Very low cost video wavelet codec

Krasimir Kolarov, William Lynch

Interval Research Corporation, 1801-C Page Mill Road, Palo Alto, CA 94304
(kolarov@interval.com , lynch@interval.com)

ABSTRACT

This paper describes a motion Wavelet transform Zero Tree (WZT) codec which achieves good compression ratios and can be implemented in a single ASIC of modest size (and very low cost). WZT includes a number of trade-offs which reduce the compression rate but which simplify the implementation and reduce the cost. The figure of merit in our codec is ASIC silicon area required, and we are willing to sacrifice some rate/distortion performance with respect to the best available algorithms to that goal.

The codec employs a group of pictures (GOP) of two interlaced video frames (i.e., four video fields). Each such field is coded by the well-known *transform* method, whereby the image is subjected to a 2-D linear transform, the transform values are quantized, and the resulting values coded (e.g., by a zero-tree method). To that goal we are using 3D wavelet transform, dyadic quantization and various entropy codecs. In the temporal direction a temporal transform is used instead of motion estimation.

Some of the technical innovations that enable the above features set are:

- Edge filters which enable blockwise processing while preserving quadratic continuity across block boundaries, greatly reducing blocking artifacts.
- Field image compression which reduces memory requirements for fields within a GOP.

Keywords: Video compression, wavelets, zerotree coding, composite source, very low cost.

1. INTRODUCTION

Compression methods use three main steps in processing data - transform, quantization and entropy coding. The standard MPEG approach uses DCT (discrete cosine transform) for the first part of the compression process. Despite its wide popularity due to the standartization process, there are a number of approaches that can perform better or more cost effectively then DCT-based methods. In particular there are a variety of wavelet-based techniques that can be more appropriate for important real world applications. This paper describes such approach that focus on low cost implementation (in hardware or software).

We use simple wavelet transform which can be implemented using only shifts and adds (no multiplies), since parallel multiplies use significant ASIC area. We also employ 2nd order boundary filters which facilitate block-by-block processing and save RAM and RAM bandwidth. In addition significant RAM savings are achieved through field buffers compression and decompression. This is achieved as well through intermediate compression in the time direction.

The target application for the proposed codec is real-time NTSC composite signal from legacy sources at moderate data rates (5 - 11 Mb/s which translates to 0.5 - 1.0 bpp) characteristic for DVD quality video. The goal of the codec is to obtain visual quality comparable to standard MPEG2.

The method performs direct compression of the composite NTSC signal with the compression code doing simultaneously the decoding of the chroma from the carrier. Thus cheap color conversion is possible. We allow for both S-video and component video modes with a very low latency - less than 6 frame times. In ASIC that also translates into low power requirements on a 28.7 MHz clock.

There are a number of alternative approaches. One of them is ADV601 compressor from Analog Devices [1] - it requires auxiliary DSP and does not do temporal compression. This compressor employs the 9-7 Daubechies wavelets which use an enormous amount of multiplies for implementation. Almost all other wavelet techniques use similar long filter wavelets which are expensive for hardware implementation, as well as motion image codecs. Motion JPEG also requires multiplies and typically has worse rate/distortion performance in our target range of interest. Finally MPEG requires multiplies as expensive motion estimation for achieving good rate/distortion performance.

Alternatively, our approach uses short filters and allows for simple implementation on a portion of a small single chip. Some of the features of our approach include:

- Motion image compression is used in place of motion compensation
- Transform filters are short and use dyadic rational coefficients with small numerators. Implementation can be accomplished with adds and shifts.
- Processing can be decoupled into the processing of stripes of 8 scan lines each. This helps reduce the RAM requirements to the point that the RAM can be placed in the ASIC itself. This reduces the chip count and also simplifies the satisfaction of RAM bandwidth requirements.
- Quantization denominators are powers of two, enabling implementation by shifts.
- Zero-Tree coding yields a progressive (i.e., embedded) encoding which is easily rate controlled
- The codec itself imposes a very low delay of less than 3.5 ms within a field and 67 ms. for a GOP.

2. VIDEO STRUCTURE AND OVERALL ALGORITHM STRUCTURE

2.1 Video Structure

We assume that the incoming video has been digitized into 8-bit positive integers as follows:

- The frame size is 480x640 pixels and extends 33.3 ms.
- A frame consists of a sequence of two fields (interlace format) of 240x640 pixels each. A field extends 16.7 ms. in the digitized stream.
- A field consists of a sequence of 30stripes of 8x640 pixels each. A stripe extends 0.55 ms. Stripes are further subdivided into 20 blocks of 8x32 pixels each.
- Pixels are sequenced horizontally into scan lines, with the vertical direction being slower.
- Color is in the 4:2:2 format. Each pixel consists of an 8 bit positive integer (without sign) representing the B/W intensity value. Half of these pixels also have an 8-bit positive (without sign) chrominance value attached.

WZT processing is performed stripe by stripe. The processing of each stripe is divided into two passes. We therefore require two stripe buffers, one for each pass. The data is read into first pass stripe buffer and the first pass processing is performed on-the-fly as that data is read**.** The data in the first pass stripe buffer at the end of the stripe is the input data required by the second pass.

Concurrently, the data in the second pass stripe buffer is processed block-by-block. Some output FIFO is required to smooth the output and accomplish rate control. Each block is compressed as a separate entity.

At the end of the processing of each stripe the roles of the stripe buffers are swapped and the process is repeated.

The above process must be triplicated with three channels - one for B/W and two additional (quarter size) channels for the two chrominance streams.

If temporal compression of a video sequence of fields is desired, a third pass reads back the corresponding compressed blocks from each of the fields in a GOP, applies filters in the temporal direction, and recompresses the resulting block temporal subbands.

2.2 First pass – video scan and fine horizontal transform

The first pass accomplishes the following tasks:

- The original digital pixel data is read in video scan order
- The first two horizontal wavelet filters are applied (see description below). This processing is not blocked as in the second pass – edge filters are used only at the left and right edge of the frame.

2.3 Second pass – transform, quantize, and zero-tree code

The second pass processes the stripe block-by-block (8x32) and performs the following tasks on each block

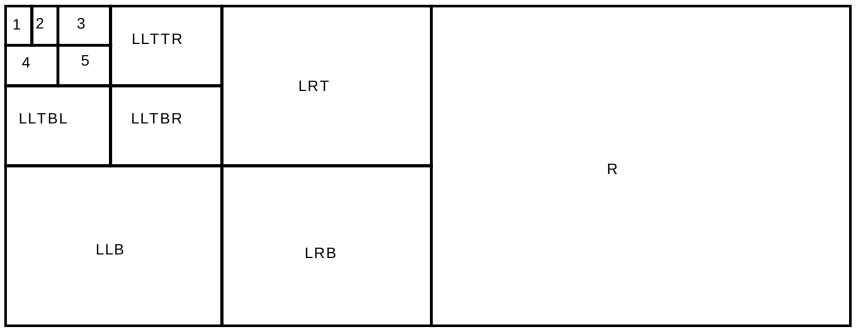
- The wavelet transform is applied, except for the first two horizontal filters which were applied in the first pass.
- The resulting wavelet coefficients are shifted, corresponding to a quantization division
- Each shifted coefficient must has its zero-tree significance value calculated
- The zero-tree is coded.

These tasks must be performed in sequence for each block. The calculation of the significance value requires referencing the significance values of some finer coefficients within the same block, but those will already have been calculated.

3. THE WAVELET TRANSFORM

3.1 The Mallat Pyramid

We restricted our attention to separable transforms that process the horizontal and vertical directions of the image independently. The wavelet transform step consists of building the Mallat pyramid [8]. The filter pair constructs a low pass and high pass transform (i.e., L, R), each of which is down-sampled by a factor of two to produce the same number of output coefficients as input coefficients. In 1-D this process is repeated on the *low-pass coefficients* (L) only, producing a pyramid in a number of steps twice the number of inputs. Here L stands for "left", R for "right", T for "top" and B for "bottom" in the pyramid structure.



1: LLTTLLTL (APEX) 2: LLTTLLTR 3: LLTTLRT 4: LLTTLLB 5: LLTTLRB
Figure 1. Mallat Pyramid (3x5) with 12 subbands

In 2-D, one application of the horizontal filters and of the vertical filters yields four bands (i.e., LT, RT, LB RB) each one quarter of the size. The Mallat pyramid is formed by repeating the process on the LT quadrant only. The entire pyramid is produced in a number of steps less than 4/3 the number of inputs.

We propose in Figure 1 a Mallat pyramid resulting from five filter applications in the horizontal direction and three applications in the vertical direction for each ($30x2^3$=240)x($20x2^5$=640) field. This results in 12 wavelet sub-bands. Since the filters are generally longer than two coefficients, the calculation of transform values near the boundary of the image would normally require image values outside the actual image itself. At higher levels of the pyramid these values extend exponentially farther beyond the image boundaries. Even in the interior of the image, filters longer that two coefficients imply looking ahead values a distance that is exponentially increasing in the pyramid level.

This problem is solved by using modified filters near the boundaries so as to utilize only image values. This technique can further simplify and reduce the processing if it is applied at block boundaries as well as at image boundaries. Therefore, the support of no filter crosses image boundaries and the support crosses no block boundaries except for the first two horizontal filters applied in the first pass.

3.2 The Mallat Block Pyramid

It is clear that the Mallat Pyramid resulting from the transform is actually a truncated pyramid that is an array of pyramids, one for each element in the scale (apex) subband LLTTLLTL. It is useful to decompose the Mallat Pyramid by partitioning the apex subband array into sub-arrays. For our purposes we partition the 30x20 apex subband array into 30 rows of 20 columns each. Each element is the apex of a pyramid whose base corresponds to a single block in the input. With the exception of the first two horizontal filters, all of the block pyramid values are generated internally to that block

3.3 The 2-6 transform

The actual transform filters are Haar filters lifted to have zero, one or two vanishing moments in the wavelet. This produces filters with dyadic coefficients, two coefficients in the low pass filter and two, four, or six coefficients in the wavelet filters.

The lifting can be performed so as to lift by scale values to the left, to the right, or to both sides of the lifted wavelet. The maximal filter which fits within the boundaries is actually selected.

The central quadratic (i.e., two vanishing moments) filters are a version of the 2-6 bi-orthogonal wavelet transform. This transform (also known as the TS-transform [2, 11]) is reported in the literature to be very effective for images, has very short filters, has coefficients that are dyadic rationals with small numerators, and has an effective two-stage factoring (i.e. *lifting*).

$$f_i = x_{2i} + x_{2i+1}$$
$$g_i = x_{2i} - x_{2i+1}$$
$$h_i = -\frac{f_{i-1}}{8} + g_i + \frac{f_{i+1}}{8}$$

2-6 forward transform

$$g_i = \frac{f_{i-1}}{8} + h_i - \frac{f_{i+1}}{8}$$
$$x_{2i} = \frac{f_i + g_i}{2}$$
$$x_{2i+1} = \frac{f_i - g_i}{2}$$

2-6 reverse transform

The 2-6 transform values (L and R) are denoted by f_i *and* h_i, and the signal values by $x_i\colon \; 0 \le i < I$. We have made the coefficients into dyadic rationals and eliminated the usual normalizing $\sqrt{2}$ by storing the transform values shifted by half a bit. This is compensated for in the reverse transform by the reverse shift of two half bits (i.e., the division by two implemented as a shift of one bit).

The reverse transform demonstrates that the original values can be recovered. Expanding those formulas we can compute the horizontal (scan line) and vertical coefficients at all levels and build the multi-level 2-D Mallat pyramid. An analysis of the pyramid determines the coefficient growth and the number of bits before and after the binary point that must be kept in binary fixed point calculations. The actual edge filters we used are included below.

3.3.1 Left and Right Edge Filters

We use specially designed edge filters that allow for fast processing and continuity across the block boundaries. In particular every field of 240x704 pixels is divided into 30x22 blocks of dimension 8x32 pixels each. The edge filters described below fit a quadratic to each block boundary from both left and right achieving 98% fit and virtually eliminating blocking artifacts. This structure also greatly reduces the memory bandwidth and vastly simplifies the data handling. As in the main filters, only adds and shifts are required to implement the filters below. The particular filters we designed are as follows:

$$f_0 = x_0 + x_1$$
$$g_0 = x_0 - x_1$$
$$h_0 = g_0 - \frac{3f_0}{8} + \frac{f_1}{2} - \frac{f_2}{8}$$

$$g_0 = h_0 + \frac{3f_0}{8} - \frac{f_1}{2} + \frac{f_2}{8}$$
$$x_0 = \frac{f_0 + g_0}{2}$$
$$x_1 = \frac{f_0 - g_0}{2}$$

Quadratic Left Edge Filter

$$f_N = x_{2N} + x_{2N+1}$$
$$g_N = x_{2N} - x_{2N+1}$$
$$h_N = g_N + \frac{f_{N-2}}{8} - \frac{f_{N-1}}{2} + \frac{3f_N}{8}$$

$$g_N = h_N - \frac{f_{N-2}}{8} + \frac{f_{N-1}}{2} - \frac{3f_N}{8}$$
$$x_{2N} = \frac{f_N + g_N}{2}$$
$$x_{2N+1} = \frac{f_N - g_N}{2}$$

Quadratic Right Edge Filter

$$f_0 = x_0 + x_1 \qquad g_0 = h_0 + \frac{f_0}{4} - \frac{f_1}{4}$$
$$g_0 = x_0 - x_1 \qquad x_0 = \frac{f_0 + g_0}{2}$$
$$h_0 = g_0 - \frac{f_0}{4} + \frac{f_1}{4} \qquad x_1 = \frac{f_0 - g_0}{2}$$

Linear Left Edge Filter

$$f_N = x_{2N} + x_{2N+1} \qquad g_N = h_N + \frac{f_{N-1}}{4} - \frac{f_N}{4}$$
$$g_N = x_{2N} - x_{2N+1} \qquad x_{2N} = \frac{f_N + g_N}{2}$$
$$h_N = g_N - \frac{f_{N-1}}{4} + \frac{f_N}{4} \qquad x_{2N+1} = \frac{f_N - g_N}{2}$$

Linear Right Edge Filter

$$f_0 = x_0 + x_1 \qquad g_0 = h_0$$
$$g_0 = x_0 - x_1 \qquad x_0 = \frac{f_0 + g_0}{2}$$
$$h_0 = g_0 \qquad x_1 = \frac{f_0 - g_0}{2}$$

Constant Left Edge Filter

$$g_N = h_N$$
$$f_N = x_{2N} + x_{2N+1} \qquad x_{2N} = \frac{f_N + g_N}{2}$$
$$g_N = x_{2N} - x_{2N+1}$$
$$h_N = g_N \qquad x_{2N+1} = \frac{f_N - g_N}{2}$$

Constant Right Edge Filter

3.4 Quantization

We can use quantization of the wavelet coefficients to achieve further improvement in compression. Accounting for the characteristics of the Human Visual System (HVS) we can also reduce significantly the number of bits used to code the chroma subbands.

Quantization is performed by dividing by a power of two (i.e., by shifting right). To minimize an L_2 norm, the transform coefficients should be divided by two for each factor of two finer in scale. To minimize an L_1 norm the transform coefficients should be divided by four for each factor of two in scale ([3]). Right shift implements dyadic quantization. Quantization shifts align coefficient bits of equal importance into the same bit planes and facilitates the progressive coding of the algorithm.

The normal viewing distance for NTSC TV is defined in the testing standards ([9]) to be 6 times the picture height. In our case the picture height is 480 lines = 480 pixels. Therefore $\sin^{-1}\left(1/6\right) \times 480 = 50$ pixels/degree. In the horizontal direction we have 640/480 = 4/3, exactly the picture ratio so that the pixels are square and we also have 50 pixels/degree in the horizontal direction.

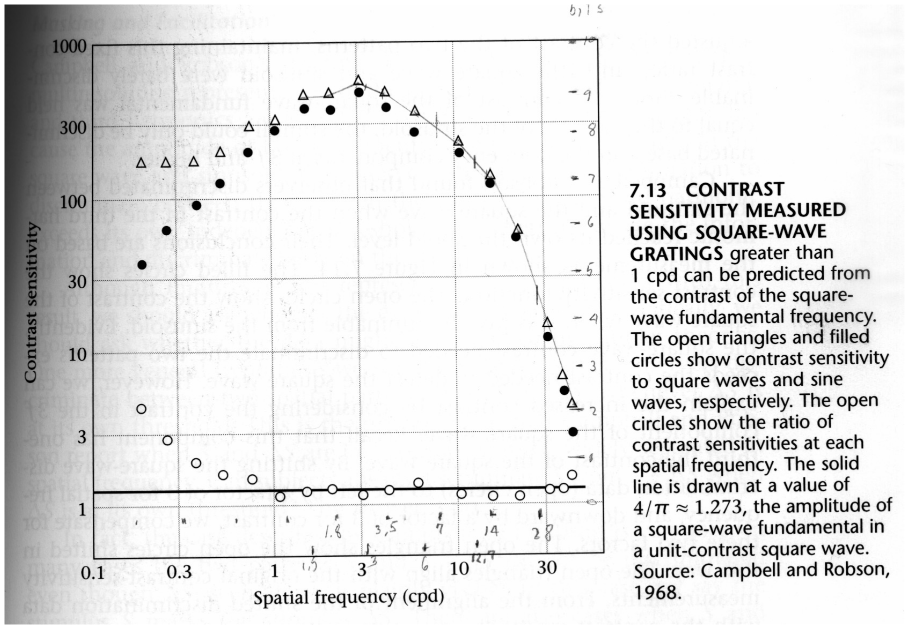
Subband	**LLT TLL TL**	**LLT TLL TR**	**LLT TLR T**	**LLT TLL B**	**LLT TLR B**	**LLT TR**	**LLT BL**	**LLT BR**	**LRT**	**LLB**	**LRB**	**R**
Horizontal cycles/deg.	**1.56**	**1.56**	**3.1**	**3.1**	**3.1**	**6.25**	**6.25**	**6.25**	**12.5**	**12.5**	**12.5**	**25**
Vertical cycles/deg.	**0.78**	**0.78**	**1,56**	**1.56**	**1.56**	**3.1**	**3.1**	**3.1**	**6.25**	**6.25**	**6.25**	**12.5**

Figure 2. Sub-band Visual Frequencies

The contrast sensitivity curve for human B/W vision peaks at 4 cycles per degree of viewing angle, where a cycle can be taken to be two pixels (Nyquist limit) in our case. At this spatial frequency humans can perceive 7-8 bits of B/W contrast. Contrast perception drops to nil at 40 cycles/degree, dropping by a factor of four for each doubling of the spatial frequency near the upper limit.

In Figure 2 sub-band 14 represents the contrast at 25 cycles / degree in the horizontal direction and 12 .5 cycles / degree in the vertical direction (the halving is due to our compressing fields, thus the vertical resolution is half). Figure 2 can be used to index the human visual system contrast sensitivity curve ([13]) and the corresponding contrast sensitivity read out. The $\log_2$ of this value, rounded to the nearest integer, gives the number of bits to keep after right shifting. After this quantizing right shift, coefficient bits occupying the same bit planes have about the same visual importance. (In this process, don't forget to account for the half bit per filter pair – one bit per pyramid level – that has been imposed by the removal of the normalizing square root of 2 from the filter coefficients.)

The test results we achieved quantizing to 0.5 bits/pixel resulted in 85% of the coefficients quantizing to zero. In performing the spatial quantization we used the curve from [13] reproduced here in Figure 3. For the color quantization we can use figure 3 from the color plate in [13] of the temporal vs. spatial frequency relationship.



7.13 CONTRAST SENSITIVITY MEASURED USING SQUARE-WAVE GRATINGS greater than 1 cpd can be predicted from the contrast of the square-wave fundamental frequency. The open triangles and filled circles show contrast sensitivity to square waves and sine waves, respectively. The open circles show the ratio of contrast sensitivities at each spatial frequency. The solid line is drawn at a value of $4/\pi \approx 1.273$, the amplitude of a square-wave fundamental in a unit-contrast square wave. Source: Campbell and Robson, 1968.

Figure 3. Contrast Sensitivity curve for the HVS.

3.5 Zero-Tree Organization

The Mallat pyramid can be organized into a *Zero-Tree*. The zerotree method has been used successfully in a number in the image compression literature (see [4, 5, 10 and 12]). The root of the zero-tree is in the apex sub-band LLTTLLTL and branches down and to the right.

• Coefficients in bands LLB, LRB, and R (on the right and bottom edges) have no children.

Subband	**LLTTLLTL**	**LLTTLLTR**	**LLTTLRT**	**LLTTLLB**	**LLTTLRB**	**LLTTR**	**LLTBL**	**LLTBR**	**LRT**	**LLB**	**LRB**	**R**
Parent Subband	**none**	**LLTTLLTL**	**LLTTLLTR**	**LLTTLLTL**	**LLTTLLTR**	**LLTTLRT**	**LLTTLLB**	**LLTTLRB**	**LLTTR**	**LLTBL**	**LLTBR**	**LRT**

Figure 4. Sub-band Parents

- Coefficients within a subband are assigned parents by taking the coefficient in the parent band by the usual proportionate scheme. The height and width of a subband's parent are divisors of the height and width of the subband.

We define the relation $u \succ v$

- $u \succ v$ iff v is a child of u

We define the relation $u \succ\succ v$ recursively as

- $u \succ\succ v$ iff $u \succ w$ and $w \succ\succ v$

Of course, the most important property of $\succ$ is that, for each block, it defines a tree - the zero-tree. The parent sub-bands for all sub-bands are depicted in Figure 4.

An important property of $\succ\succ$ and the zero-tree is that the forward transform that creates the Mallat pyramid computes the transform values of all v, $u \succ\succ v$ before calculating the value of u.

3.6 Coefficient Significances

The zero-tree coding executed in the second pass requires certain information about the forward transform coefficients. In particular we define

- $Sig(u) = \lfloor \log_2(u) \rfloor$ which computes the bit position of the leading bit of u
- $SigDecd(u) = \max\left(Sig(v) \ni u \succ\succ v\right)$
- $SigGDecd(u) = \max\left(Sig(v) \ni \exists w \ni u \succ w \wedge w \succ\succ v\right)$

$SigDecd(u)$ computes the position of the leading bit of the largest coefficient toward the leaves of the zero-tree from u (children and beyond).

$SigGDecd(u)$ computes the position of the leading bit of the largest coefficient toward the leaves of the zero-tree from u, excluding its children (grand-children and beyond).

There are useful recursive calculations of these functions, given $Sig(u)$. We have

- $SigDecd(u) = \max\left(SigGDecd(u), \max\limits_{v \in Child(u)}\left(Sig(v)\right)\right)$ and
- $SigGDecd(u) = \max\limits_{v \in Child(u)}\left(SigDecd(v)\right)$

These functions are calculated from the leaves of the zero-tree toward the root and can be carried out as each transform coefficient is calculated.

3.7 Zero Tree Coding

In this section we outline the algorithms for encoding the trees of transform coefficients generated by the previous steps of the algorithm. Inspired by the SPIHT algorithm ([10]), we have designed a tree walking procedure particularly appropriate for hardware implementation. The next two subsections contain the code for heavily recursive tree walks that perform the encoding and the decoding part of our zero-tree implementation. The apex (DC) coefficient of each Mallat block tree is called *Root*. Bit *n* of the absolute value of the wavelet coefficient at the zero-tree node *b* is denoted by $\gamma_b[n]$ whereas the sign of the coefficient at node be is denoted by $\gamma_b.sign$.

3.7.1 Encode (recursive tree walk)

```
Write SigDecd(Root) into the preamble
for each a pre-calculate Sig(a), SigDecd(a), and SigGDecd(a)
for each n from SigDecd(Root) step -1 until 0 do Walk(Root)

Boolean Inline Procedure Output(Boolean: $x$)
        This procedure writes the Boolean argument and also returns it as the result

Boolean Inline Procedure CodeOut($x$)
        if $x \le n$ then Output($x = n$)
        return $x \ge n$
```

```
**Procedure** Walk( $a$ )
    **for each** $b,\quad a \succ b$ **do if**
        **set** $u$ := (*Sig(b)* = n)
        **if** CodeOut(*Sig(b)*) **then** Output(**if** $u$ **then** $\gamma_b.sign$ **else** $\gamma_b[n]$ )
    **if** CodeOut(*SigGDecd(a)*) **then**
        **for each** $b,\quad a \succ b$ **do if** CodeOut(*SigDecd(b)*) **then** Walk( $b$ )
```

3.7.2 Decode (recursive tree walk)

```
Read *SigDecd(Root)* from the preamble
**for each** $a$ **initialize** *Sig(a), SigDecd(a),* and *SigGDecd(a)* **to** 0
**for each** $n$ **from** *SigDecd(Root)* **step** -1 **until** 0 **do** Walk(*Root*)
**Boolean Inline Procedure** Input
    This procedure reads and returns the next bit
**Boolean Inline Procedure** DecodeIn( $x$ )
    **if** $x \le n$ **then if** Input **then set** $x := n$
    **return** $x \ge n$
**Procedure** Walk( $a$ )
    **for each** $b,\quad a \succ b$ **do**
        **set** $u$ := (*Sig(b)* = 0)
        **if** DecodeIn(*Sig(b)*) **then set** (**if** $u$ **then** $\gamma_b.sign$ **else** $\gamma_b[n]$ ) := Input
    **if** DecodeIn(*SigGDecd(a)*) **then**
        **for each** $b,\quad a \succ b$ **do if** DecodeIn(*SigDecd(b)*) **then** Walk( $b$ )
```

We can also write those tree-walking procedure in a non-recursive structure which is easier to understand and in some cases better for implementation (see [3]).

4. THIRD PASS - TEMPORAL COMPRESSION

We have applied the same transform compression in the temporal direction to a GOP of four fields. A two level temporal Mallat pyramid is used as a tensor product with the spatial pyramid. The linear edge filters are used at the fine level and the modified Haar filters at the coarse level, resulting in four temporal subbands. Each of these temporal subbands is formed into a zero-tree and compressed as described in Section 3.

Temporal compression is best performed block-by-block so that it is necessary to read four corresponding blocks from each of the four fields and perform the temporal filtering and zero-tree compression. This also reduces the working storage required in the temporal phase.

Third pass processing for a GOP can be overlapped with the first and second pass processing for the following GOP.

In order to keep the RAM requirement from being dominated by the field storage, a transient and temporary zero-tree compression is performed block-by-block as a field is processed. The transient quantization is less severe than that used for the final quantization used in the temporal phase, aiming for a transient compression of a little more that 1 bit/pixel.. During the temporal phase the four incoming blocks are decompressed, processed, and recompressed forming a compressed zero-tree for each block-temporal subband.

In the MPEG terminology we are using a 4 fields (2 frames) GOP (group of pictures) structure. In that every field is temporarily compressed as a still image, thus requiring about 200 Kbits (= 1.18 bits/pixel). Afterwards all four fields are partially decompressed in parallel lockstep. In that the temporal linear filters are applied with serial-by-bit arithmetic and the result is recompressed. This temporary internal compression reduces memory, bandwidth and processing requirements. It also makes our short temporal filters an effective competition for the expensive motion estimation.

5. RESOURCE REQUIREMENTS

5.1 Memory Requirements

We can now make a rough estimate of the required RAM. RAM is required both for the stripe storage and for the storage of compressed blocks passed to the temporal phase. The RAM for luminance processing will then be increased by 50% to include the chrominance storage.

While the incoming digitized video consists of 8 bit values, the filters produce values with somewhat more bits. However the pyramid depth of the fields has been restricted so that all values will (just) fit into 16 bits. We will assume that all computed values require 16 bits of RAM.

Stripe storage requires

2 stripes x 8 lines/stripe x 640 pixels/line x 16 bits/pixel = 163840 bits

By creating a buffer pool and packing the compressed blocks into buffers, the block storage can be held to 4 fields while still being able to spread the processing throughout the next GOP of four fields. Compressed field storage thus requires

4 fields x 240 lines/field x 640 pixels/line x 1.5 bits/pixel = 921600 bits

for a total of 1085440 bits for the luminance channel. Here the substantial effect of field image compression can be seen.

For luminance and chrominance this totals up to 1628160 bits = 203520 bytes.

For working storage (often data cache memory) we require 1 block in pass 2 and 4 blocks in pass 3 for a total of 5 blocks. So we have

5 blocks x 256 coeff/block x 16 bits/coeff = 20480 bits = 2560 bytes

This amount of memory requires but a few mm^2 of silicon and can easily be integrated on the same chip as the processing.

5.2 Processing Requirements

The 2-6 transform requires 2 adds and one shift per pixel. For one level of horizontal and vertical processing this doubles to 4 adds and two shifts. The entire pyramid requires less than 4/3 times as much computation so that an image transform requires 16/3 adds and 8/3 shifts per pixel. Temporal processing requires at most another 4 adds and 2 shifts per pixel for an upper bound of 10 adds and 5 shifts per pixel.

A reasonable estimate for the hardware required to implement this with a slow clock rate of 1 clock per pixel time is 40K gates, equivalent to less than the space required for 1 million bits.

The overall conclusion is that it is reasonable to expect that this method can be implemented, including memory, in a few mm^2 of silicon.

6. SIMULATION RESULTS

6.1 Test Material

We have tested a software version of the above algorithm on several NTSC clips which vary in content and origin. For reference, these clips have also been processed by the commercial ADV601 wavelet codec from Analog Devices [1].

We experimented with test sequences that come with the test suite of ADV601, as well as with the video sequences used for evaluation the MPEG4 proposals. We also generated a number of test sequences from laserdisk (high quality material - zooming in bridge; panning of a fence with vertical bars and additional motion on the background; staircase sequences; involved dancers sequence), TV (NTSC cable programming of talk show sequence), satellite (basketball sequence which was previously MPEG compressed and decompressed), camcorder footage recorded in our building and video-game sequences (Nintendo games).

All various sequences were collected on a Betacam SP source and digitized in 8 bits/color ITU601 format at a professional outlet. The resulting digital tapes (in Abekas format) were read using Arriba Pro and saved in bitmap and targa formats. These sequences of frames were then processed with our codec to generate compressed and decompressed targa results. The same input sequences were processed with ADV601 software codec. The results were rendered using Sierra DDR (digital disk recorders) for visual as well as quantitative comparison (in terms of the peak signal-to-noise ratio PSNR). Figure 5 illustrates the PSNR results for two bit rates (corresponding to 11 Mb/s and 5.5 Mb/s resp.) for three video sequences. 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 last clip is a DSS (satellite) recording of a basketball game (already MPEG2 compressed/decompressed) with a lot of motion and detailed crowd and field.

6.2 Comparison of wavelet component codec vs. ADV601

The simulations we have performed demonstrate significantly better performance of WZT with respect to ADV601. In the experiments WZT outperforms ADV601 significantly both perceptually and in PSNR (by 1-2.3db).

The ADV601 was used for comparison purposes. This commercial codec uses the 7-9 Daubechies wavelets which require on the average 6 multiplications per wavelet coefficient for both encoding and decoding. For 480x640 video we need to perform 55 million multiplications per second. WZT has NO multiplies. In addition, the memory requirements of WZT are minimal compared to the full image storage (without temporal compression) required by ADV601 and other previous codecs.

	WZT at 1.0 bpp	ADV601 at 1.0 bpp	WZT at 0.5 bpp	ADV601 at 0.5 bpp
TalkShow	40.2 dB	38.4 dB	36.2 dB	33.9 dB
Fence	30.6 dB	30.8 dB	27.1 dB	26.5 dB
Basketball	28.5 dB	27.5 dB	25.0 dB	22.8 dB

Figure 5. PSNR rates - bit rate vs. video clips (basketball, fence, talkshow)

6.3 Comparison of wavelet composite codec vs. PixelTools' MPEG2 codec

We have also preformed extensive comparison of our wavelet video codec with a commercial MPEG2 codec. In those comparisons we used our more recent composite wavelet codec which incorporates color conversion as part of the codec (as described in Section 3). In order to build a composite source sequences, we used specialized hardware to generate D2 composite frames from the targa frames by modulating the chroma and the luma components (i.e. chroma 1 and chroma 2 are quadrature modulated on a 3.58 MHz sub-carrier).

The corresponding composite 2-6 wavelet pyramid includes two-level Haar filters in the time direction on a 4 field GOP. The pyramid is depicted in Figure 6 and in effect processes the luma in the top left corner of the pyramid. Chroma 1 and chroma 2 are processed in the bottom middle and right hand detailed branches. We have performed again dyadic quantization with coefficients being powers of 2.

The entropy coding for that experiment used separate arithmetic coder for each separate bitplane of the coefficients.

Overall in comparison to MPEG2 we are using:

- 2-6 wavelet pyramid and reverse wavelet pyramid as opposed to DCT (discrete cosine transform) and inverse DCT.

- Temporal wavelet and reverse temporal wavelet instead of Motion Estimation and Motion Compensation.

- Aggressive but simple quantization with no multiplications (as opposed to MPEG).

- Different entropy coding methods - zerotree (described in Section 3), arithmetic or z-coding (see [6] and [7]) vs. Huffman coding (typically used in MPEG).

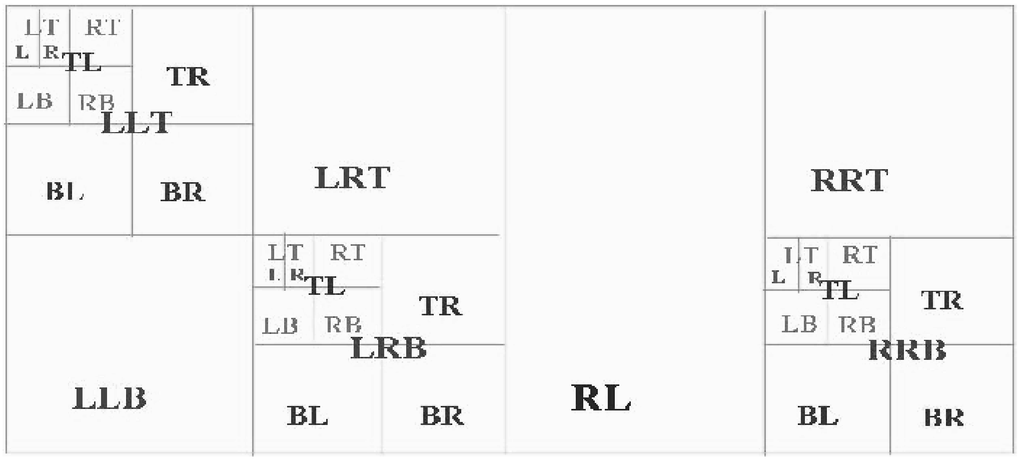


Figure 6. Composite Wavelet 3-D pyramid structure

The results of applying our codec and MPEG2 for the sequences described earlier in the Section are summarized in Figure 7. The particular MPEG2 codec that we used in the experiments is MPEGRepair - a commercially available software codec from PixelTools Inc. which is commonly used for high quality broadcast encoding. All sequences were compressed using (the standard) 15 GOP structure with two B frames between the I and P frames. We used medium search range for the motion estimation (recommended for reasonable running time of the algorithm) as well as 16x16 field and frame prediction range. In addition we used double DCT precision and fixed bit rate structure.

	WZD 1.0 bpp	**MPEG2** 1.0 bpp	**WZD** 0.5 bpp	**MPEG2** 0.5 bpp
TalkShow	36.10 dB	37.03 dB	34.30 dB	34.85 dB
Fence	31.27 dB	29.62 dB	27.13 dB	26.00 dB
Basketball	28.33 dB	31.69 dB	25.33 dB	28.42 dB
Bridge	39.86 dB	39.35 dB	36.85 dB	37.34 dB

Figure 7. Comparing WZD and PixelTools software MPEG2 coder.

As we can see from the results, on the average, our algorithm performed as well as MPEG2 in terms of PSNR (as well as visual perception quality) for most of the tested sequences. While for the basketball sequence MPEG2 outperformed our codec, for the fence sequence we did significantly better. What is importance for us is that the proposed coder is several orders of magnitude simpler than MPEG. In particular the software version of our non-optimized code ran 4 times faster on encoding than the highly optimized commercial MPEG2 codec. With optimization and restructuring of the code we can easily achieve at least 20 times faster coder than MPEG2. In hardware that advantage translates into much cheaper implementation in terms of number of gates and complexity. In particular based on our extensive estimates, the full encoder should fit into not more than 60K gates (a conservative estimate). In addition as outlined earlier in the paper our codec is completely symmetric in terms of encoding vs. decoding.

Clearly we can adjust the parameters in the MPEG2 codec to improve the quality of the fence sequence compression. In particular if we use just a 2 frames GOP with only I and P frames, we can get close to the results achieved by our coder. However in hardware implementation that will require a high quality 2-pass encoder. There on the first pass the input sequence is analyzed and the appropriate running parameters are computed. The second pass does the actual encoding. Such encoders are used in the broadcasting industry and are generally very expensive (tens to hundreds of thousands of dollars) and correspondingly complex. Thus they would be completely inadequate for consumer applications, the area of interest that we are targeting.

7. CONCLUSION

This paper introduces a very low cost video wavelet codec that achieves comparable or better performance than commercially available MPEG2 and wavelet codec. The simplicity of the coder comes from the choice of wavelet filters (implemented with shifts and adds only), the use of simple temporal wavelets instead of motion estimation, and the use of temporary compression of the intermediate results. The coder uses a 2 frame (4 field) GOP structure and performs the color conversion as part as of the codec implementation.

For a comparable performance in software, the proposed coder runs 20 or more orders of magnitude faster than commercial MPEG2. In hardware, the encoder can be implemented on an ASIC in less than 60K gates. In 0.35m technology the anticipated cost in SRAM is sub $2 for the chip, with opportunities of packaging the implementation in 1 Mbit of DRAM for a sub $1 cost.

REFERENCES

[1] Analog Devices ADV601 Low Cost Multiformat Video Codec, *ADV601 Preliminary Data Sheet*, January 1996.

[2] M. Boliek and A. Zandi (RICOH California Research Center), Coding of Still Pictures, *CREW Lossless/Lossy Image Compression Contribution to ISO/IEC JTC 1.29.12*, July 1995.

[3] K. Kolarov and W. Lynch, Compression of Functions Defined on Surfaces of 3D Objects, In J. Storer and M. Cohn, editors, Proc. Of Data Compression Conference, IEEE Computer Society Press, 1997.

[4] A. Lewis and G. Knowles, A 64Kb/s Video Codec using the 2-d Wavelet Transform, *Proceedings of the Data Compression Conference*, Snowbird, Utah, March 1991.

[5] A. Lewis and G. Knowles, Image Compression Using the 2-D Wavelet Transform, *IEEE Transactions on Image Processing, Vol. 1, No. 2*, April 1992.

[6] W. Lynch, K.Kolarov and W. Arrighi, Low Cost Video Compression Using Fast Modified Wavelet Pyramids, *Proceedings of the International Conference on Image Processing* ICIP'99, October 99.

[7] W. Lynch, K.Kolarov, W. Arrighi and R. Hoover, WZD - A Very Low Complexity Wavelet Video Codec for and ASIC Macro, *Proceedings of the Picture Coding Symposium* PCS'99, April 99, pp. 355-358.

[8] 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.

[9] C. Poynton, A Technical Introduction to Digital Video, *John Wiley & Sons, Inc*. 1996.

[10] A. Said and W. Pearlman, A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees, *IEEE Transactions on Circuits and Systems for Video Technology 6:243-250*, June 1996.

[11] 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.

[12] J. Shapiro, Embedded Image Coding Using Zerotrees of Wavelet Coefficients, *IEEE Transactions on Signal Processing, Vol. 41, No. 12*, December 1993.

[13] B. Wandell, Foundations of Vision, *Sinauer Associates, Inc. Publishers*, 1995.