Embedded Zerotree Wavelet (EZW) Image Compression

The Original Paper that introduced the EZW Innovation

These Notes are Based on (or use material from):

1. **J. M. Shapiro**, "Embedded Image Coding Using Zerotrees of Wavelet Coefficients," *IEEE Trans. on Signal Processing*, Vol. 41, No. 12, pp. 3445 – 3462, Dec. 1993.
2. J. S. Walker and T. Q. Nguyen, "Wavelet-Based Image Compression," Ch. 6 in *The Transform and Data Compression Handbook*, edited by K. R. Rao and P. C. Yip, CRC Press, 2001.
3. C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG2000 Still Image Coding System: An Overview," *IEEE Trans. Cons. Elect.*, Vol. 46., No. 4, pp. 1103 – 1127, Nov. 2000.
4. B. E. Usevitch, A Tutorial on Modern Lossy Wavelet Image Compression: Foundations of JPEG 2000, *IEEE Signal Processing Magazine*, pp. 22 – 35, Sept. 2001.

Part of Abstract from Shapiro's Original Paper [1]

The embedded zerotree wavelet algorithm (EZW) is a **simple, yet remarkably effective**, image compression algorithm, having the **property that the bits in the bit stream are generated in order of importance**, yielding a **fully embedded code**. The embedded code represents a sequence of binary decisions that distinguish an image from the "null" image. **Using an embedded coding algorithm, an encoder can terminate the encoding at any point** thereby allowing a target rate or target distortion metric to be met exactly. Also, given a bit stream, the **decoder can cease decoding at any point** in the bit stream and still produce exactly the same image that would have been encoded at the bit rate corresponding to the truncated bit stream. In addition to producing a fully embedded bit stream, **EZW consistently produces compression results that are competitive with virtually all known compression algorithms** on standard test images. Yet this performance is achieved with a technique that **requires absolutely no training, no pre-stored tables or codebooks, and requires no prior knowledge of the image source**…

The EZW algorithm is based on four key concepts [1]:

1. Discrete wavelet transform (hierarchical subband decomp.)
2. Prediction of the absence of significant information across scales by exploiting the self-similarity inherent in images
3. Entropy-coded successive-approximation quantization
4. “Universal” lossless data compression which is achieved via adaptive arithmetic coding.

Why Wavelets? [1]

- Traditional DCT & subband coding: trends “obscure” anomalies that carry info
  - E.g., edges get spread, yielding many non-zero coefficients to be coded
- Wavelets are better at localizing edges and other anomalies
  - Yields a few non-zero coefficients & many zero coefficients
  - Difficulty: telling the decoder “where” the few non-zero’s are!!!
- Natural images in general have a low pass spectrum.
  - the wavelet coefficients will, on average, be smaller in the higher subbands than in the lower subbands.
- Large wavelet coefficients are more important than smaller wavelet coefficients.
- Significance map (SM): binary array indicating location of zero/non-zero coefficients
  - Typically requires a large fraction of bit budget to specify the SM
  - Wavelets provide a structure (zerotrees) to the SM that yields efficient coding

Motivation for EZW [1]

- Transform Coding Needs "**Significance Map**" to be sent:
  - At low bit rates a large # of the transform coefficients are quantized to zero ➔ **Insignificant Coefficients**
  - We'd like to not have to actually send any bits to code these
    - That is… allocate zero bits to the insignificant coefficients
  - But… you need to somehow inform the decoder about which coefficients are insignificant
    - JPEG does this using run-length coding: (Run Length, Next Nonzero)
  - In general… Send a **significance map**

Quantized Coefficients

64	56	48	32	24	16	0	0
56	48	40	24	16	23	0	0
40	40	30	24	16	8	0	8
32	32	32	24	24	16	0	0
24	24	16	8	0	0	8	0
16	16	8	0	0	8	0	0
0	0	0	8	0	0	0	0
0	0	0	0	0	0	0	0

Significance Map

1	1	1	1	1	1	0	0
1	1	1	1	1	1	0	0
1	1	1	1	1	1	0	1
1	1	1	1	1	1	0	0
1	1	1	1	0	0	1	0
1	1	1	0	0	1	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0

Motivation for EZW (cont.)

Here is a two-stage wavelet decomposition of an image. Notice the large number of zeros (black) but that run-length coding is not likely to be the best approach:

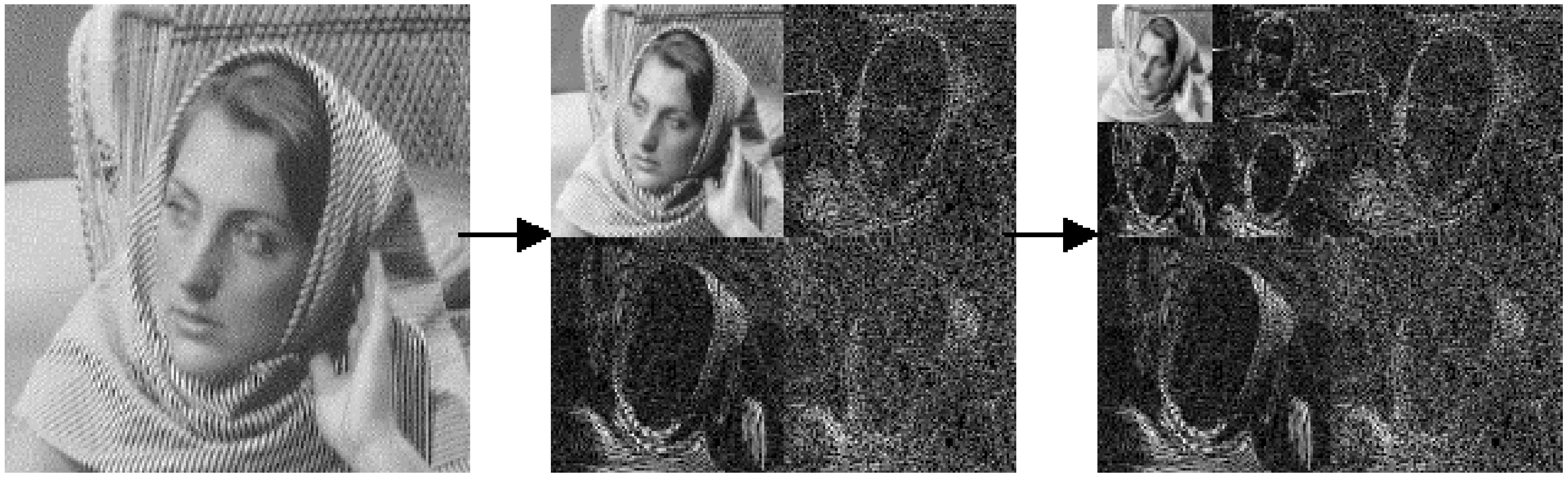


Fig. 5. Example of dyadic decomposition into subbands for the test image ‘barbara’

(Figure from [3])

Motivation for EZW (cont.)

But… Couldn’t we use entropy coding to make this more efficient? Yes… in fact, that *is* what JPEG does. But it is easy to see that even with entropy coding the significance map (SM) idea get “expensive” as we go to low bit rates [1].

Total Bit Cost = Bit Cost of SM + Bit Cost of Nonzero Values

Under some simplifying conditions (see [1]) Shapiro argued using entropy calculations that the percentage of total cost taken up by SM coding increases as the bit rate decreases!!!

➔SM approach gets increasingly inefficient as we try to compress more!!!!! The cost of specifying where the few significant coefficients are gets large at low rates

Example (see [1]): at 0.5 bit/pixel must use 54% of bits for SM

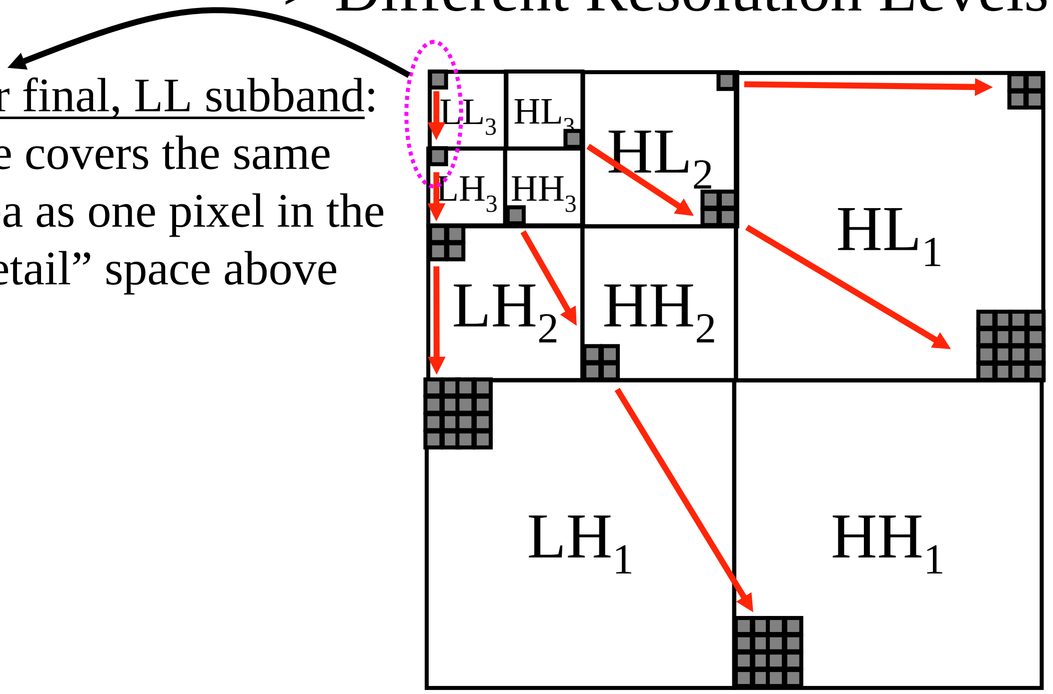
Shapiro’s Wavelet Idea for Solving SM Problem

Quad Trees:

> Same Spatial Region

> Different Resolution Levels (sub-bands)

For final, LL subband: one covers the same area as one pixel in the “detail” space above

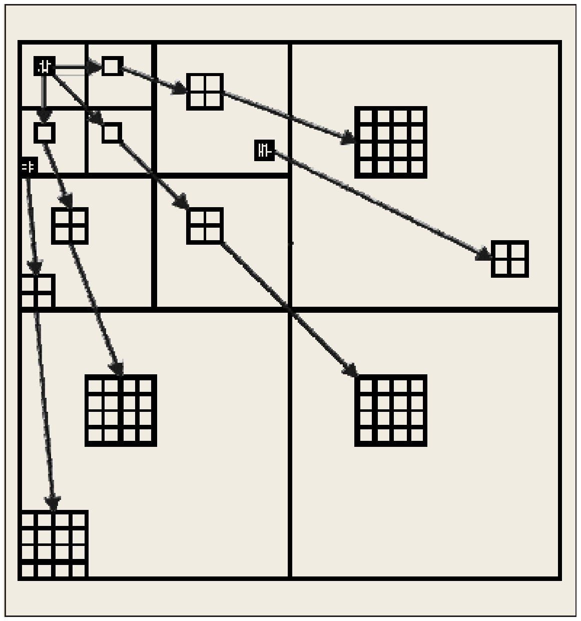


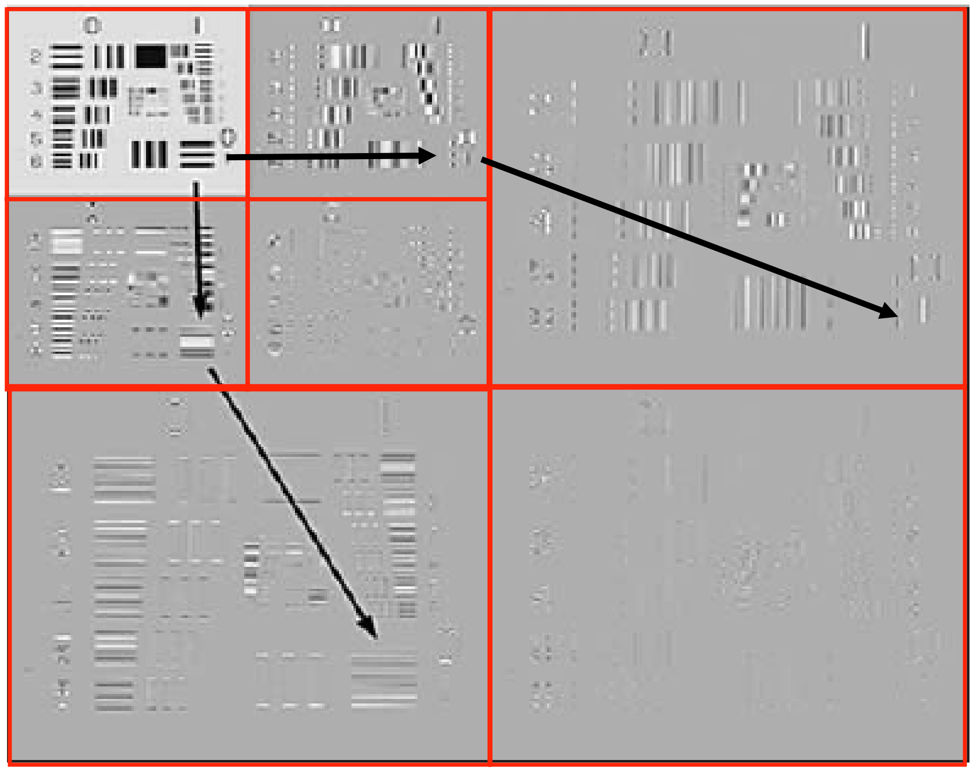
Idea: An insignificant coefficient is VERY likely to have all of its “descendents” on its quad tree also be insignificant

> Such a coefficient is called a “**Zerotree Root**”

Zerotree Coding

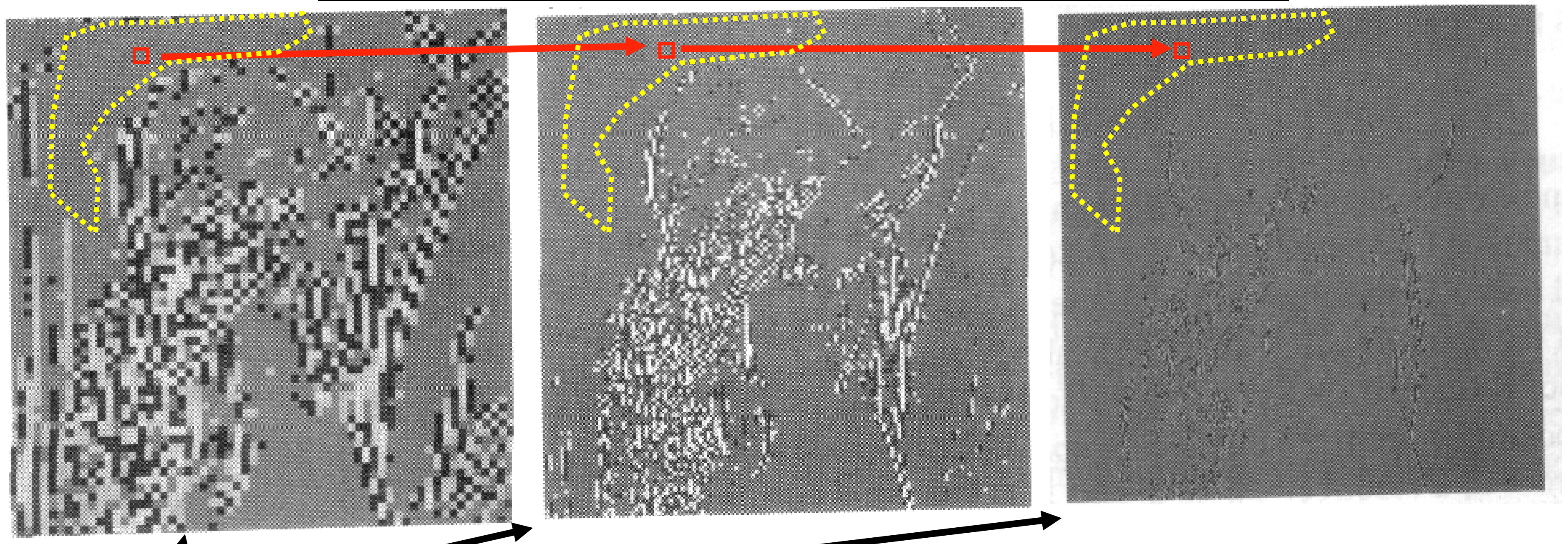
- Every wavelet coefficient at a given scale can be related to a set of coefficients at the next finer scale of similar orientation
- Zerotree root (ZTR) is a low scale "zero-valued" coefficient for which all the related higher-scale coefficients are also "zero-valued"
- Specifying a ZTR allows the decoder to "track down" and zero out all the related higher-scale coefficients

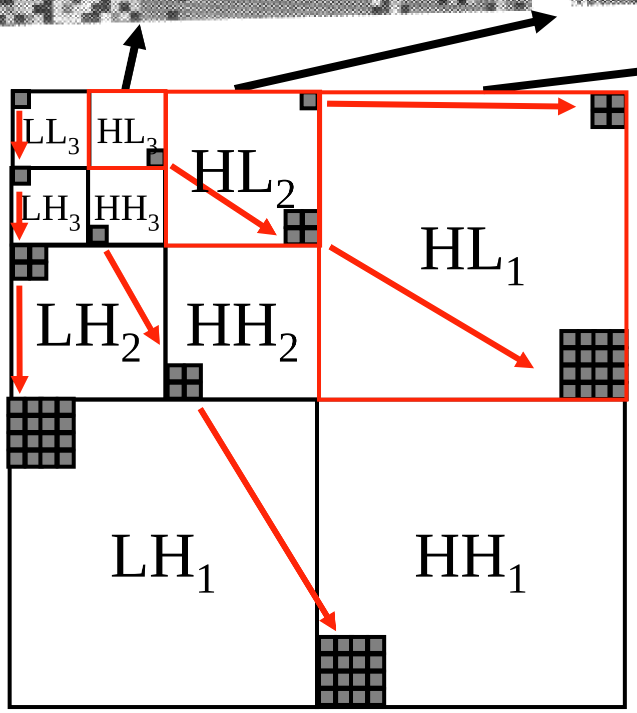

▲ 15. Example trees that can be defined on the wavelet transform. The roots of the three trees, indicated by shading, originate in the LL_3, LH_3, and HL_2 subbands.



From: B. E. Usevitch, A Tutorial on Modern Lossy Wavelet Image Compression: Foundations of JPEG 2000, IEEE SP Mag, Sept. 2001

Illustration of Zerotree Occurance





Coefficients are “thresholded at 16” in this example

Original Image

Image and its WT are from [2]

How Do Zerotrees Help?

The previous chart showed the prevalence of zerotrees. Now… how do they help with the SM problem?

You only have to tell the decoder where a zero root lies… it can figure out where all the descendent zeros on the tree lie by using the rule for generating quadtrees.

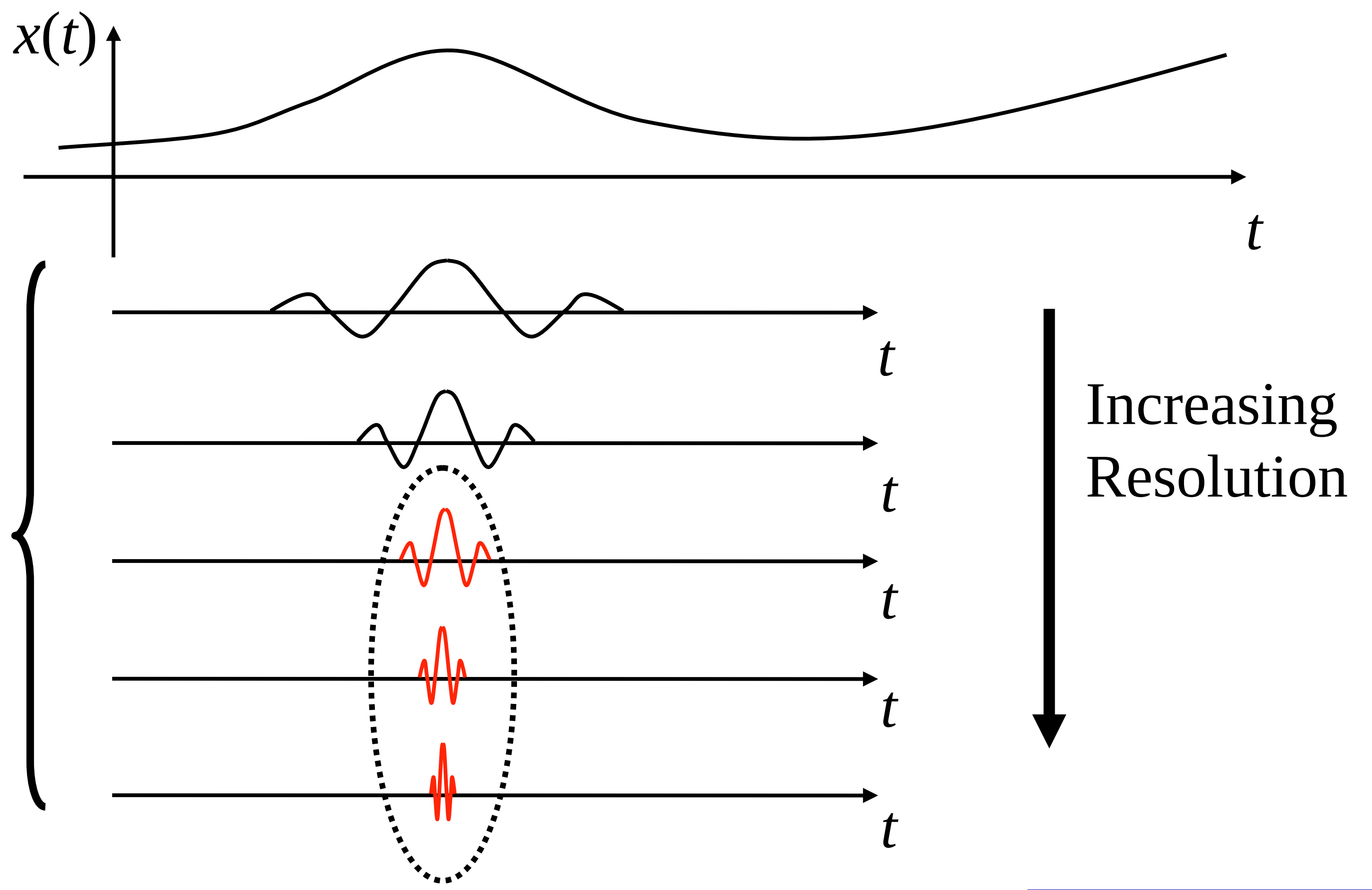
So… there is an agreed upon a rule and it so happens that zerotree roots happen alot when trying to code at low bit rates…

At low bit rates, zerotree roots occur frequently even at the coarse subband levels and that leads to long trees… and that very efficiently conveys the SM info

Note: we don’t really rely on a true SM, but we convey it using a model

What Causes Zerotrees?

Use 1-D example to illustrate:



Wavelets at various resolution levels

Increasing Resolution

These wavelets have insignificant inner product with signal at this location

Note: the wavelets themselves integrate to zero…

Successive Approximation: The Other Part of EZW

While zerotrees are a major part of EZW they are not the only significant part…

The other part has to do with embedded coding.

The goal of embedded coding is to create a bit stream that can be truncated at any point by the decoder….. AND you get a reconstructed signal that is R-D optimal for the number of bits so far received!

There are many ways to do this. EZW uses a successive approximation view of quantization…
… and it links this idea to zerotree coding in a way that allows zerotrees to be highly exploited.

Successive Approximation Quantizer

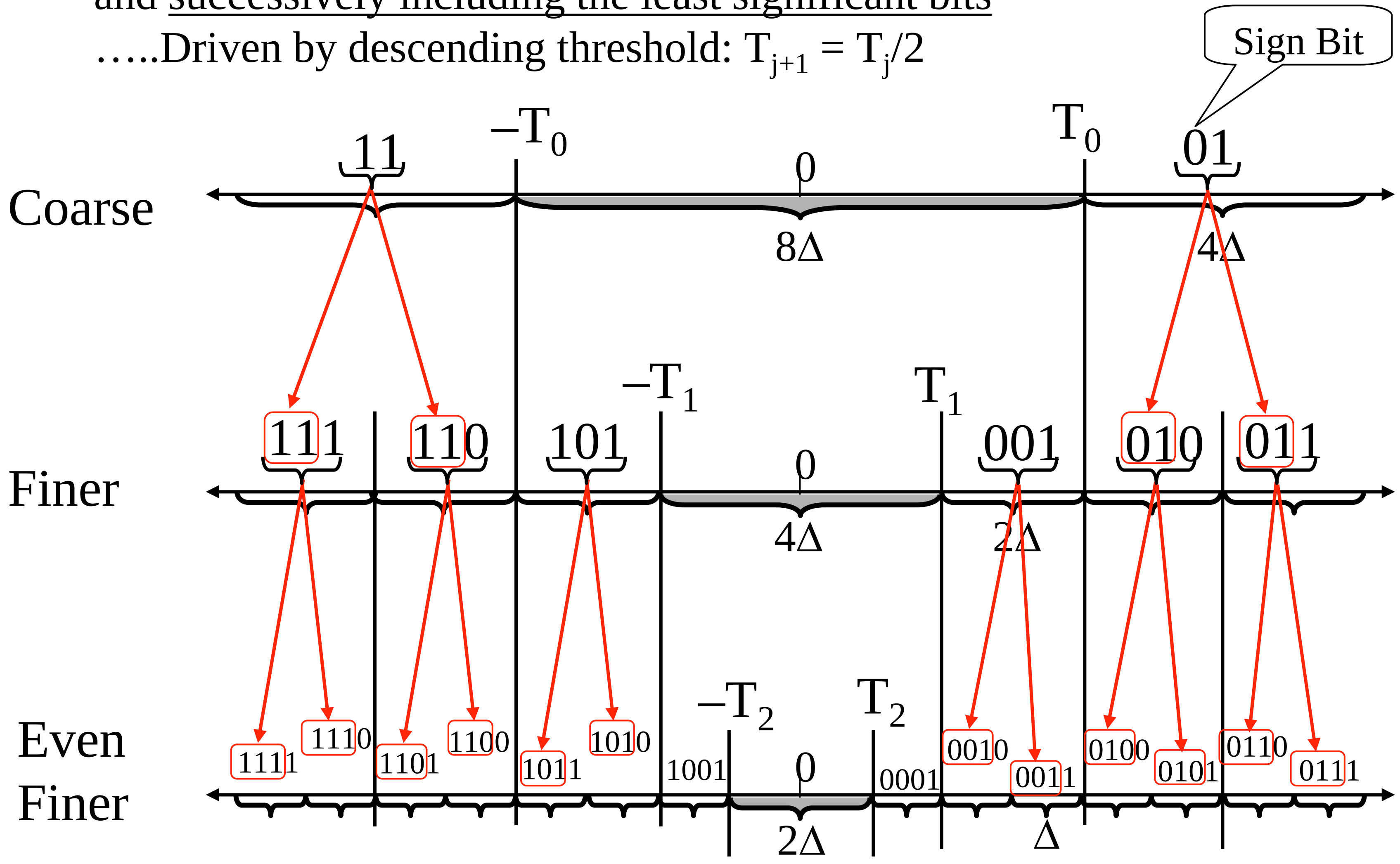
Start with coarsest and successively refine to the finest…

…. equivalent to starting with most significant magnitude bit

(sign bit is handled separately)

and successively including the least significant bits

…..Driven by descending threshold: $T_{j+1} = T_j/2$



Successive Approximation Quantizer (cont.)

Applying the SA quantizer with in EZW:

- Compute the wavelet transform of the image
- Set a threshold T_0 near the middle of the range of WT coefficient magnitudes
- This gives a large “dead zone” that creates of lots of “insignificant values”
  - These give rise to lots of zerotrees
  - Zerotrees efficiently handle significance map problem
  - Send MSB’s of significant coefficients
- Then reduce threshold: $T_{j+1} = T_j/2$
  - This causes some former insig coeff to become significant
  - ➔ only have to tell where new significance has occurred
  - For previously significant: refine by sending next finer bit

EZW Algorithm

Sequence of Decreasing Thresholds: $T_o, T_1, \ldots, T_{N-1}$

with $T_i = T_{i-1}/2$ and $|\text{coefficients}| < 2\,T_o$

Maintain Two Separate Lists:

- Dominant List: *coordinates* of coeffs not yet found significant
- Subordinate List: *magnitudes* of coefficients already found to be significant

For each threshold, perform two passes: Dominant Pass then Subordinate Pass

Dominant Pass (Significance Map Pass)

- Coeff's on Dominant List (i.e. currently insig.) are compared to T_i
  - asking: has this coeff become significant at the new threshold?
- The resulting significance map is zero-tree coded and sent:
  - Code significance using four symbols:
    - Zerotree Root (ZTR)
    - Isolated Zero (IZ)
    - Positive Significant (POS)
    - Negative Significant (NEG)
  - For each coeff that has now become significant (POS or NEG)
    - put its magnitude on the Subordinate List (making it eligible for future refinement)
    - remove it from the Dominant List (because it has now been found significant)

Entropy Code using an Adaptive AC

<u>EZW Algorithm (cont.)</u>

<u>Subordinate Pass (Significance Coefficient Refinement Pass)</u>

- Provide next lower signif. bit on the magnitude of each coeff on Subord List
  - Halve the quantizer cells to get the next finer quantizer
  - If magnitude of coeff is in <u>upper half</u> of old cell, <u>provide "1"</u>
  - If magnitude of coeff is in <u>lower half</u> of old cell, <u>provide "0"</u>
- Entropy code sequence of refinement bits using an adaptive AC

Now repeat with next lower threshold

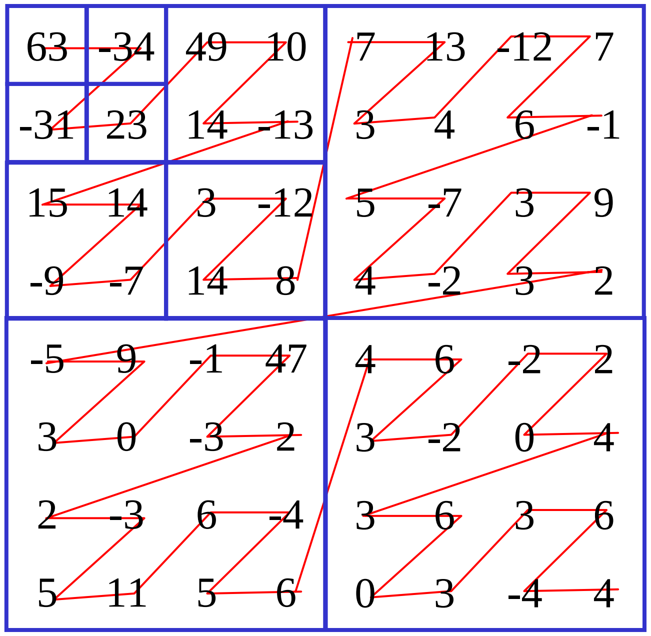
- Stop when total bit budget is exhausted

- Encoded stream is an embedded stream
  - At first you get an "optimal" low rate version
  - As more bits come you get a successively better distortion
  - Can terminate at any time prior to reaching the "full-rate" version

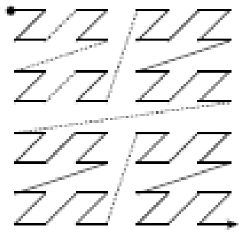
EZW Example (from [1])

First Thresholding

Example of 3-level WT of an 8x8 image

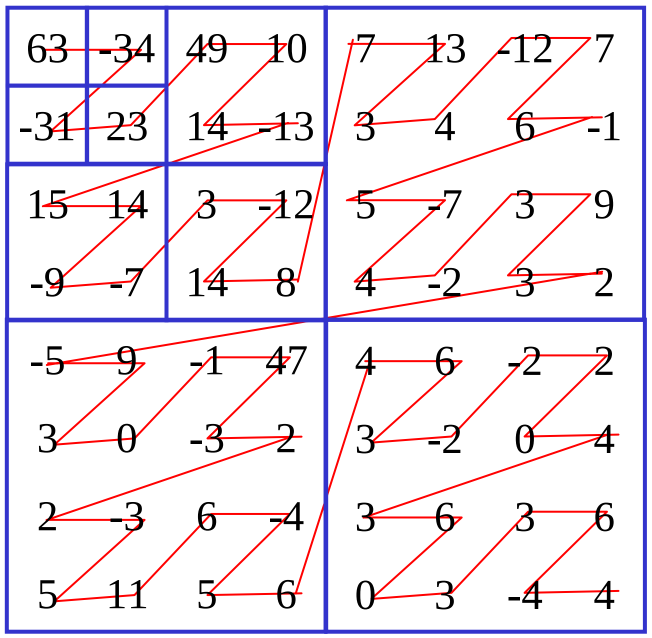


63	-34	49	10	7	13	-12	7
-31	23	14	-13	3	4	6	-1
15	14	3	-12	5	-7	3	9
-9	-7	14	8	4	-2	3	2
-5	9	-1	47	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

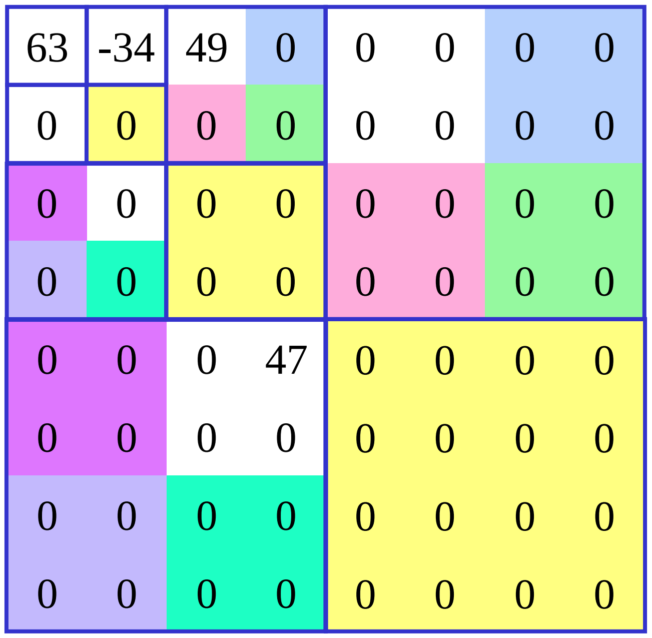


Largest coefficient magn = 63 ➔ $T_0 = 32$
… So after thresholding we have:

63	-34	49	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	47	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



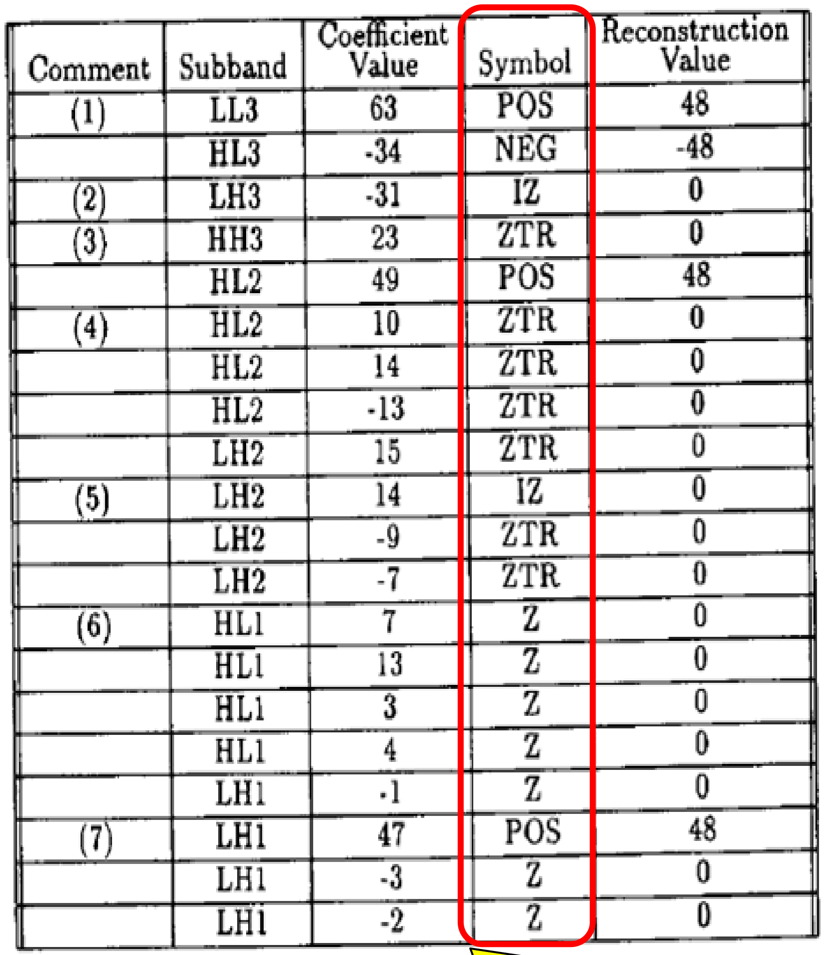
63	-34	49	10	7	13	-12	7
-31	23	14	-13	3	4	6	-1
15	14	3	-12	5	-7	3	9
-9	-7	14	8	4	-2	3	2
-5	9	-1	47	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4



63	-34	49	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	47	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

First Dominant Pass

PROCESSING OF FIRST DOMINANT PASS AT THRESHOLD $T = 32$. SYMBOLS ARE POS FOR POSITIVE SIGNIFICANT, NEG FOR NEGATIVE SIGNIFICANT, IZ FOR ISOLATED ZERO, ZTR FOR ZEROTREE ROOT, AND Z FOR A ZERO WHEN THERE ARE NO CHILDREN. THE RECONSTRUCTION MAGNITUDES ARE TAKEN AS THE CENTER OF THE UNCERTAINTY INTERVAL

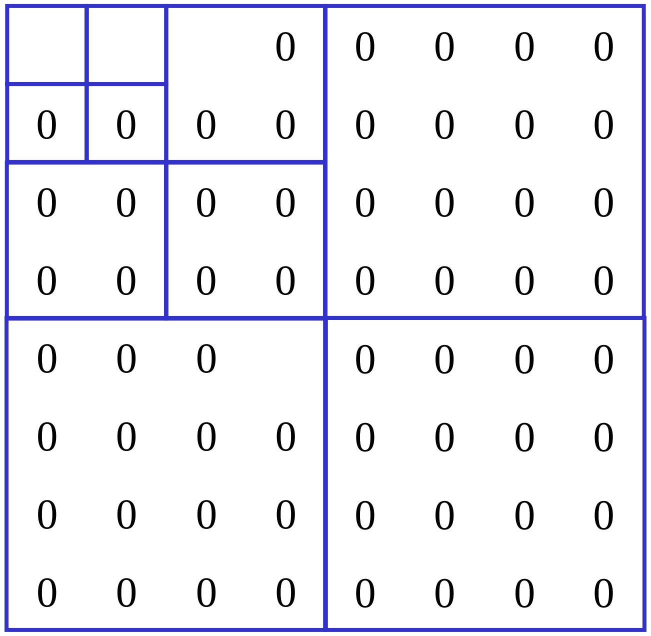


Comment	Subband	Coefficient Value	Symbol	Reconstruction Value
(1)	LL3	63	POS	48
	HL3	-34	NEG	-48
(2)	LH3	-31	IZ	0
(3)	HH3	23	ZTR	0
	HL2	49	POS	48
(4)	HL2	10	ZTR	0
	HL2	14	ZTR	0
	HL2	-13	ZTR	0
	LH2	15	ZTR	0
(5)	LH2	14	IZ	0
	LH2	-9	ZTR	0
	LH2	-7	ZTR	0
(6)	HL1	7	Z	0
	HL1	13	Z	0
	HL1	3	Z	0
	HL1	4	Z	0
	LH1	-1	Z	0
(7)	LH1	47	POS	48
	LH1	-3	Z	0
	LH1	-2	Z	0

Sequence of Symbols sent… but via Arith. Coding

- Dominant List: *coordinates* of coeffs not yet found significant
- Subordinate List: *magnitudes* of coefficients already found to be significant

Dominant List Contains
Pointers to all these zeros

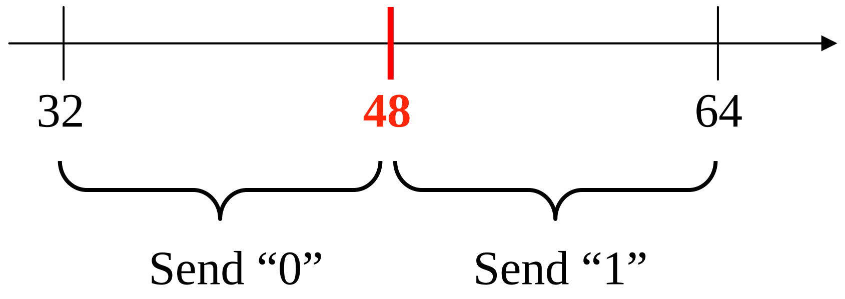


			0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0		0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Subordinate List
63
34
49
47

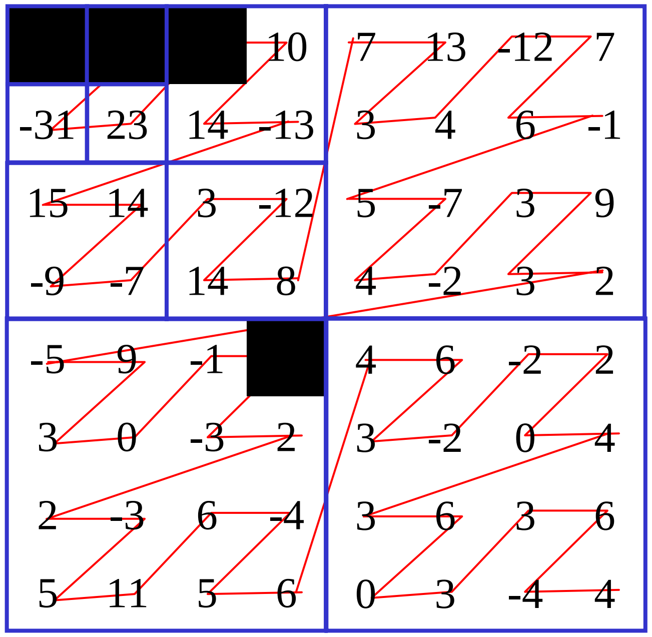
First Subordinate Pass

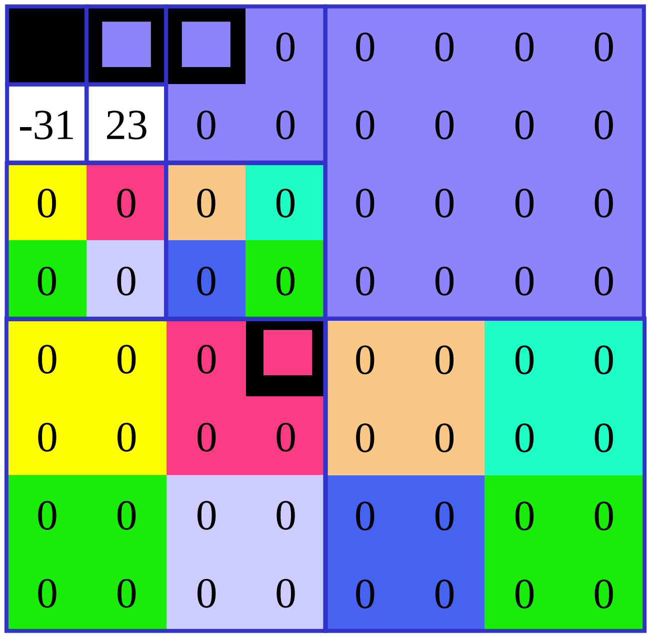
Now refines magnitude of each element on Subordinate List… Right now we know that each element's magnitude lies in (32,64]



Stream due to 1st Sub. Pass: 1 0 1 0

2nd Dominant Pass





New Thresh: $T_1 = T_0/2 = 16$

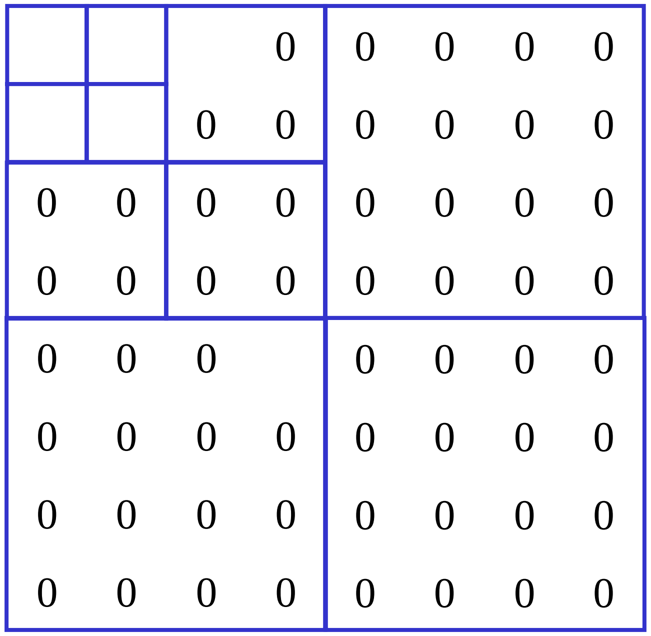
Only need to re-visit those on the Sub-Ord List… (not those on the Dom List… which are blacked on WT to the left) ***But*** *previously Significant values can be part of a zerotree!!!*

There is some ambiguity as to if the can be ZT Roots (Shapiro did NOT… some later papers DID)… We allow it here.

Coeff Value	Symbol	Recon Value
xxx	IZ	
xxx	ZTR	
-31	Neg	-24
23	Pos	24
15	ZTR	
14	ZTR	
-9	ZTR	
-7	ZTR	
3	ZTR	
-12	ZTR	
14	ZTR	
8	ZTR	

- Dominant List: *coordinates* of coeffs not yet found significant
- Subordinate List: *magnitudes* of coefficients already found to be significant

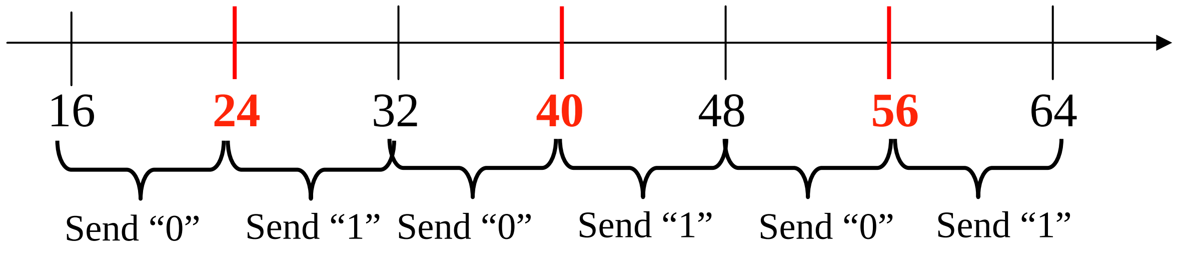
Dominant List Contains
Pointers to all these zeros



Subordinate List
63
34
49
47
-31
23

2nd Subordinate Pass

Now refines magnitude of each element on Subordinate List…



Stream due to 2nd Sub. Pass: 1 0 0 1 1 0

The algorithm continues like this until the threshold falls below a user specified minimum threshold

It is important to remember that the sequence of symbols (alphabet size of 4) output during each dominant pass is arithmetic coded.