# Low Cost Video Compression using Fast, Modified Z-Coding of Wavelet Pyramids

William Lynch, Krasimir Kolarov, Bill Arrighi
Interval Research Corporation, 1801-C Page Mill Road, Palo Alto, CA 94304
( lynch@interval.com, kolarov@interval.com )

## Abstract

*This paper describes a fast, low-complexity, entropy efficient video coder for wavelet pyramids. This coder approaches the entropy-limited coding rate of video wavelet pyramids, is fast in both hardware and software implementations, and has low complexity (no multiplies) for use in ASICs. It consists of a modified Z-coder used to code the zero/non-zero significance function and Huffman coding for the non-zero coefficients themselves. Adaptation is not required. There is a strong speed-memory trade-off for the Huffman tables allowing the coder to be customized to a variety of platform parameters.*

## 1. Introduction

An image transform codec consists of three steps: 1) a reversible transform, often linear, of the pixels for the purpose of decorrelation, 2) quantization of the transform values, and 3) entropy coding of the quantized transform coefficients. This paper presents an entropy codec which is fast, efficient in silicon area, coding-wise efficient, and practical when the transform is a wavelet pyramid [4]. We will focus on natural scene images quantized to match the human visual system (HVS).

We will begin by discussing the statistical characteristics of quantized wavelet pyramids derived from NTSC video quantized to be viewed under standard conditions. These *video* pyramids have substantial runs of zeros and also substantial runs of non-zeros.

We will develop a modification of the Z-codec [1] and explore an application of it to code zero vs. non-zero in quantized video pyramids. Z-codecs have the advantage of a simple (no multiplies) and fast implementation combined with coding performance approximating that of an arithmetic codec (see [6]). Our Z-coder implementation approximates adaptive binary arithmetic coder using dyadic broken line approximation. It has a very short "fastpath" and is attractive for application to wavelet significance function.
The non-zero coefficients of the pyramid are coded (coefficient-by-coefficient) reasonably efficiently with standard Huffman coding. We will examine the statistics of this situation and present the results.

## 2. Wavelet pyramid characteristics

### 2.1 Wavelet magnitude distribution

The typical wavelet pyramid acts as a filterbank separating the image into subbands each covering approximately one octave (factor of 2). At each octave typically there are three subbands corresponding to horizontal, vertical, and checkerboard features. Pyramids are typically three to five levels deep, covering the same number of octaves.

If the original image is at all smooth (images typically have a Hölder coefficient of 2/3 meaning roughly that the image has 2/3 of a derivative) the magnitude of the wavelet coefficients decreases rapidly. If the wavelet coefficients are arranged in descending order of absolute value, those absolute values will be seen to decrease as $N^{-s}$ where N is the position in the sequence and s is the smoothness of the image (see [2]).

### 2.2 HVC quantized wavelet pyramids

The wavelet pyramid is further sharpened if the wavelet pyramid is scaled to match the characteristics of the human visual system (HVS). We use fewer bits in the chroma subbands.

We think of quantization as a process of assigning scaling factors to each subband, multiplying each coefficient in a subband by the corresponding scaling factor, and *fixing* the scaled coefficient to an integer. The HVS contrast sensitivity function (CSF) peaks (1 part in $2^8$) at 4 cycles/degree of viewing angle, reducing to not noticeable (0) at 40 cycles/°. The CSF can be found in [7].

In a 300 pixel/in image viewed from a distance of 12 in, a pixel pair subtends 1/(150x12) of a radian or about 1/30 of a degree. The CSV justifies reducing the finest scale wavelets by a factor of $2^6$ (a shift right of 6) with respect to maximum contrast sensitivity to account for the reduced visibility of this fine detail. The $7^{th}$ bit of the fourth level wavelet (at 3.75 cycles/°) has about equivalent importance to the first bit of the finest scale wavelet (at 30 cycles/°). Such shifting by subband should bring bits of the same relative importance into the same bitplane, enabling progressive coding by bitplane scanning.

Right shifts implement dyadic quantization. Quantization shifts align coefficient bits of equal importance into the same bit planes. That also facilitates progressive coding.

### 2.3 NTSC video pyramids

We consider wavelet pyramids drawn from interlaced video consisting of fields of 240x640 pixels. A frame consists of two interlaced fields and is 480x640 pixels. A standard viewing condition is to view such video from six picture heights away ([3]) so that each pixel subtends 1/(480x6) radians or about (1/48)°. There are therefore 24 pixel pairs (cycles)/° in both the horizontal and vertical directions.

After forming the wavelet pyramid, the wavelet coefficients are scaled (quantized) consistent with the viewing conditions above and the CSF. Each block has the coefficients arranged by subband, with the coarsest subbands first and the finest subband last. Each subband is scanned out video-wise, by row, left to right, from the top row to the bottom row. Thus the magnitude of the coefficients decrease significantly through a block with the significant coefficients clustered at the beginning of the block and the insignificant ones clustered at the end of the block.

### 2.4 The significance function run statistics

The video wavelet pyramid coefficients, by block, were quantized to about 0.5 bits/pixel. About 85% of the wavelet coefficients are zero. The significance value of a coefficient is most likely to be the significance value of the preceding coefficient. Using this rule, 95% of the significance values are correctly predicted. There is an asymmetry in that a significant coefficient preceded by an insignificant coefficient is much more likely than an insignificant coefficient preceded by a significant one. An isolated significant coefficient embedded in a (fine subband) run of insignificant ones is much more likely than an isolated insignificant one in a (coarse subband) of significant ones.

Extending the preceding context to more than just the preceding coefficient does not qualitatively change the prediction but it does affect the probability of the significance of the next coefficient. Initially we thought that a context of 9 coefficients would be useful since this would allow better prediction due to vertical adjacency. To our surprise, the "knee" of this effect was after 3 bits so that just 8 context states are practical.
The resulting statistics are remarkably stable over a wide range of clips. As a result we have eliminated the adaptation of the probability tables and used fixed probabilities.

### 2.5 Non-Zero coefficient statistics

The histogram of the non-zero coefficients is very spread out. Even the histogram of the binary exponent, $\lfloor \log_2(|coeff|) \rfloor$, is very spread out, with the largest probability being about 1/8. There is also surprisingly little correlation between one coefficient and the next.

## 3. Requirements – a fast, low complexity algorithm for ASICs

We are interested in fast algorithms implementable in a small amount of silicon area, even at some modest cost in coding efficiency. With only 15% of the coefficients requiring coding of the coefficient value, speed and efficiency in identifying that minority of values via the significance function is clearly the most important problem.

The average run of correct prediction of significance values is about 20, so efficient run coding is important. Additionally, the importance of the 3 bits of context and the asymmetry strongly indicates the use of an arithmetic coder.

However, the requirement for a fast algorithm implementable in minimal silicon area demands that something other than a traditional arithmetic coder be used. In particular, multiplies are to be avoided as they are very expensive in silicon area. The chosen algorithm should have a very good "fast path" for the individual elements of the runs.

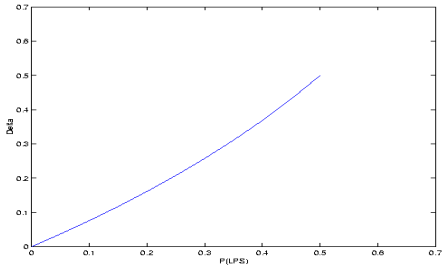
The fact that the significance function has only two values is a specialization not taken advantage of by arithmetic coders in general.

## 4. A modified Z-coder

The Z-coder, described in [1], can be viewed as a coder for a binary symbol set which approximates an arithmetic coder. As described in [1] it approximates the coding curve by a dyadic broken line. This enables a binary coder with a short "fastpath" and without requiring multiplies. These properties make it an attractive candidate for coding the wavelet coefficient significance function.

Familiarity with [1] is assumed in the following description. Recall that the preceding context (ctx) predicts with probability $P(ctx) = P(\mathbf{MPS})$ the next symbol. The bit predicted is referred to as **MPS** (Most Probable Symbol) whereas the other choice (there are only two symbols in the set) is referred to as **LPS** (Least Probable Symbol). We always have $P(ctx) \geq \frac{1}{2} \geq 1 - P(ctx) = P(\mathbf{LPS})$. In

[1] $\Delta$ is given implicitly, as a function of $P(\mathbf{LPS})$, as




$$P(\mathbf{LPS}) = \Delta - (\Delta + \tfrac{1}{2})\log_e(\Delta + \tfrac{1}{2}) - (\Delta - \tfrac{1}{2})\log_e(\tfrac{1}{2}) \quad (4.0)$$

**Figure 1. $\Delta$ vs. P(LPS)**

The graph of P is in figure 1. We always have $0 < \Delta \le \tfrac{1}{2}$ (n. b., *sure* symbols are eliminated).

As in an arithmetic coder, the code word $C$ is a real binary number with a normalized lower bound $A$. The *split point* Z computation is given in equation (4.3).

$$0 \le A \le C < 1 \quad (4.1)$$
$$A < \tfrac{1}{2} \quad (4.2)$$
$$A < Z = A + \Delta < 1 \quad (4.3)$$

In other words a split point *Z* in (*A*, 1) is determined by the probability

prob(**MPS**|context) = 1-prob(**LPS** |context)

*C* in [*A*, *Z*) codes **MPS** while C in [*Z*, 1) codes **LPS.** If we wish to code an **MPS** we arrange to output code bits so that $Z \le C < 1$; we have $A \le C < Z$ to code an **LPS**.

On encode we start not knowing any of the bits of $C$. However, if the lead (i.e., $2^{-1}$) bit of $Z$ and of $A$ are identical then we know that the lead bit of $C$ must agree. So we shift (normalize) the binary point of everything one place to the right and subsequently ignore bits to the left of the binary point as $A, C$ and $Z$ must agree there.

The normalizing shift ensures that (4.2) holds at the beginning of the next symbol. The **MPS** case is relatively easy since the normalizing shifts ensure that $A_{new} = Z_{old} \le C_{new} < 1$.
A head of $C$ is the codeword for a head of the input symbol string. It is dyadically normalized into the interval *[0, ½)*, becoming the lower bound $A$. $A$ becomes renormalized $C$ and the process is repeated for the next symbol (renormalization being a multiplication by 2).

The **LPS** case is more delicate. Since the correct $C$ is completely unknown right of the binary point, we must perform binary point shifts (bits shifted out of $A$ go to the code string) until we are assured that the $C$ that appears in the decoder will be not less than the $A$ that appears in the decoder. Since $C$ is completely unknown at encode time the only way to ensure this is to shift until $A = 0$. In Z-coding, the **LPS** sharp invariants must be maintained, since not sharp implies coder inefficiency. Here *C* is in [*A*, 1), *A* is in [0, ½) and *Z* is in (*A*, 1).

At this point there are two possible cases. If the shifted out part of $Z$ does not match the shifted out part of $A$ then we are assured that $Z > C = A$ as it is recomputed in the decoder and an **LPS** will be decoded. Otherwise, the shifted out part of $C$ equals the shifted out part of $Z$ with the subsequent bits of $C$ unknown. It is possible that subsequent bits of $C$ will match or exceed $Z$, resulting in the incorrect decode of an **MPS**. To prevent this case we must translate $A$ up by $1 - Z$, making use of the $C < 1$ property to keep $C < Z$. Now (4.2) may not be satisfied so we need more normalizing shifts to finally ensure (4.2) and (4.1).

At the end of **LPS** encode, if *C* is completely unknown that implies normalization until *A=0* and *Z>A* as computed by the decoder. It also means that extra normalization shifts may be required.

## 5. Z-encoder and Z-decoder

The resulting algorithms are given in Figure 2, as close to the style of [1] as possible.

***Modified Z-encoder***

```
Z:=A+Δ;
if (bit = MPS and Z<½) { A := Z; return; }
else {
   if (Z>½) { Z := Z/2+¼; }
   if (bit = LPS) {
      Zgtr := false;
      while (A>0) {
   if (A<½ and not Z<½ and Zgtr =false) { Zgtr := true;}
       emit(floor(2A)); A := mod(2A, 1); Z := mod(2Z, 1); }
         if (not Zgtr) {
           A := 1-Z;
           while (not A<½) { emit(0); A := mod(2A, 1); } } }
   else {
      A := Z;
      while (A>½) { emit (1); A := mod(2A, 1); } } }
```

***Fast modified Z-decoder***

**Z:=A+Δ;**
*if* **(Z<F) { A := Z; bit :=** *MPS;* **}**
**else** {
  **if** (*Z*>½) { Z := Z/2+¼; }
  **if** (**not** *Z*>*C*) { bit := **MPS**; *A* := *Z*; }
  **else** { bit := **LPS**;
    *Zgtr* := **false**;
    **while** (*A*>0) {
   **if** (*A*<½ **and not** *Z*<½ **and** *Zgtr* =**false**) { *Zgtr* := **true**;}
     *A* := mod(2*A*, 1); *C* := mod(2*C*, 1); *Z* := mod(2*Z*, 1); }
    **if** (**not** *Zgtr*) { *A* := 1-*Z*; } } }
  **while** (*A*>½) { *A* := mod(2*A*, 1); *C* := mod(2*C*, 1); }
*F* := min(*C*, ½); }

**Figure 2. Z-coder Algorithm**

## 6. Huffman coding the non-zero coefficients

### 6.1 Encoding

The distribution of the values of the non-zero coefficients demonstrates the preponderance of small values. We also know that the bits after the first few have little effect on the distribution and can encode themselves (self-encode). The sign (non-zeros only) also has nearly a 50-50 probability and can efficiently self-encode. Encoding can therefore be done efficiently by table look-up.

We begin by taking the absolute value of the coefficient and self-encoding the sign. We then take the last few bits (e.g., bits 0-7) of the coefficient, test to see if the remainder bits (8-N) are only leading zeros, and if so use the last few bits to index into a table E1 ($2^8$ entries). The table will contain the Huffman code and the number of bits in the Huffman code. The Huffman codes can be prepared by lumping all values greater than 255, making coding room for the larger values.

If bits 8-N are not zero but bits 14-N are zero, bits 6-13 are used to index into another table E2. It will also contain the Huffman code and its length. The codes for this table can be prepared by separating the lumps described in the previous paragraph. Appropriate coding room is left for even larger values (after emitting the Huffman code for bits 6-13, the self-coded bits 0-5 are emitted).

This process may be iterated as required. The sizes of the tables and the number of levels can be varied in the obvious ways.

### 6.2 Decoding

The decoder is a bit more complicated. The first step is to input the sign bit. Then the next 8 bits are used to index into a table D1 (without removing them from the input string). There is a high probability that the next Huffman code will be a head of this index, but this is not guaranteed. A flag in the table will indicate which case holds.

In the first, high probability, (terminal entry) case table D1 needs to contain the decoded bits (8 of them in our example) and the number of bits in the Huffman code. The indicated number of bits are removed from the input string. Table D1 also needs a count of the number of self-coded bits that follow and these bits must be removed from the input string and composed with the decoded value and the sign to recover the coefficient.

In the second case, the table D1 entry must contain the location and $\log_2$ length (k) of a follow-up table $Df_i$. The 8 bits used to index D1 are but a head of the full Huffman code and must be removed from the input string. The next k bits are used to index $Df_i$. The process is repeated until a terminal table entry is located. The "k"s may vary from entry to entry. Optimization of these values will trade off table space for execution time.

## 7. Preliminary results

This section presents the results of some initial experiments we performed using the algorithm described above.

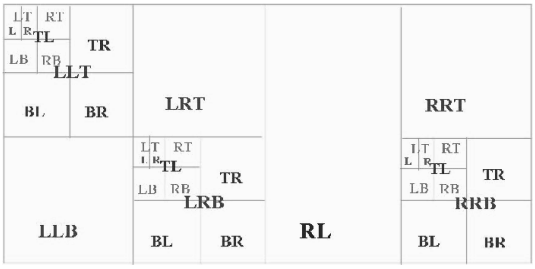



**Figure 3. Component wavelet pyramid**

We applied our WZD coder on NTSC wavelet video. The input is a D2 digitization of NTSC video where the chroma1 and chroma2 are quadrature modulated on a 3.58 MHz sub-carrier. WZD uses a composite 2-6 wavelet pyramid described in [3] plus two levels of Haar pyramid in the time direction (4 field GOP). The dyadic quantization coefficients are powers of 2. The resulting composite pyramid is depicted in Figure 3.

The algorithm was tested on several NTSC clips which vary in content and origin. The first one is a cable broadcast of an interview ("talking heads") without much motion. The second clip is a clean, high quality sequence from a laserdisk with a panning motion of a fence with vertical bars close together and motion of cars on the background. The next clip is a DSS (satellite) recording of a basketball game (already MPEG2 compressed /decompressed) with a lot of motion and detailed crowd and field. The last clip is a high quality sequence from a laserdisk with a zooming motion on a bridge with a number

of diagonal cables. The size of the frames is 720x486 (standard NTSC) in .tga (targa) format.

The probability values that were used for this experiment are as follows:

$P_0$=0.0107696; $P_1$=0.2924747; $P_2$=0.5; $P_3$=0.1588221;

$P_4$=0. 2924747; $P_5$=0. 2924747; $P_6$=0.5; $P_7$=0. 1588221

We have used 3 bits of context and the subscripts above denote the different contexts. We chose this because return diminish after a few bits of context and we can achieve 95% prediction with 3 bits of context.

In the results described below, we used a very crude scheme for non-zero coefficients coding. We only code off leading zeroes, the sign and the other bits are coded as themselves. Significance bits in the interval (0, 9) are coded in 9 bits, those in (8, 14) in 23 bits and those in (13,19) in 37 bits. Most non-zeros are coded in 9 bits.

| | **WZD** 1.0 bpp | **MPEG2** 1.0 bpp | **WZD** 0.5 bpp | **MPEG2** 0.5 bpp |
|---|---|---|---|---|
| TalkShow | 35.22 dB | 37.03 dB | 33.90 dB | 34.85 dB |
| Fence | 30.33 dB | 29.62 dB | 26.12 dB | 26.00 dB |
| Basketball | 27.37 dB | 31.69 dB | 24.69 dB | 28.42 dB |
| Bridge | 39.31 dB | 39.35 dB | 36.14 dB | 37.34 dB |

**Figure 4. Our z-coder vs. an MPEG2 coder.**

Figure 4 illustrates the PSNR results for two bit rates (corresponding to 11 Mbs and 5 Mbs resp.) for the four video sequences. For comparison we have used high quality commercially available MPEG2 codec from PixelTools. The MPEG2 was generated using the best possible settings for high-quality compression. We used 15 frames in a GOP (group of pictures), 3 frames between anchor frames, 29.97 frame rate, 4:2:0 chroma format, medium search range double precision DCT prediction, stuffing enabled, motion estimation sub-sampling by one. The sequences were compressed at 1.0 bpp and 0.5 bpp.

We have also compared our z-coder (with identity Huffman tables), with an arithmetic coder that we built (see [3]). That algorithm used a separate arithmetic coder for each bitplane. The transform part for both the z-coder and the arithmetic coder is the same 2-6 wavelet pyramid (see [4] and [5]). This arithmetic coder is on par with MPEG2 in a number of sequences in terms of PSNR (signal-to-noise ratio - mean-square error).

The only sequence that MPEG2 achieves statistically better PSNR is the basketball sequence in which MPEG can take advantage of the significant amount of (expensive) motion estimation characteristic for that method. Also remember that this sequence was a recording from DSS, i.e. it was already MPEG compressed and decompressed before being tested with the coders.

We should also mention that perceptually the quality of MPEG2 vs. arithmetic vs. z-coder is very similar. For the fence sequence in particular the quality of MPEG2 compressed video deteriorates significantly for lower bitrates, even though the PSNR is comparable to our coder. On the other side even though the basketball sequence presents an advantage for MPEG in terms of PSNR, visually the three methods are very comparable.

The other big advantage of the method we describe in this paper is its simplicity and speed in view of hardware implementation. In fact even the research version of our code, non-optimized for speed, is several orders of magnitude faster than the commercial (well optimized) MPEG2 software encoder we used for the same quality. Our optimized coder should achieve 20-30 times improvement in performance with respect to MPEG2.

## Bibliography

[1] L. Bottou, P. G. Howard, and Y. Bengio, The Z-Coder Adaptive Coder, *Proceedings of the Data Compression Conference, pp. 13-22,* Snowbird, Utah, March 1998.

[2] R. DeVore, B. Jawerth and V. Popov, Compression of wavelet decompositions, *American J. of Mathematics 114 (1992), 737-785.*

[3] K. Kolarov, W. Lynch, Very Low Cost Video Wavelet Codec, *SPIE Conference on Applications of Digital Image Processing, Vol. 3808*, Denver, July 1999.

[4] S. Mallat, A Theory for Multiresolution Signal Decomposition: The Wavelet Representation, *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.11, pp. 674-693*, 1989.

[5] E. Schwartz, A. Zandi, and M. Boliek, Implementation of Compression with Reversible Embedded Wavelets, *Proceedings of the SPIE 40th Annual Meeting, Vol. 2564-04*, July 1995.

[6] L. Stuiver and A. Moffat, Piecewise Integer Mapping for Arithmetic Coding, *Proceedings of the Data Compression Conference*, Snowbird, Utah, March 1998, pp.3-12.

[7] Brian Wandell, "Foundations of Vision", Sinauer Associates, Inc. Publishers, 1995.