Real-Time Layered Video Compression using SIMD Computation

Morten Vadskær Jensen and Brian Nielsen

Aalborg University
Department of Computer Science
Fredrik Bajersvej 7E
DK-9220 Aalborg, Denmark
{mvj | bnielsen}@cs.auc.dk

Abstract. We present the design and implementation of a high performance software layered video codec, designed for deployment in bandwidth heterogeneous networks. The codec facilitates layered spatial and SNR (signal-to-noise ratio) coding for bit-rate adaption to a wide range of receiver capabilities. The codec uses a wavelet subband decomposition for spatial layering and a discrete cosine transform combined with repeated quantization for SNR layering.
Through the use of the Visual Instruction Set on SUN's UltraSPARC platform we demonstrate how SIMD parallel image processing enables layered real-time software encoding and decoding. The codec partitions our $384 \times 320 \times 24$-bit test video stream into 21 layers at a speed of 39 frames per second and reconstructed at 28 frames per second. The Visual Instruction Set accelerated encoder stages are about 3-4 times as fast as an optimized C version. We find that this speedup is well worth the extra implementation effort.

1 Introduction

Smart technology for multicasting live digital video across wide area networks is necessary for future applications like video conferencing, distance learning, and tele-commuting. The Internet Multicast Backbone (MBone) [1] is already popular and allows people from anywhere on the planet to exchange modest quality video and audio signals. However, a fundamental problem with nearly all large computer networks is that network capacity (bandwidth) varies extremely from one network segment to another. This variation is the source of a serious video multicast problem. All users wish to participate in the video conference with the highest quality video their connection capacity (varying between 128kbps for ISDN dialup connections to 100 Mbits LAN connections) and host computational resources allow. High capacity receivers are capable of processing good quality video with bit-rates of 4Mbps or more, but this outperforms the low capacity receivers by more than an order of magnitude. Even a modest bit rate may overflood low capacity receivers.

Conventional video compression techniques code the video signal to a fixed target bit rate, and is then multicasted to all receivers. A low target bit rate is typically chosen to enable as many receivers as possible to participate. However, the resulting quality is unacceptable for the high capacity receivers. Alternatively, the video signal can be encoded repeatedly to a multitude of data rates. This is, however, very inefficient: Firstly, it involves extra computational overhead from compressing the same frame repeatedly. Secondly, it introduces bandwidth redundancy as each high bit-rate video stream contains all the information also included in the lower bit-rate ones. This translates to inefficient bandwidth utilization from the root of the the multi-cast tree.

We propose *layered coding* coupled with multi-casting, where the video stream is coded and compressed into several distinct layers of significance as shown in Figure 1. In principle, each layer is transmitted on its own multi-cast channel. The receivers may subscribe to these channels according to their capabilities and preferences. The more layers received, the higher video quality, but also higher bandwidth and processing power requirements. The most significant layer constitutes the *base layer* and the following layers *enhancement layers*.

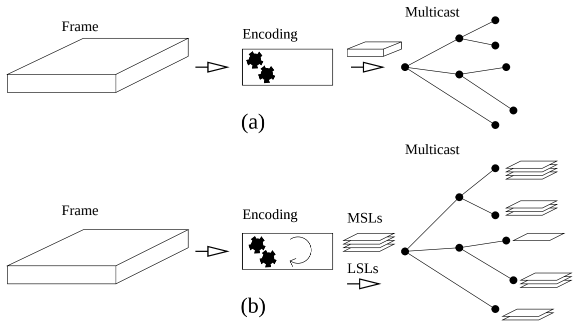


Fig. 1. Conventional single bit-rate video coding (a) vs. layered coding (b).

Live video requires that video frames are encoded and decoded in real-time. This is even more challenging for layered coding than for conventional coding because layer construction and reassembly requires use of additional image filter functions and repeated processing of image data, and hence requires more CPU-processing. We believe that it is important that this coding is possible in real-time on modern general purpose processors without dedicated external codec hardware—partly because no current hardware unit has the functionality we advertise for, but more importantly, because applications in the near future will integrate video as a normal data type and manipulate it in application dependent manner. This require significant flexibility. Fortunately, most modern CPU-architectures have been extended with SIMD instructions to speed up digital signal processing. Examples include VIS in Sun's UltraSPARC, MMX in Intel's Pentium(II), MAX-2 in Hewlett-Packards PA-RISC, MDMX in Silicon Graphics' MIPS, MVI in Digital's Alpha, and, recently, Altivec in Motorola's PowerPC CPUs.

In this paper we show a high performance implementation of a layered codec capable of constructing a large set of layers from reasonably sized test-video in real-time. We demonstrate how SIMD parallelism and careful considerations of superscalar processing can speedup image processing significantly. Our encoder implementation exists in both an optimized C-version and in a version almost exclusively using SUN microsystem's Visual Instruction Set (VIS) available on the SUN UltraSPARC platform. The decoder only exists in a VIS-accelerated version.

The remainder of the paper is structured as follows. Section 2 presents our layered coding scheme and our codec model which incorporates it. Section 3 presents the implementation of an instance of our codec model along with our performance optimization efforts. Section 4 evaluates the coding scheme through a series of measurements. Finally, section 5 discusses related work, and section 6 concludes.

2 The Codec Model

Our codec combines two independent layering facilities: spatial layering and signal-noise-ratio (SNR) layering. Spatial layering controls the resolution of the image—subscription to more spatial layers means higher image resolution. SNR layering controls the number of bits used to represent image pixels—the more bits the higher pixel precision, and thus a more sharp and detailed image.

The wavelet filter performs a subband decomposition of the image and decomposes it into one low (L) and one high frequency (H) subband per dimension. The effect of subband decomposition on the Lena-image is depicted in Figure 2. The LL subband is a subsampled and filtered version of the original image. The three subbands, HL, LH, and HH contain the high frequency information necessary to reconstruct the horizontal, vertical, and diagonal resolution, respectively. Therefore, the HL, LH, and HH layers act as refinement (or enhancement) layers for the LL layer. Further, the LL subband can itself be subject to further subband decompositions, or can be subjected to the familiar block-based DCT coding scheme known from other codecs for further psycho-visual enhanced compression. We use a 2D-version of the 4-tap 1-3-3-1 spline wavelet [5, p. 136], because it achieves *good pixel value approximation* and because it is highly summetric and consequently can be implemented very efficiently.

SNR layering divides the incoming coefficients into several levels of significance, whereby the lower levels include the most significant information, resulting in a coarse image representation. This is then progressively refined with the number of layers. SNR layering is a flexible and computationally efficient way of reducing the video stream bit-rate, and allows fine-grain control of image quality at different layers. In our codec SNR-layering is achieved through a DCT-transform combined with repeated uniform quantization.

Figure 3 shows our codec model. Encoding is performed by the following components: The *Wavelet Transformer* transforms YUV images into LL, HL, LH, and HH layers corresponding to four spatial layers. The enhancement layers HL, LH, HH are sent directly to the quantizer, while the LL layer is passed on to the DCT stage. The *DCT transformer* performs the Discrete Cosine Transform on the LL image from above. The resulting coefficient blocks are passed on to the

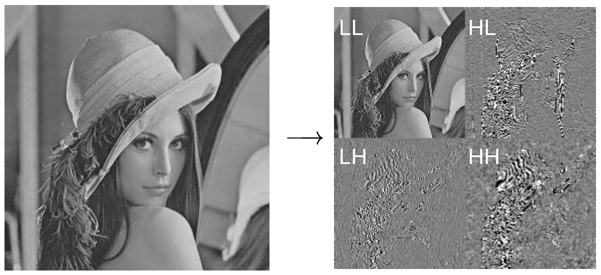


Fig. 2. Wavelet transformation of the Lena image into LL, HL, LH, HH layers. The LL layer is a downsampled version of the original image and the HL, LH, and HH layers contain the information required to restore horizontal, vertical, and diagonal resolution, respectively.

quantization stage. The *Quantizer* splits incoming coefficients into several layers of significance. The base layer contains a coarse version of the coefficients, and each layer adds precision to coefficient resolution. The *Coder* takes the quantized coefficients from each separate layer and Huffman compresses them to produce a single bit-stream for each layer.

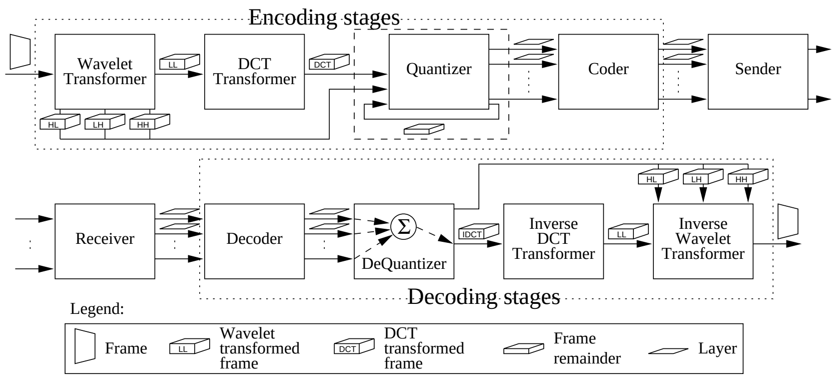


Fig. 3. Our codec model. In the encoding stages, the wavelet transformer sends the LL layer to the DCT stage before all layers go through the quantizer and on to transmission. The quantizer reuses coefficient remainders, as depicted by the arrow reentering the quantizer.

The decoding stages essentially perform the inverse operations. The *Decoder* uncompresses the received Huffman codes. The *Dequantizer* reconstructs coefficients based on the decoded layers; the dequantizer works both as coefficient reconstructor and as layer merger. Like the encoder, there are two different dequantizers, one for DCT coefficients, which are passed on to the IDCT stage, and one for wavelet coefficients which are passed directly on to the wavelet reconstruction stage. The *Inverse DCT Transformer* performs IDCT on incoming DCT coefficient blocks, which reconstructs the lowest resolution spatial layer, LL. The LL layer is used in the *Inverse Wavelet Transformer*, which performs wavelet reconstruction.

The *Sender* and *Receiver* modules, which are outside the actual codec model, must support multi-casting for efficient bandwidth utilization. The layers are transmitted to different multi-cast groups, so receivers can adjust their reception rate by joining and leaving multi-cast groups. Further details on this design can be found in [4].

Our design does not offer temporal layering besides simply dropping frames. The most effective temporal compression schemes use motion compensation, but this does not in it self add any layering, only a lower bit rate. Our goal is to provide high bandwidth versatility. Moreover, it is untrivial to obtain scalability from a motion compensated stream, because frames are interdependent. That is, both the desired frame and the frames it depends on must be received. We have therefore decided to postpone temporal motion compensated layering to future work.

3 Implementation

The visual instruction set found on SUN's UltraSPARC CPUs [10] is capable of processing 8×8 bit, 4×16 bit, or 2×32 bit partitioned data elements in parallel. In addition to the usual SIMD multiplication and addition instructions VIS contains various special instructions dedicated to video compression and manipulation of 2-3 dimensional data. Furthermore, the UltraSPARC CPU has two pipelines and is therefore able to execute pairs of VIS instructions in parallel [7].

The VIS instructions are available to the application programmer through a set of macros which can be used from C-programs with a reasonable amount og effort, although it must be realized that even with these C-macros, programming VIS is essentially at the assembler level. The programmer is, however, alleviated from certain aspects of register allocation and instruction scheduling.

Our implementation is optimized at the architecture level as well as at the algorithmic level. The architecture level optimizations fall in three categories: 1) using the VIS to achieve SIMD parallel computation of 4 data elements per instruction, 2) using super scalar processing to execute two (independent) instructions in parallel per clock cycle, and 3) reducing memory access by keeping constants and input data in registers, and by using the 64-bit load/store capabilities. These optimization principles are applied in all stages of the codec. Below we exemplify these on the layer decomposition stage.

The the decomposition filter is applied by multiplying each pixel in a 4×4 input block with the corresponding element in the filter coefficient matrix. The 16 results are then summed into a single value and divided by 16 to produce an "average", which is the final output. This is done for each layer type. The filter dictates a two-pixel overlap between the current and the next input block, which therefore becomes the image data starting two pixels to the right relative to the current block. The implementation of the decomposition routine operates on 8×4 pixel blocks at a time, see Figure 4, producing 4 32-bit outputs; one for each of the LL, HL, LH, HH layers. It uses 64-bit loads to load input data, and since all data fits in registers, it requires only 4 64-bit loads and 4 32-bit stores pr. pixel block. The routine spills overlapping pixels from one 8×4 block to the next, so horizontal traversal causes no redundant memory accesses. Vertically there is a 2 pixel overlap, dictated by the filter, so each pair of vertical lines is read twice.

By utilizing the VIS `fmul8x16` instruction, which multiplies four 16-bit values with four 8 bit values producing four 16 bit results, the four pixels in row 1 in block 1 can be multiplied with row 1 of the filter coefficients in parallel. Similarly, row 2 in group 1 can be SIMD-multiplied with row 2 of the filter coefficients. Moreover, both of these multiplications can execute in parallel: The super scalar processing in the UltraSPARC CPU allows execution of two independent VIS instructions in parallel per clock cycle. Two instructions are independent if the destination of one instruction is different from the source of the next instruction, as is indeed the case here.

The UltraSPARC CPU is incapable of out-of-order execution, so instructions must be carefully scheduled, either manually or by the compiler, to fully exploit instruction independence. However, we found that the compiler supplied with the platform did not do a satisfactory job on this point and we have therefore manually organized the VIS-code to pair independent instructions, and thereby maximize the benefit from super scalar processing. Thus, SIMD parallelism reduces the number of calculations by three quarters, and super scalar parallelism further halves this number.

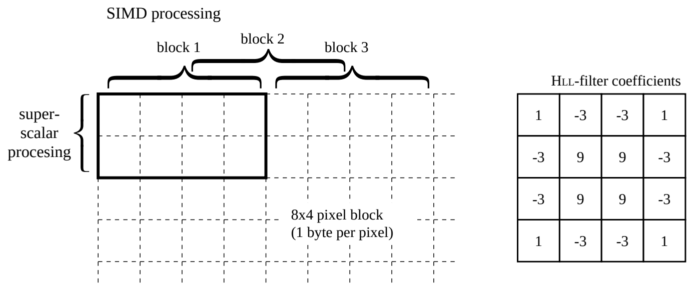


Fig. 4. Utilizing SIMD and super scalar processing. SIMD instructions allow 4 elements in the same row to be multiplied with the corresponding filter coefficient row in parallel. Super scalar processing enables two rows in the same block to be computed in parallel. Thus the 8 enclosed pixels are processed in parallel.

At the algorithmic level we exploit the fact that the filter coefficients for the four filters are highly symmetric. This permits computation of the LH-block after the LL-block has been computed through simple sums and differences of data already cached in CPU-registers, that is, without the need for multiplication of a new set of filter coefficients. Similarly, the HH-block can be derived from the HL-block. Therefore, all four types of data blocks are constructed from a single scan of each YUV picture components (Y is the luminance component, and U, V are chrominance components).

With these optimizations, the implementation uses only 0.505 memory accesses per pixel for creating all 4 wavelet layers, and a maximum of 0.344 memory accesses per pixel per reconstructed layer.

4 Performance Measurements

The input is a test video stream with 174 frames of 384×320 pixels at 24-bit colour, digitized at 18 frames per second. The video stream is a typical "talking head" sequence, with one person talking and two persons entering and leaving the image as background movement. The CPU-usage was measured on a Sun Ultra Creator-1 machine with one 167MHz UltraSPARC CPU and 128MB RAM. The test video stream is stored in YUV format, so the colour space conversion stage is not included in the encoding CPU-usage measurements. Likewise the colour space conversion and display times for the decoder are not included as they combined take a nearly constant 13ms.

Our codec configuration uses one level of subband decomposition, and has a total of 21 layers as seen in Figure 5. Layer 0 is the most significant, down to layer 21 as the least significant. The first 12 layers, DCT_0 - DCT_{11}, are DCT coded layers constructed from the wavelet LL layer. The remaining 9 layers, HL_0 - HL_2, LH_0 - LH_2, HH_0 - HH_2, are constructed from the corresponding wavelet enhancement layers, distributed with 3 SNR layers from each. The 3 most significant SNR layers, level 0, consist of the upper 4 bits of each coefficient from each of the wavelet layers. Layers from levels 1 and 2 consist of 2 additional bits per coefficient per layer. All three levels thus add $4+2+2=8$ bits of precision. The most significant layers, i.e. those on level 0, are Huffman coded. The layers on levels 1 and 2 are sent verbatim because their contents are very random and Huffman coding would add very little, if any, compression.

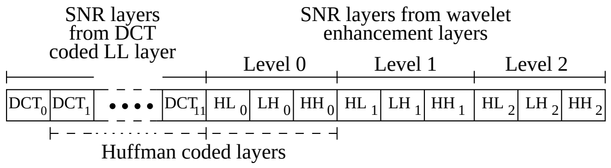
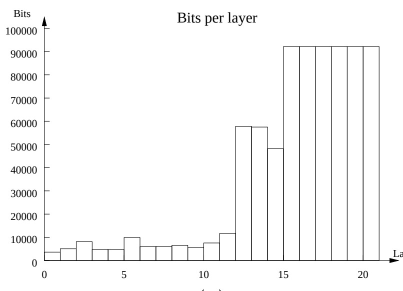


Fig. 5. The layer definitions used for testing along with their significance ordering. DCT_0-DCT_{11} are layers constructed from the DCT coded LL layer. HL_0, LH_0, and HH_0 contain the most significant bits from the corresponding wavelet enhancement layers. Levels 1 and 2 are refinement layers to these.

The resulting accumulated bitrates are evenly distributed between 3.5kbits with one layer and 777.3kbits with all 21 layers as shown in Figure 6. The lowest number of layers necessary to produce clearly recognizable persons in the test-movie was found to be 3 (DCT_0–DCT_2), corresponding to an average frame size of 16.4kbits. This corresponds to a factor 47.4 in difference between lowest capacity receiver and highest capacity receiver—a significant variation. Sample images can be found on the World Wide Web `http://www.cs.auc.dk/~bnielsen/codec/`



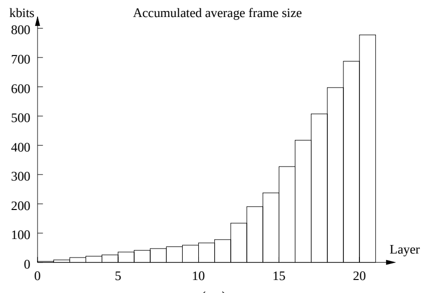


Fig. 6. Bandwidth distribution on layers on the 21-layer codec. (a) Average size of the individual layers. (b) Accumulated frame size versus number of added layers.

The CPU-usage measurements show average encoding times (for both the C- and VIS-version), average decoding times (for the VIS-version), and identifies the cost distribution between all codec-stages. The results are summarized in Table 1. On average, the encoder uses 25.4ms to construct and compress all 21 layers. This enables the encoder to process 39 frames per second, which is more than fast enough for real-time software encoding. Similarly, the total average decoding time is 34.81ms or 28fps. An additional 13ms is required for color space conversion and display drawing, but this still allows real-time decoding and display of the test video stream. Further, in most real-life applications, one would rarely reconstruct all 21 layers, meaning even faster decoding times.

Table 1. Average encoding and decoding time. The Wavelet Transform produces all 4 wavelet layers, the DCT transforms the LL layer, and the Quant & Huff (DCT/wavelet) stages quantize and code the 9 output wavelet enhancement layers, and the 12 DCT SNR layers, respectively. The DeQuant & UnHuff (DCT) stage decodes all 12 SNR layers. DeQuant & UnHuff(wavelet) decodes the 9 wavelet enhancement layers. Wavelet reconstruction upscales the LL image, and adds resolution by reconstructing the HL, LH, and HH layers.

Encoding time (ms)			Decoding time (ms)	
	VIS	C		VIS
Wavelet Transform	7.89	28.15	DeQnt&UnHuff(DCT)[‡]	4.58
Quant & Huff (wavelet)[‡]	7.62	9.11	IDCT[†]	2.13
DCT[†]	2.51	8.51	DeQnt&UnHuff(wavelet)[‡]	14.94
Quant & Huff (DCT)	7.38	22.01	Wavelet reconstruction	13.36
Total	25.40	67.78		35.01

[†] The C and VIS versions of DCT and IDCT functions are from SUN's MediaLib graphics library[9].
[‡] Only the C version exists of the (de)quantization and huffman (de)coding stages.

Decoding appears more expensive than encoding, which is unusual. However, the reason for this is that the decoder is designed to be very flexible with respect to which frames it decodes, and in what order it does so. It permits seperate decoding of each layer which require a separate iteration across the image per layer. This makes it easier to change the set of layered subscribed to dynamically, e.g., to reduce congestion. Also decoding can begin as soon as a layer is received. Finally note that two of the stages are not VIS-accellerated.

The SIMD implementation provides a significant performance improvement. For comparison, an otherwise identical, efficient and compiler optimized non-VIS accelerated C-implementation of the encoder is capable of encoding only 14.8 fps. A more detailed inspection of the individual stages reveals that VIS provides speedups in the range 3-4 for these particular algorithms: The C-version of wavelet transformation takes 28.15ms on average as opposed to the 7.89ms needed by the VIS-version. This yields a speedup factor of 3.6. The speed of quantization and Huffman coding of the DCT coefficients is increased by a factor 3. Also, we measure a speedup of similar magnitude, factor 3.4, for SUN's MediaLib [9] implementation of DCT. The overall effect of VIS acceleration is that real-time coding becomes possible—with time to spare for network communication, display updates and other application processing.

5 Related Work

Layered video coding and related network design issues are active and recent research areas. Related work on layered video codecs exist, notably [3], [6]. The codec in [6] resembles our codec in that they use wavelets for subband decomposition and DCT coding for the LL wavelet layer. Their design does not like ours allow for several levels of subband decomposition. They include temporal layering in the codec using conditional block replenishment. The authors stress the need for error resilient coding mechanisms for error prone networks such as the Internet. Their receiver-driven layered multicast (RLM) scheme, where receivers adjust their reception rate by joining and leaving multi-cast groups, was developed to work in environments like the MBone [1] using IP multi-casting. Although suggestions for implementation optimizations are presented in the paper, they present very little information about run-time performance.

MPEG-2 was the first standard to incorporate a form of layering, called *scalability*, as part of the standard [2]. But MPEG-2 is intended for higher bit-rate video streams, and therefore only allows for three enhancements to the base layer; one from each of scalability category: spatial, SNR, and temporal scalability. Also, no MPEG-2 implementation exists that includes the scalabilities. A combination of MPEG and wavelet based spatial layering for very high bandwidth video is proposed in [11]. The video is repeatedly downsampled until the resolution reaches the resolution of the common intermediate format (cif). The cif-sized LL^n-layer is then further processed by an MPEG based codec. Their proposal also offer hierarchical motion compensation of the high frequency subbands, but not temporal scalability.

Compression performance of our codec may be improved by using other methods than Huffman compression. One of the most efficient methods is the embedded zerotree wavelet coding [8], but it is most effective when using several levels of decomposition.

6 Conclusions

This paper addresses the problem of efficiently distributing a video stream to a number of receivers on bandwidth heterogeneous networks. We propose layered coding and multi-cast distribution of the video. We design a proprietary video codec incorporating wavelet filtering for spatial layering and repeated quantization for SNR layering. We contribute with a high performance implementation using the SIMD capabilities of our platform. Our measurements show that the codec is capable of real-time software encoding a test movie at 39fps and decoding at 28fps. We found that the Visual Instruction Set accelerated stages in the codec was 3-4 times as fast as an otherwise identical, efficient C-implementation; we therefore find SIMD acceleration to be worth the extra implementation effort.

As this work was carried out as part of a master's thesis, our experience suggest that the new media instructions can be successfully applied by programmers without years of signal processing experience. Indeed, in several cases, the implementation was in fact simplified by using SIMD, although it requires a different line of thought. Since SIMD is integrated into virtually all modern CPU-architectures, we think that developers should consider using it, given the possible speedups. Unfortunately, the SIMD engines are not compatible between CPU-architectures.

References

1. Hans Eriksson. MBONE: The Multicast Backbone. *Communications of the ACM*, 37(8), 1994.
2. Barry G. Haskel, Atul Puri, and Arun N. Netravali. *Digital Video: An Introduction to MPEG-2*. Chapman and Hall, 1997.
3. Klaus Illgner and Frank Müller. Spatially Scalable Video Compression Employing Resolution Pyramids. *IEEE Journal on Selected Areas in Communications*, 15(9):1688–1703, December 1997.
4. Morten Vadskær Jensen and Brian Nielsen. Design and Implementation of an Efficient, Layered Video Codec. Technical Report R-98-5008, Aalborg University, Dept. of Computer Science, 9220 Aalborg SØ, Denmark, August 1998.
5. Martin Vetterli and Jelena Kovačević. *Wavelets and Subband Coding*. Prentice Hall, 1995.
6. Steven McCanne, Martin Vetterli, and Van Jacobson. Low-Complexity Video Coding for Receiver-Driven Multicast. *IEEE Journal on Selected Areas in Communications*, 15(6):983–1001, August 1997.
7. Sun Microsystems. UltraSPARC and New-Media Support, 1995. WPR-95-028.
8. Jerome M. Shapiro. Embedded Image Coding Using Zerotrees of Wavelet Coefficients. *IEEE Trans. on Signal Processing*, 41(12):3445–3462, December 1993.
9. Sun Microsystems. *mediaLib Users Guide*, June 1997. `http://www.sun.com/microelectronics/vis/mlib_guide.pdf`.
10. Marc Tremblay, J. Michael O'Connor, V. Narayanan, and Liang He. VIS Speeds New Media Processing. *IEEE Micro*, 16(4):10–20, August 1996.
11. Qi Wang and Mohammed Ghanbari. Scalable Coding of Very High Resolution Video Using the Virtual Zerotree. *IEEE Transactions on Circuits and Systems for Video Technology*, 7:719–727, October 1997.