

Image Compression

- Run-length encoding
- Bi-level images
- Lossless grayscale
- Lossless color

Run-Length Encoding

0	0	0	0	0
2	2	2	2	2
4	6	6	6	0
0	0	6	6	6

RLE = sequence of pairs (*count, value*)

5*0, 5*2, 1*4, 3*6, 3*0, 3*6

BinHex 4.0 Format

- used for “safe” file transfer on Mac (esp. binary files)
begins with **comment & header**, includes filename, version, CRC,...
(This file must be converted with BinHex 4.0)
- **use RLE** for initial conversion, hex **90** is RLE marker
runlengths 3-255 are compressed, runlength=0 converts marker

<i>source bytes</i>	<i>packed bytes</i>
00 11 22 33	00 11 22 33
11 22 22 22 22 23	11 22 90 04 23
11 22 90 33	11 22 90 00 33
- **output sequence of ascii characters:**
 - use bytes as stream of 6-bit blocks, used as ptrs into BinHex tbl
! " # \$ % & ' () * + , - 0 1 2 3 4 5 6 8 9 @ A B C D E F G H I
J K L M N P Q R S T U V X Y Z [' a b c d e f h i j k l m p q r (note missing 70Wgno)
 - output chars in groups of 64 (except last)
each group preceded and followed by pair of colons

Run-Length Encoding – Predictive Coding

0	0	0	0	0
2	2	2	2	2
4	6	6	6	0
0	0	6	6	6

- sample presented as error term:
 $value = sample - prediction$
- Differential Encoding (DE):
prediction = previous sample
5*0, 2, 4*0, 2*2, 2*0, -6, 2*0, 6, 2*0
- Linear Predictive Coding (LPC)
uses two previous samples

Run-Length Encoding – Predictive Coding

0	0	0	0	0
2	2	2	2	2
4	6	6	6	0
0	0	6	6	6

- **PDQ** – Predictive Differential Quantizer
 - encode differences between RLE of successive scan lines
 - special codes for introducing/merging runs

RLE	PDQ
(5,0)	(5,0)
(5,2)	(0,2)
(1,4)(3,6)(1,0)	N(1,4)(-2,4)N(1,0)
(2,0)(3,6)	(1,-4)(0,0)M

Bi-Level Images

- Facsimile compression
- Quadtrees
- Space-filling curves
- JBIG

Bi-Level Images – Facsimile Compression

- lossless
- based on RLE (plus modified Huffman)
- lines converted into pels (Picture **EL**ements)
 - 8.05 pels/mm (205 pels/in), 8.2" wide \Rightarrow 1664 pels/line

<i>mode</i>	<i>scan lines per mm</i>
standard	3.85
fine	7.7
very fine	15.4

Facsimile Compression

Two coding schemes

- 1-dimensional coding
 - each line RLE separately (alternate colors)
 - assumes first run is a white
- 2-dimensional coding
 - encode boundaries of runs (**not** runlengths)
 - relative to previous line (**not** absolute position)

Facsimile Compression – Gp 3

- Group 3 – for public switched telephone network
- **one-dimensional coding**, run-length encoding
 - convert run-length r into $64m + t$
codes for $t = 0-63$ (t_{ermination}), $m = 1-27$ (m_{ake-up})
encode (if-needed) make-up code then termination code
 - each line has: 1 white pel before, EOL code (11 zeros & 1 one) after
if # of bits/line must be $8k$, then insert 0's before EOL
 - bit after EOL is 0 \Rightarrow next line uses 2-d coding
(only after 1-d coding) for ≤ 1 (3 for fine) lines
 - each page: preceded by 1 EOL, ended by 6 EOLs
- **predetermined coding** of b/w runlengths
 - uses **modified Huffman** encoding of expected freqs
(length constrained + EOL code can't appear as substring)
 - most common runlengths: 2-4 black, 2-7 white

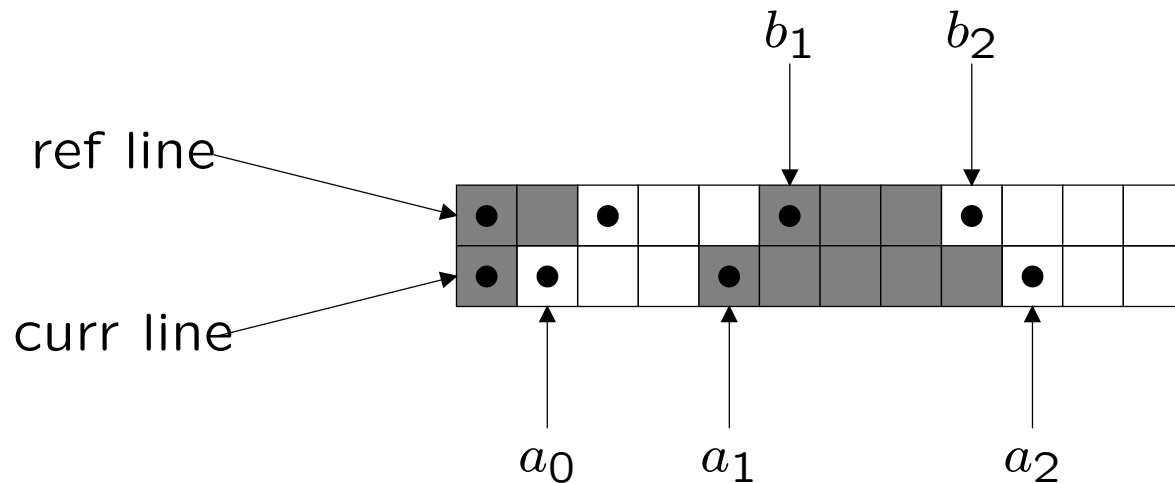
Modified Huffman – Termination Codes

run	White	Black	run	White	Black	run	White	Black
0	00110101	0000110111	22	0000011	00000110111	43	00101100	000011011011
1	000111	010	23	0000100	00000101000	44	00101101	000001010100
2	0111	11	24	0101000	00000010111	45	00000100	000001010101
3	1000	10	25	0101011	00000011000	46	00000101	000001010110
4	1011	011	26	0010011	000011001010	47	00001010	000001010111
5	1100	0011	27	0100100	000011001011	48	00001011	000001100100
6	1110	0010	28	0011000	000011001100	49	01010010	000001100101
7	1111	00011	29	00000010	000011001101	50	01010011	000001010010
8	10011	000101	30	00000011	000001101000	51	01010100	000001010011
9	10100	000100	31	00011010	000001101001	52	01010101	000000100100
10	00111	0000100	32	00011011	000001101010	53	00100100	000000110111
11	01000	0000101	33	00010010	000001101011	54	00100101	000000111000
12	001000	0000111	34	00010011	000011010010	55	01011000	000000100111
13	000011	00000100	35	00010100	000011010011	56	01011001	000000101000
14	110100	00000111	36	00010101	000011010100	57	01011010	000001011000
15	110101	000011000	37	00010110	000011010101	58	01011011	000001011001
16	101010	0000010111	38	00010111	000011010110	59	01001010	000000101011
17	101011	0000011000	39	00101000	000011010111	60	01001011	000000101100
18	0100111	0000001000	40	00101001	000001101100	61	00110010	000001011010
19	0001100	00001100111	41	00101010	000001101101	62	00110011	000001100110
20	0001000	00001101000	42	00101011	000011011010	63	00110100	000001100111
21	0010111	00001101100						

Facsimile Compression – Gp 4

- Group 4 – for digital network (ISDN)
also called MMR (Modified Modified Read)
- two dimensional coding only (never use 1-d)
compare current (*scan*) line
with predecessor (*ref*) line
- initial reference line is all white
- each line assumed to start with white pel
- relative positions of next runs
(on ref & curr lines) determine encoding mode

Facsimile Compression – 2-D Coding



transition cells are bulleted

a_0 = last cell already encoded on curr line
not necessarily start or end of run

a_1 = first transition pixel after a_0

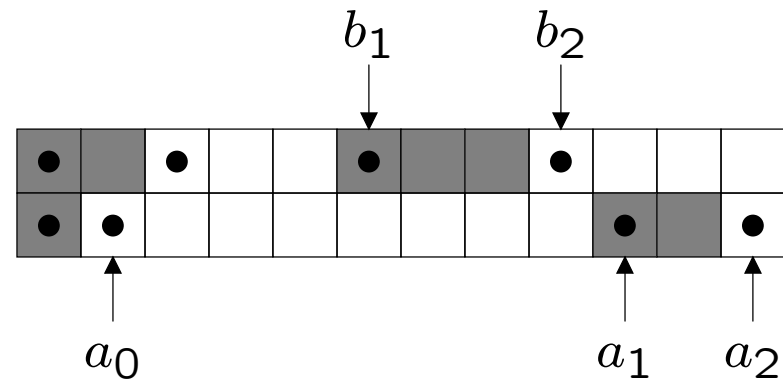
a_2 = second transition pixel after a_0

b_1 = first transition pixel (opp. color of a_0)
on ref line to right of a_0

b_2 = next transition pixel to right of b_1

Three modes – Pass, Vertical, Horizontal

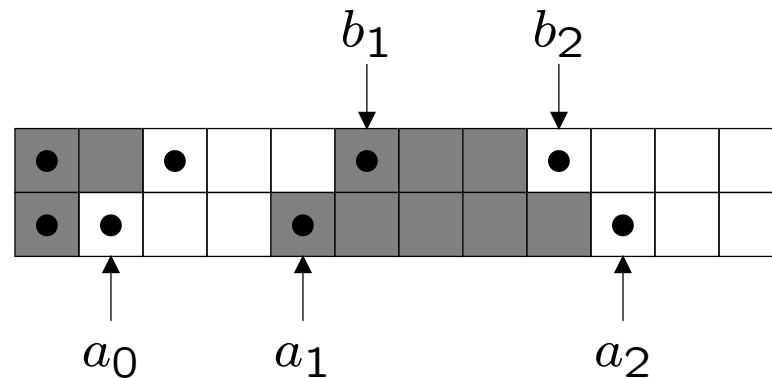
Facsimile Compression – 2-D Coding



PASS mode: b_2 before a_1

- can conclude that all pixels are the same in the new line thru position below b_2
- transmit 0001
- set $a_0 \leftarrow$ cell below b_2

Facsimile Compression – 2-D Coding



VERTICAL mode: $a_1 \leq b_2$ and $|dist(a_1, b_1)| \leq 3$

- transmit Huffman code for $dist(a_1, b_1)$

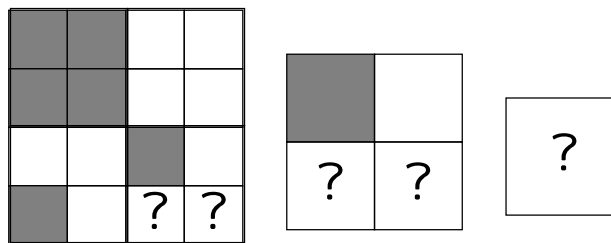
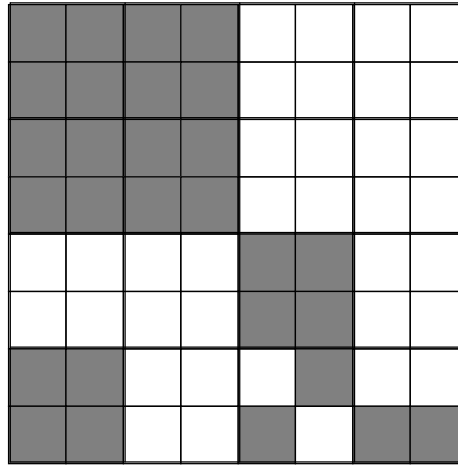
$a_1 - b_1$	code	$a_1 - b_1$	code
0	1	-1	011
1	010	-2	000011
2	000010	-3	0000011
3	0000010		

- set $a_0 \leftarrow a_1$

HORIZONTAL mode: $a_1 \leq b_2$ and $|d(a_1, b_1)| > 3$

- transmit 001
- transmit Huffman codes for $dist(a_0, a_1), dist(a_1, a_2)$
- set $a_0 \leftarrow a_2$

Bi-Level Images – Quadrees



- ? = mixed colors
- header indicates # of levels, then 1-bit indicates expansion, stored in sequence $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}$

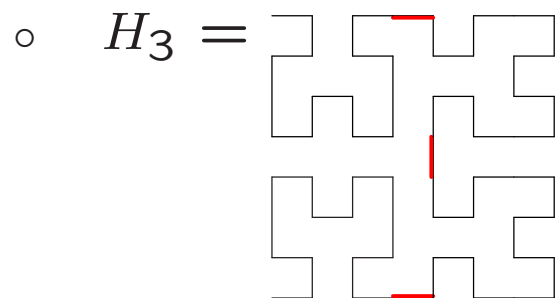
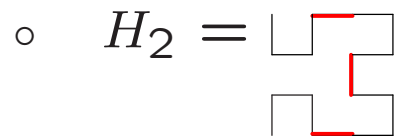
⇒ 1 01,00,1 00 00 01 00,1 01 00 10110 10011

Bi-Level Images – Space-Filling Curves

- contiguous space-filling traversal such as Hilbert curve

- $H_i =$ Hilbert curve of order i

- $H_i =$ connect 4 (rotated) instances of H_{i-1}
- $H_0 =$ single dot
- $H_1 =$ □



- use order just sufficient for pixel dimensions
 - analogous use to quadtrees, or can use RLE

Hilbert Curve Traversal

- level i Hilbert curve is $2^i \times 2^i$ matrix
 - coordinates have $2i$ bits
 - each bit pair determines quadrant
 - need orientation
- can transform node number \rightarrow coordinates by using tables:

bits	T1		T2		T3		T4	
	xy	tbl	xy	tbl	xy	tbl	xy	tbl
00	00	2	00	1	11	4	11	3
01	10	1	01	2	01	3	10	4
10	11	1	11	2	00	3	00	4
11	01	4	10	3	10	2	01	1

Example: node 109 [256] = 01101101

start in T1 $\Rightarrow x, y$ start with (1,0), goto T1

then T1 $\Rightarrow x, y$ next have (1,1), goto T1

then T1 $\Rightarrow x, y$ have (0,1), goto T4

then T4 $\Rightarrow x, y$ have (1,0)

thus $x = 1101 = 13$, $y = 0110 = 6$

Hilbert Curve Traversal

- can transform coordinates \rightarrow node number by using tables:

	T1		T2		T3		T4	
bits	<i>xy</i>	tbl	<i>xy</i>	tbl	<i>xy</i>	tbl	<i>xy</i>	tbl
00	00	2	00	1	10	3	10	4
01	11	4	01	2	01	3	11	1
10	01	1	11	3	11	2	01	4
11	10	1	10	2	00	4	00	3

Bi-Level Images – JBIG

- Joint Bi-level Image processing Group
- parameter P = source bpp
 - for $P > 6$, better to use other algorithms
 - compress images on a bitplane basis
 - using Gray coding on pixel values
 - limits bit changes between adjacent pixel values
- high-level view
 - more efficient compression than Fax Gp 4
 - progressive coding — iterate doubling resolution
 - may enable user to determine that the image is unwanted and stop transmission early with no penalty if image is wanted
 - image divided into stripes with user-defined height
 - user specifies order of stripes, resolutions, bitplanes

JBIG - Lossless B/W – Using Gray Code

- adjacent values differ in only one bit

0	0000	15	1000
1	0001	14	1001
2	0011	13	1011
3	0010	12	1010
4	0110	11	1110
5	0111	10	1111
6	0101	9	1101
7	0100	8	1100

- to convert pixel values to Gray Code values complement each bit that is preceded by a 1
- can also be expressed (to convert $b_0b_1\dots$) by:

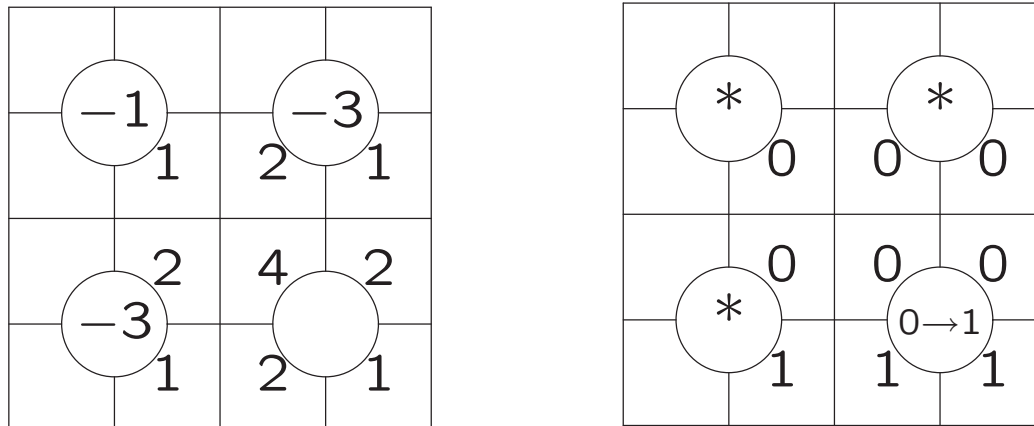
$$g_0 = b_0$$

$$g_k = b_k \oplus b_{k-1}$$

Example: 10011011 \rightarrow 11010110

JBIG - Lossless B/W – Progressive

- within a bit plane, determine pixels for lower resolutions
- majority value of $2 \times 2 \rightarrow 1$ pixel
- 2:2 tie \Rightarrow use 9 high-res plus 3 low-res pixels weighted as shown below (left), cut-off at 4.5



- **exception tables** to handle known anomalies
reverses polarity of derived pixel
Example bit pattern (above right) '*' denotes 'any'
enables preservation of single-pixel lines

JBIG - Lossless B/W – Encoding

- encode pixels of resolution layer
 - use 10-pixel context modeler
 - different near boundaries
 - different for lowest-res layer \Rightarrow 4096 contexts
- use multi-context adaptive arithmetic coder (IBM's Q-coder)
 - assumes $\text{Prob}(\text{MPS}) \approx 1$
(rescales until $\text{Prob} \geq 0.75$)
 - eliminates the multiplication by $\text{Prob}(\text{MPS})$

JBIG2 – Final Draft July 1999

- designed for compact lossy/lossless encoding
- designed to be embedded in other file formats
- leverages cross-page compression
- **segments a page** into ≤ 3 regions (may overlap)
 - regions use different coding methods
 - can be lossy, lossless, or progressive
- **regions**
 - **textual** regions contain "symbols"
 - **halftone** regions contain "patterns"
 - **other** regions contain "generic data"
 - use AC of template context (JBIG1)
 - or Huffman coding of pixel runs (MMR / Gp4)

JBIG2 – Characteristics

- compared with JBIG1 or MMR (Gp 4)
 - lossless coded images 2-5 times smaller
 - using cross-page compression
 - can make files additional 2x smaller
 - lossy coded images additional 3-10x smaller
 - with no noticeable differences
- compression takes 0.5-10 secs/page
decompression takes 0.02-1 secs/page
- can be annotated, is searchable, has limited editability

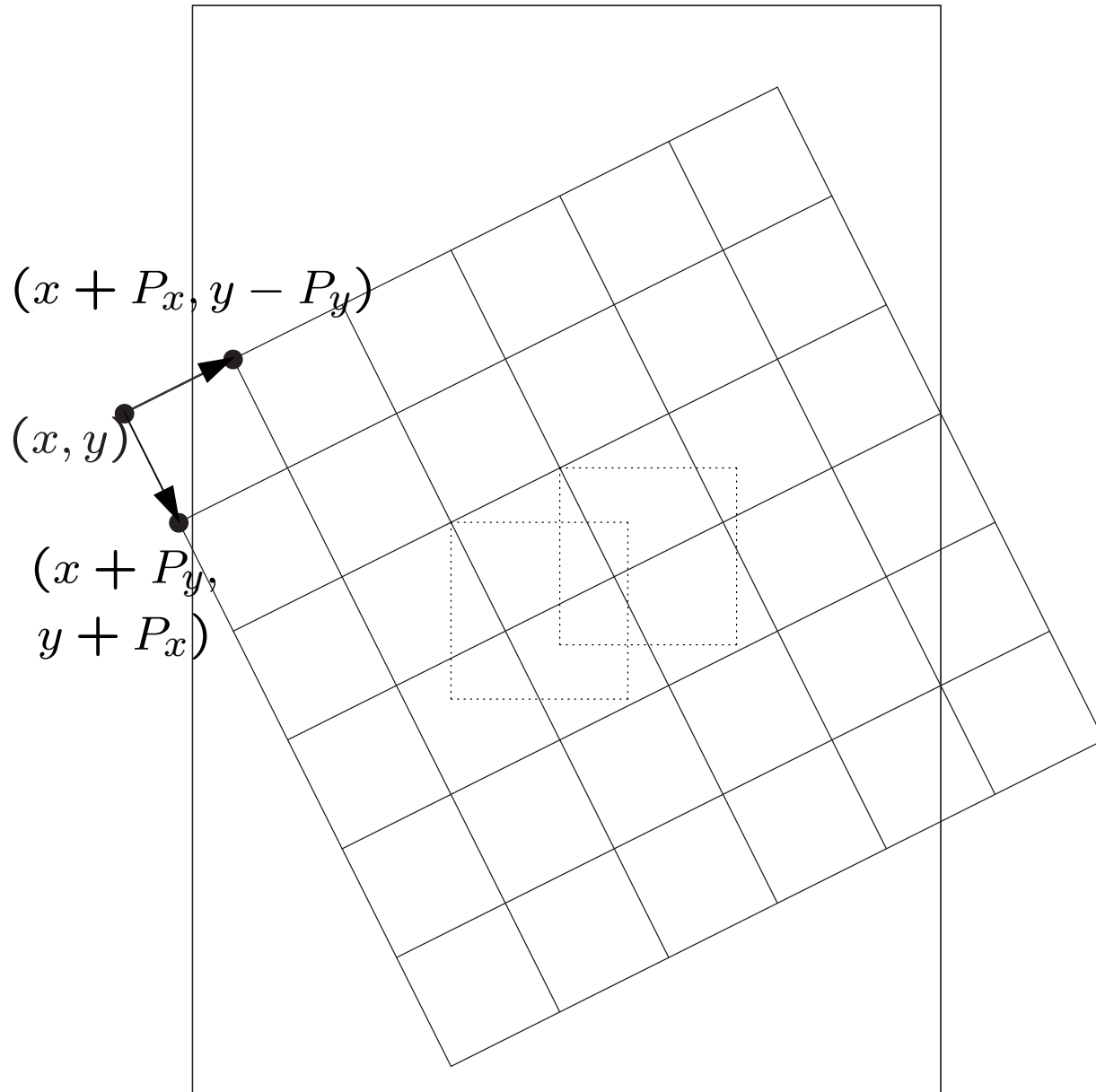
JBIG2 – Textual Regions

- consists of symbols at specified locations
- identify repeated symbols – intra and inter-page
 - need not be exact matches
 - Hausdorff distance is a possible basis for symbol match
$$H(A, B) = \max\{ h(A, B), h(B, A) \}$$
where $h(A, B) = \max_{a \in A} \min_{b \in B} \{ dist(a, b) \}$
 - see: godel.livia.etsmtl.ca/~normand/projets/hausdorff/
 - and: www.cs.cornell.edu/vision/hausdorff/hausmatch.html
- create symbol dictionaries containing symbol bitmaps
 - groups of symbols with similar ht, sorted by width
 - each group compressed by MMR or AC, symbol sizes delta-encoded
- encode
 - location – (X,Y) or (Y,X) for transposed text
 - differential encode Huffman-like
 - symbol ID
 - refinement information (if any)

JBIG2 – Halftone Regions

- consists of *patterns* on a regular grid
 - sequences of bits that occur periodically
 - often correspond to gray-scale values
- **create dictionary** containing pattern bitmaps
- **encode**
 - **grid origin** (x, y) scaled by 256 (8 bits), *i.e.*, origin at $(x, y) / 256$
 - x, y set so grid's origin is leftmost corner
 - at top-left when grid: is axis-aligned (AA), or
 - is a slight counter-clockwise rotation of AA
 - at bottom-left when grid is slight clockwise rotation of AA
 - **grid period** (P_x, P_y) unsigned, scaled by 256
 - **location** of pattern on grid
 - **indices** of patterns (think of gray levels)
 - first apply Gray code
 - then use generic region encoding of bitplanes

JBIG2 – Halftone Grid



dotted box = pattern bitmap area (dimensions independent of grid)

Lossless Grayscale

- Extend B/W methods
- Conditional RLE
- Multi-Level Progressive method (MLP)
- Progressive FELICS
- JPEG-LS

Lossless Grayscale – extend B/W methods

- convert pixel grayscale values to Gray Code
- apply a B/W method iteratively to the bitplanes
 - facsimile
 - quadtree
 - RLE of Hilbert Curve traversal
 - JBIG

Lossless Grayscale – conditional RLE

- high-level algorithm
 - in raster scan order, assign an adaptive codeword to each pixel
 - modify assigned codewords
 - RLE the concatenated codewords
- assign adaptive codeword to pixel P
 - consider the set of previously scanned pixels having same context
(pixel values on left & above)
 - rank the values by frequency (in stable fashion)
 - if P is rank r then assign $\text{GrayCode}(r)$
- modify assigned codewords
for pixel i in raster scan order
 if $\text{new}(i - 1)$ ends in "1"
 then $\text{new}(i) \leftarrow \text{complement}(\text{oldcode}(i))$
 else $\text{new}(i) \leftarrow \text{oldcode}(i)$

Example: old = 0001, 0000, 0011, 0000, 1110, 0001, 0000
 new= 0001, 1111, 1100, 0000, 1110, 0001, 1111

Lossless Grayscale – MLP

Multi-Level Progressive method (MLP)

[Howard and Vitter, 1992]

- **progressive pixel sequencing**
each level encodes twice as many pixels as previous level
- **image modeling (prediction)**
use context to predict intensities
- **error modeling**
Laplace distribution to estimate error probab
either use predefined variance V
or use pre-pass to calculate variance
- **arithmetic coding** to encode prediction error

Lossless Grayscale - MLP

- prediction

	○		○		○		○	
○		○		●		○		○
	○		●		●		○	
○		●		●		●		○
	●		●	?	●		●	
○		●		●		●		○
	○		●		●		○	
○		○		●		○		○
	○		○		○		○	

pixel predicted from
normalized weighted avg
of ≤ 16 neighbors at distances:

1,0 \Rightarrow weight +81

2,1 \Rightarrow weight -9

3,0 \Rightarrow weight +1

if all 16 neighbors present

normalization \Rightarrow divide by 256

Lossless Grayscale - Progressive FELICS

combines MLP with FELICS

(Fast, Efficient, Lossless Image Compression Systems)

- **pixel sequencing** – progressive levels, as in MLP
- **prediction** – from 4 neighbors
context = absolute difference
of 2 median values (referred to as L, H)
ties broken consistently
- **encoding**
one bit for “new value in/out range”
if in-range **then** use **modified binary code** for value
else 1 bit (defines range as hi/lo)
plus **subexponential code** (value in range)

Lossless Grayscale - Progressive FELICS

- Modified binary code

- need $n = H - L + 1$ codes

$$k \leftarrow \lfloor \lg n \rfloor$$

$$a \leftarrow 2^{k+1} - n$$

(even/odd as is n)

$$b \leftarrow 2n - 2^{k+1}$$

(always an even number)

Example: $n = 10, k = 3, a = 6, b = 4$

- the a **middle** values
are given k -bit codes **backwards** from **111...1**
- the b **extreme** values
are given $(k + 1)$ -bit codes **forwards** from **000...0**

Example:

middle: 111, 110, 101, 100, 011, 010

extreme: 0000, 0001, 0010, 0011

Lossless Grayscale - Progressive FELICS

- **subexponential code** (var. of Rice) with **parameter k**
to encode n ,
 - evaluate
$$b = \begin{cases} k, & \text{(if } n < 2^k\text{), or } \lfloor \lg n \rfloor, & \text{(if } n \geq 2^k\text{)} \\ u = \begin{cases} 0, & \text{(if } n < 2^k\text{), or } b - k + 1, & \text{(if } n \geq 2^k\text{)} \end{cases}$$
 - output u as unary ($u + 1$ bits), then low-order b bits of n
- **estimate coding param k** for each context adaptively
 - for each reasonable k , **maintain** cumulative **total t**
of total codelengths if k had been used for all values in this context
 - as each context is encountered
use k having smallest t for that context
 - at start of level, divide all stats by scaling factor s
(recommend $s = 12$)

Lossless Grayscale - JPEG-LS

- based on: LOCO-I (revised as of January, 1999)
see <http://www.hpl.hp.com/loco/>

c	b	d
a	x	

- **prediction**
primitive test to detect vertical or horizontal edge
else predict value in "plane" of 3 neighbors

$$\begin{aligned} c \geq \max(a, b) &\rightarrow \min(a, b) \\ c \leq \min(a, b) &\rightarrow \max(a, b) \\ \text{else} &\rightarrow a + b - c \end{aligned}$$

Lossless Grayscale - JPEG-LS

c	b	d
a	x	

- context model

- $g_1 = d - b$, $g_2 = b - c$, $g_3 = c - a$
each gradient g_i is quantized into 9 equiprobable regions
⇒ total of 729 contexts
- one region centered at 0, others symmetric
so quantizer $\kappa(g_i) = -\kappa(-g_i)$
- to predict residual from context $[q_1, q_2, q_3]$
if first non-zero element is negative
then use negative of prediction from $[-q_1, -q_2, -q_3]$
so, can merge opposite-sign contexts
⇒ only 365 contexts
- for 8 bpp,
regions: $\{0\}$, $\pm\{1,2\}$, $\pm\{3-6\}$, $\pm\{7-20\}$, $\pm\{21^+\}$

Lossless Grayscale - JPEG-LS

- prediction residual

modeled as 2-sided geometric distribution (TSGD)

with frequency = $c\tau^{|\epsilon+\rho|}$

- c is a normalization factor
- $\tau \in (0,1)$ is a decay rate parameter
- $0 \leq \rho < 1$ is a shift parameter (DC offset)
due to integer constraints & prediction bias

Lossless Grayscale - JPEG-LS

- **coding** – use Golomb-Rice code
 - TSGD parameter space partitioned into classes
 - depending on class:
 - determine parameter k for Rice code
 - use map $[-\alpha/2, \alpha/2 - 1] \rightarrow [0, \alpha - 1]$
 - then apply Rice code
 - sometimes transmit sign bit after
 - use slightly modified Rice code
 - to limit code length
 - **Example maps:**
 - $M(\epsilon) = 2\epsilon$ (if $\epsilon \geq 0$), else $2|\epsilon| - 1$
 - $M'(\epsilon) = M(-\epsilon - 1)$
 - use error feedback
 - re-center (bias) prediction residuals via ρ

Lossless Color - PNG

- **P**ortable **N**etwork **G**raphics (Oct 1996)
- **color representations** – choice of 5 methods
 - RGB – each 8 or 16 bits
 - Palette – bit depth 1, 2, 4, or 8
 - Gray scale – any bit depth, R=G=B = gray level
 - RGB with alpha channel
 - Gray scale with alpha channel (8/16 bits only)
- **alpha channel**

each pixel has α value $\in [0, \max = 2^{\text{depth}} - 1]$

 - α value = 0 \Rightarrow image is fully transparent
= max \Rightarrow image completely covers background
in between \Rightarrow for each $Q \in \{R, G, B\}$
 - $\text{output}.Q = \frac{\alpha * \text{img}.Q + (\max - \alpha) * \text{bkgd}.Q}{\max}$
 - **or** can implement transparency by associating α values with colors, not with pixels

Lossless Color - PNG

- optional interlacing – "Adam 7"
 - 8×8 pixel blocks updated over 7 passes

1	6	4	6	2	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7
3	6	4	6	3	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7

- can store additional information
 - background color
 - image smaller than display window
 - image has transparencies
 - text (possibly compressed)
 - last modified time

Lossless Color - PNG

- **compress** the (filtered) interlaced data stream
LZ77-based (32 Kb sliding window) **Deflate**
 - codestream of *literal* or (*offset,length*)
 - *literal* → code (0-255)
 - end of block → code (256)
 - (*offset,length*) → *length* code (257-285) + **extra bits**
then *offset* code (0-29) + extra bits
 - encode lit/len codes (0-285) using Huff code
 - encode offset codes (0-29) using 2nd Huff code

- **representing lengths** (can be in range 3-258)

code	baseval	extra bits	<i>length</i> values
257	3	0	3
...			
265	11	1	11-12
...			
269	19	2	19-22

- **represent offsets** (can be in range 1-32768) similarly

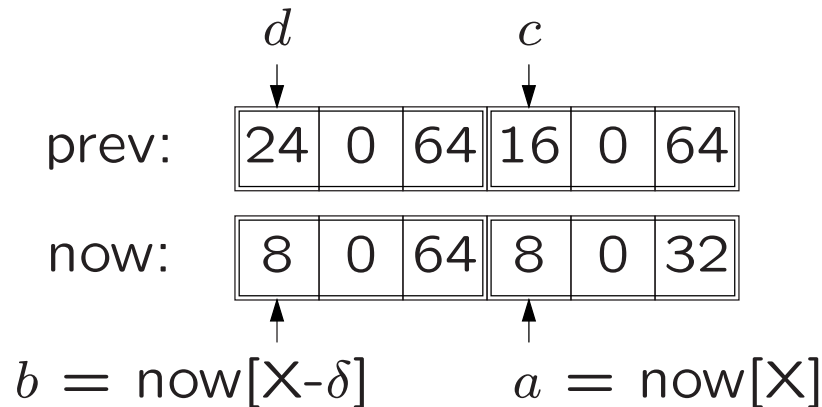
Lossless Color - PNG

- **Huffman codes** for *lit/len* and *offset* tables
 - can use **predetermined** codes
 - can use **static** code (requires two passes during compression)
 - pass 1: calculate then transmit Huffman code table
 - symbol code lengths are Huffman encoded!
 - pass 2: encode and transmit image
 - can end the first pass early if limited buffer space because compressed data can be stored in multiple blocks

Lossless Color - PNG

- optional filter – transform data to improve compressibility
 - each data row is preceded by a byte that specifies the filter
 - transform based on values of adjacent pixels
 - performed on per-byte basis

Example: RGB depth 16 \Rightarrow interval $\delta = 6$



- no filter $v = a$
- sub filter $v = a - b$
- up filter $v = a - c$
- avg filter $v = a - \lfloor \frac{1}{2}(b + c) \rfloor$
- Paeth filter $v = a - \text{PaethPredictor}()$

PaethPredictor is $\in \{b, c, d\}$ closest to $b + c - d$

Lossless Color - PNG

- how filter helps, when not to use
 - if color is changing on a gradient then values are *not* repeated but value differences *are* repeated
 - if palette is used then gradient color does *not* give gradient palette indices
 - if bit depth < 8 then gradient gray does *not* give gradient byte values
- filter selection suggestions (can change for each row)
 - how to optimize is unknown (**research!!**)
 - 1: perform *all* filters on a row
 - sum row values – treat each filtered value as a signed byte (–128...127)
 - select filter producing smallest sum
 - 2: perform *all* filters on a row
 - select filter giving longest repetition of values
 - 3: use no filter when palette or bitdepth < 8
 - else use sub for first row, Paeth for others