)**, pp. 146–156, 1997.
2. L. Bottou, P. Haffner, P. G. Howard, P. Simard, Y. Bengio, and Y. LeCun, "High quality document image compression with DjVu," *Journal of Electronic Imaging* **7**(3), pp. 410–425, 1998.
3. 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**, pp. 1174–1179, November 1974.
4. L. Bottou, P. G. Howard, and Y. Bengio, "The Z-coder adaptive binary coder," in *Proceedings of IEEE Data Compression Conference*, pp. 13–22, (Snowbird, UT), 1998.
5. S. Inglis, *Lossless Document Image Compression*. PhD thesis, University of Waikato, March 1999.
6. W. Sweldens, "The lifting scheme: A custom-design construction of biorthogonal wavelets," *Journal of Applied Computing and Harmonic Analysis* **3**, pp. 186–200, 1996.
7. J. M. Shapiro, "Embedded image coding using zerotrees of wavelets coefficients," *IEEE Transactions on Signal Processing* **41**, pp. 3445–3462, December 1993.
8. A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology* **6**, pp. 243–250, June 1996.
9. D. Taubman, "High performance scalable image compression with ebcot.," *IEEE Transactions on Image Processing* **9**, pp. 1158–1170, July 2000.
10. M. Antonini, M. Barlaud, P. Mathieu, and D. I., "Image coding using wavelet transform," *IEEE Transactions on Image Processing* **1**, pp. 205–220, 1992.
11. J. Rissanen, "Stochastic complexity and modeling," *Annals of Statistics* **14**, pp. 1080–1100, 1986.
12. W. N. J. Sheinvald, B. Dom and D. Steele, "Unsupervised image segmentation using the minimum description length principle," in *Proceedings of ICPR 92*, 1992.
13. J. Morrison, "EA IFF 85 : Standard for interchange format files," in *Amiga ROM Kernel Reference Manual: Devices (3rd edition)*, Addison-Wesley, 1991.
14. M. J. Gormish, D. Lee, and M. W. Marcelling, "JPEG-2000: Overview, architecture and applications," in *Proceedings of the IEEE International Conference on Image Processing 2000*, vol. II, pp. 29–32, 2000.