High quality document image compression with DjVu

Leon Bottou

Joint work with
Patrick Haffner
Paul Howard

Yoshua Bengio
Yann Le Cun
Motivation

Much of the world’s knowledge is still paper.

Converting to a web format is expensive and imperfect. What about image compression techniques?

TIFF/MMR/G4 falls short for bitonal

50–100K per page at 300 dpi

Continuous-tone techniques are inadequate (e.g. jpeg).

Large files (150K–1M), fuzzy text, large memory requirements, no document structure.

PDF for scanned documents

is merely an encapsulation for TIFF/G4 or JPEG

New/emerging standards

are getting there, but are not tuned for web viewing, and need to work on efficiency (JBIG2, JPEG-2K, T.44)
What is DjVu

A compression technique, a file format, and a software platform for distribution documents on the web (scanned or digitally produced, in color, gray, or b&w.)

Scanned bitonal 300 dpi : 5–40K per page
  ( 3–10 times better than TIFF/G4 )
Scanned color 300 dpi : 30–100K per page
  ( 5–10 times better than JPEG or PDF )
Photos : 2dB better than JPEG
  ( quality similar to JPEG–2000, but faster )
Digital documents 300 dpi : half the size of ps.gz or pdf.
High quality text, good enough for printing.
Progressive decoder/renderer (text appears first.)
Seamless panning and zooming.
Good integration with web browsers.
Available today ( already through its 3rd revision)
1– Examples

2– Entropy coding with the Z–Coder

3– Bitonal images (DjVuBitonal, JB2)

4– Continuous tone images (DjVuPhoto, IW44)

5– Color documents (DjVuDocument)

6– Foreground / background segmentation
   – digital documents
   – scanned documents

7– Efficient browsing
DjVu : Z–Coder : Entropy Coding

- short codes for high probability events
- long codes for low probability events

Huffman: \[ \langle \text{code\_length} \rangle = \lceil -\log_2 (p) \rceil \]

Fixed estimate of probability distribution

BAD

Independent events

Encoder

Bits
- use context to lower entropy

**Diagram:**

- **Context** (previous events) → Estimate of Conditional Distribution $P(\text{Event}|\text{Context})$
- **Events** → **Encoder**
- **Bits**
- **Context** → **P(\text{Event}|\text{Context})** → **Decoder**
- **Events**
DjVu: Z-Coder: Adaptive Entropy Coding

- learn the conditional distributions
- encoder and decoder remain in sync.
- **Arithmetic coders**
  
  The string of code bits is viewed as a number in [0,1)

  Events are coded by maintaining an interval of possible code strings

- **Binary arithmetic coders** *(efficient – adaptive)*

  Q–Coder [ibm] MQ–Coder [+mel] QM–Coder [+lucent]
DjVu : Z–Coder : Golomb

– Golomb Coder

Assume 1 is more probable than 0.
Chances are that one codes words like 11111110

Example: Golomb coding with M=4

<table>
<thead>
<tr>
<th>Events</th>
<th>Bits</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1</td>
<td>1 1 1 1</td>
<td>–1 group of 4 ones</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>0 1 0</td>
<td>–1 zero preceded by two ones.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Events</th>
<th>P(1)/P(0) large</th>
<th>P(1)/P(0) near 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>111111111111111111110</td>
<td>1110</td>
<td></td>
</tr>
<tr>
<td>100010</td>
<td>&lt; better</td>
<td></td>
</tr>
<tr>
<td>111111111100</td>
<td></td>
<td>00011</td>
</tr>
<tr>
<td>M=16</td>
<td></td>
<td>101 &lt; better</td>
</tr>
<tr>
<td>M=2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- Arithmetic Interpretation of Golomb Coders
- Z–Coder

Each 1 takes an arbitrary code bit fraction that depends on its estimated probability.
Z–Coder Mathematics

Analytic relation determines best bit fraction for a given probability.

Provably within 0.25% of Shannon bound.

Z–Coder Speed

Coding one (typical) event:

1 ADD + 1 COMPARE
Z-Coder Adaptation

Finite state machine.

Each context is represented by a small integer (state).

Coding 1 or 0 causes transitions in a state machine.

Adaptation noise increases bitrate by about 3%.
DjVu : Z–Coder : Conclusion

- Z–Coder is a binary adaptive entropic coder.  
  It supports multiple contexts.  
  It learns probabilities on the fly.

- Z–Coder typically performs 1 ADD + 1 COMPARE per decoded event

- Compressed bitrates are 3–5% above the theoretical lower bound (Shannon)
Documents contain multiple repetitions of nearly identical shapes (e.g. characters)

**JB2 data** = Sequence of encoded records that
– describe shapes
– display selected shapes at specified positions

Cross-Coding: Coding shapes by comparing them with a previously coded shape.

Soft Pattern Matching: Small differences between shapes can be safely ignored.

ZP-Coder: Fast adaptive arithmetic coder that squeezes JB2 data (almost) to the theoretical limit.

Shared Shapes: Shape description can be shared by several pages of a document.
DjVu : DjVuBitonal : Schema (a.k.a. JB2)

- **TEXT**
  - Page (bitonal)

- **connected component analysis**

- **shape cleaning filtering**

- **shape clustering**

- **Bits**
  - z-coding of shapes and positions
DjVu : DjVuBitonal : Direct

1024 different contexts

$Z$–Coder learns $P(\text{current pixel} | \text{known pixels})$
DjVu: DjVuBitonal: More Coding

Cross Coding

Current shape

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Similar shape

<table>
<thead>
<tr>
<th></th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Position Coding

brown fox

\[ \Delta x \]

\[ \Delta y \]
DjVu : DjVuBitonal : Results vs TIFF/G4

File size (per page)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>TIFF/G4</th>
<th>DjVuBitonal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon patent</td>
<td>(20.8)</td>
<td>(166.5)</td>
</tr>
<tr>
<td>19 pages 300 dpi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NIPS 10</td>
<td>(16.7)</td>
<td>(115.9)</td>
</tr>
<tr>
<td>1090 pages 400 dpi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Snowbird</td>
<td>(18.5)</td>
<td>(188.2)</td>
</tr>
<tr>
<td>133 pages 300 dpi</td>
<td></td>
<td>(42)</td>
</tr>
</tbody>
</table>

Notes:
PDF encodes scanned bitonal documents using TIFF/G4 compression.

More...
DjVu : DjVuBitonal : Results vs JBIG2

File size

- CCITT F4 dense text 200 dpi: JBIG2 32.5, DjVuBitonal 46.4
- CCITT F7 japanese text 200 dpi: JBIG2 22.8, DjVuBitonal 46.4
- CCITT F10 w/dithered images 200 dpi: JBIG2 (10), DjVuBitonal (10)

Total time (secs)

- Comp.: JBIG2 6, DjVuBitonal 6
- Decomp.: JBIG2 2, DjVuBitonal 2

Notes:
All files contain a single page (no multipage encoding)
- JBIG2 (CSM,MQ), PIII–500, (ImagePower encoder)
- DjVuBitonal, PII–400
DjVu : DjVuBitonal : Conclusion

- Lossy JB2 is typically 3–10 times smaller than MMR/G4 on text and 2–4 times smaller on line art.

- Multipage compression with shared shape dictionary between pages boosts compression by 1.4 – 2.0.

- Lossy JB2 is significantly smaller than arithmetically coded lossy JBIG2 on mostly textual images, and about the same on line art and halftones.

- JB2 compression / decompression is faster than JBIG2.

- Lossless JB2 compression is 2 times smaller than MMR/G4 and 1.4 times smaller than JBIG1 [Inglis, 1999]
Wavelet Image Compression with specific features:

- **Fast lifting transform (without multiplications)**
  Enables fast panning in viewer.

- **Progressive images**
  Browser displays successive refinements.

- **Decoder has very small memory footprint**
  Even for large progressive images.

- **One can specify “don’t care” pixels while encoding**
  - non-rectangular images
  - partially masked images
DjVu : DjVuPhoto : Lifting (a.k.a. IW44)

Standard wavelet filter

Lifting wavelet filter

Even

Odd

Easily reversible → integer transforms → faster implementations

Integer transforms usually confined to lossless compression.

IW44 uses fixed point lifting for lossy image compression.
DjVu : DjVuPhoto : Results

**SNR (dB)**

- **Lena512 0.25 bpp**
- **Lena512 0.5 bpp**
- **Lena512 1.0 bpp**

**Total Decoding Time (s)**

- **JPEG**
- **JPEG2K (9–7)**
- **JPEG2K (5,3)**
- **DjVuPhoto**

DjVuPhoto

- 2dB better than JPEG
- Between JPEG2K (5,3) and (9–7) transforms
- Decodes 3 times faster than JPEG–2000

JPEG–2000 results using JasPer Software (9–7 transform)
Decoding times measured on a R10K–195
We do not want to spend bits on pixels located below foreground objects.

The coefficients of these wavelets are set to zero.

The coefficients of these wavelets are modified.

The coefficients of these wavelets are left unmodified.
An image is a vector
A wavelet transform is a linear operator
(1) over–relaxation

(2) each scale is treated separately

Overall cost = 3 x ordinary wavelet
<table>
<thead>
<tr>
<th>Image</th>
<th>Size</th>
<th>% Masked</th>
<th>Compressed File Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>reg.</td>
</tr>
<tr>
<td>hobby1</td>
<td>825x1074</td>
<td>19%</td>
<td>131K</td>
</tr>
<tr>
<td>hobby2</td>
<td>825x1074</td>
<td>19%</td>
<td>140K</td>
</tr>
<tr>
<td>metric</td>
<td>744x1074</td>
<td>26%</td>
<td>170K</td>
</tr>
<tr>
<td>missel</td>
<td>610x429</td>
<td>40%</td>
<td>64K</td>
</tr>
<tr>
<td>plugin</td>
<td>757x1035</td>
<td>32%</td>
<td>189K</td>
</tr>
</tbody>
</table>
• State-of-the-art wavelet compression (i.e. far superior to jpeg)

• Fast lifting wavelet transform (even for lossy compression)

• Unique masking technique avoids wasting bits on masked pixels.
DjVu : DjVuDocument

Page Image = Background + Foreground

- Subsampled DjVuPhoto
- Transparency Mask DjVuBitonal

Colors encoded either:
- as one solid color per shape.
- as a lores color image

Size of average DjVu 300 dpi color page = Size of average web page (html + images)
DjVu : DjVuDocument : Schema

Image (color)

Big Title
Subtitle

Background

segment

sub-sample

IW44

IFF file

JB2

foreground
Segmentation

- **For scanned documents**
  
  We start with a high resolution RGB image.
  
  Example: PPM image  
  (22MB for a 8’x11’ page, 300 dpi, 24 bpp)


- **For digital documents**
  
  We start with an electronical document.
  
  Application level: XML/HTML, MSOffice, ...
  Printer level: PDF, PostScript

  Apparently easier. Expectations are much higher.
Printing language provides a document decomposition.
Superposed objects can be monochromatic (e.g. text) polychromatic (e.g. pictures)

Good start ... but simple solutions are not sufficient.
### Segmentation: Digital: Rate/Distortion Criterion

[Bottou–Haffner–LeCun 2001]

<table>
<thead>
<tr>
<th>Background or foreground?</th>
<th>Coding as foreground (with jb2)</th>
<th>Coding as background (with iw44)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimate of the coding cost (bits)</td>
<td>$P_{\text{clipped}}$</td>
<td>$P_{\text{background}} \times \text{AvgColorDiff}$</td>
</tr>
<tr>
<td>Estimate of the distortion cost (bits)</td>
<td>0</td>
<td>$P_{\text{background}} \times \text{AvgColorDiff}$</td>
</tr>
</tbody>
</table>
For all monochromatic objects in reverse drawing order

Process occlusions
- by objects classified as foreground objects
- by objects classified as background objects

If $P_{\text{background}} \times \text{AvgColorDiff} > P_{\text{clipped}} \times \text{Threshold}$:
- Foreground

If not:
- Background
DjVuDigital (ps2djvu)

Aladdin GhostScript driver “djvusep”
Back-end encoders “msepdjvu”
http://djvuserver.research.att.com/~leonb/ps2djvu

Example

PostScript
Encoded from PostScript
Encoded from RGB image
## Segmentation: Digital: Results

<table>
<thead>
<tr>
<th>Document</th>
<th>Type</th>
<th>Pages</th>
<th>ps.gz / pdf</th>
<th>ps2djvu</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask.ps.gz</td>
<td>latex</td>
<td>10</td>
<td>400K</td>
<td>78K</td>
</tr>
<tr>
<td>paper2web.pdf</td>
<td>book</td>
<td>327</td>
<td>4230K</td>
<td>2925K</td>
</tr>
<tr>
<td>sgi.pdf</td>
<td>flyer</td>
<td>4</td>
<td>484K</td>
<td>106K</td>
</tr>
<tr>
<td>stanford.pdf</td>
<td>map</td>
<td>1</td>
<td>412K</td>
<td>170K</td>
</tr>
</tbody>
</table>

Total time with ps2djvu: \(23 + 1230 + 27 + 30\) seconds

Total time before*: \(383 + 7900 + 201 + 123\) seconds

* rendering a 300dpi rgb image with GhostScript and converting this image to DjVu
Segmentation : Scanned : Schema

Big Title
Subtitle

Image (color)

make overlapping blocks

generate candidate foreground objects

find two dominant colors

filter candidates using MDL criterion
**Browsing : Goal**

**DjVu browsing goal:**
<< Replicating the “paper” experience. >>

<table>
<thead>
<tr>
<th>Action</th>
<th>Equivalent for Paper Documents</th>
<th>Acceptable Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zooming/Panning</td>
<td>Move the eyes</td>
<td>Immediate</td>
</tr>
<tr>
<td>Next/Prev Page</td>
<td>Turn a page</td>
<td>&lt; 1 second</td>
</tr>
<tr>
<td>Go To Page #</td>
<td>Find a page</td>
<td>&lt; 3 seconds</td>
</tr>
</tbody>
</table>
Revisiting a well known trick: DjVu Progressive Decoding.

Chunk order determines what information comes first.

<table>
<thead>
<tr>
<th>Text &amp; lineart</th>
<th>fgc</th>
<th>bg1</th>
<th>bg2</th>
<th>bg3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b&amp;w text</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Color text</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text+background</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Background</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Background sharpens twice.

<table>
<thead>
<tr>
<th>Text &amp; lineart</th>
<th>fgc</th>
<th>bg1</th>
<th>bg2</th>
<th>bg3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b&amp;w text</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Color text</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text+background</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Background sharpens four times.

<table>
<thead>
<tr>
<th>bg1</th>
<th>fgc</th>
<th>Text &amp; lineart</th>
<th>bg2</th>
<th>bg3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b&amp;w text</td>
<td></td>
<td>Color text over coarse background</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Background sharpens twice</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Browsing: Modeless GUI**

Save a split second by minimizing delay between

**User intention and user action!**

---

**Modal GUI**

BAD

---

**Modeless GUI**

GOOD

---

<table>
<thead>
<tr>
<th>Select ...</th>
<th>... then act</th>
</tr>
</thead>
<tbody>
<tr>
<td>Panning mode</td>
<td>Point</td>
</tr>
<tr>
<td>Zooming mode</td>
<td>Drag</td>
</tr>
<tr>
<td>Selecting mode</td>
<td>Release</td>
</tr>
</tbody>
</table>

---

*Single action*

Drag whole image
Click zoom button
Click hyperlink
Browsing : Panning : Bad

Big memory requirement --> swapping --> slow panning.
Progressive display --> full decoder run after each chunk.

Decoder runs only on exposed parts --> image tiling:
large tiles --> slow panning
small tiles --> poor compression
Browsing : Panning : Intermediate Representation

[Bottou & al. 2000]

- Predecoding time overlaps data transfer time.
- Predecoder incrementally builds the intermediate representation whenever a chunk arrives.
- Intermediate memory representation is optimized for rendering any portion of the image at any resolution.

GOOD

--- Fast panning.
--- CPU efficient progressive display
Intermediate Image Representation

**DjVuBitonal (JB2)**
- Run-length-encoded shapes
- Array of (shapeno, position) records

**DjVuPhoto (IW44)**
- Blocks of 32x32 quantized wavelet coefficients.
  - Each block is represented by a sparse array holding only the non zero coefficients.
  - Sparse array structured as 2-level 4x16x16 tree that matches the IW44 bitstream.
Bundled

- Initial request eventually downloads the whole document
- Efficient random access requires a byte-server.

Convenient for small documents and email attachments

Indirect

- Fine grain web-serving without a byte-server.
- Supports sophisticated caching and prefetching.
- Initial web request only downloads the directory.

Follows the Web metaphor!
Browsing: Multipage: Caching etc.

Network → (pre)Fetch → Compressed data cache → (pre)Decode → Intermediate data cache → Render → Display

User

<table>
<thead>
<tr>
<th>User</th>
<th>click...</th>
<th>read...</th>
<th>flip'n read...</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prefetch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>predecode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>render</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Annotation Chunks

Annotations can define hyperlinks
Annotations can hilite specified areas
Servers can generate annotations on the fly

Searchable Hidden Text

Generated by OCR or otherwise

CGI–Style URLs

http://localhost/stanley.djvu?djvuopts&page=39
Browsing: Implementations

Lightweight browser plugin (Lizardtech).

Autoinstall size for version 3.0 (win32) < 550 KB
Multiplatform: Win, Mac, Linux, Solaris
http://www.djvu.com/download.html

DjVu Reference Implementation

Free source code software (GPL'd)
http://www.djvuzone.org/open
Conclusion

DjVu enables hires color scan-to-web

Sophisticated viewer provides
  – unmatched user experience
  – solid web integration

Just scan it!