Design and Implementation of an Efficient, Layered Video Codec for Heterogeneous Networks

by

Morten V. Jensen and Brian Nielsen

R-98-5008 November 17, 1998

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF ELECTRONIC SYSTEMS
Fredrik Bajers Vej 7E ▪ DK 9220 Aalborg Ø ▪ Denmark
Phone: +45 96 35 80 80 ▪ Fax: +45 98 15 98 89
URL: http://www.cs.auc.dk

Design and Implementation of an Efficient, Layered Video Codec for Heterogeneous Networks

Morten V. Jensen and Brian Nielsen*

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

November 17, 1998

Abstract

We present the design and implementation of a high performance layered video codec, designed for deployment in bandwidth heterogeneous networks. The codec combines wavelet based subband decomposition and discrete cosine transforms to facilitate layered spatial and SNR (signal-to-noise ratio) coding for bit-rate adaption to a wide range of receiver capabilities. We show how a test video stream can be partitioned into several distinct layers of increasing visual quality and bandwidth requirements, with the difference between highest and lowest requirement being $47:1$.

Through the use of the Visual Instruction Set on SUN's UltraSPARC platform we demonstrate how SIMD parallel image processing enables real-time layered encoding and decoding in software. Our $384 \times 320 \times 24$*-bit test video stream is partitioned into 21 layers at a speed of 39 frames per second and reconstructed at 28 frames per second. Our VIS 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

Distribution of live digital video on wide area computer networks is becoming increasingly important 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. Not all receivers, organizations, or service providers posses or afford the same amount of bandwidth. Also, the user's access technology varies. Typically users connect through 128 kbps ISDN lines, via 500 kpbs cable modems, or 10-100 Mbits local area networks. These variations are 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, host computational resources, and finances allow. Conventional video compression techniques code the video signal to a fixed target bit rate, and is then multicasted to all receivers. Good quality video bit-rates of 4Mbps or more are within the high-bandwidth receivers' capability, but this outperforms the low capacity receivers by more than an order of magnitude. The target bit rate is therefore typically chosen quite low to enable as many receivers as possible to participate. However, this yields an unacceptable quality for the high capacity receivers.

We examine *layered coding* coupled with multicasting, where the video stream is coded and compressed into several distinct layers of significance as shown in Figure 1. All layers are transmitted on different multi-cast channels which receivers may join according to their capabilities and preferences. The more layers received, the higher video quality, but also higher bandwidth and processing requirements. The most significant layer constitutes the *base layer* and the following layers *enhancement layers.*

*Author for contact

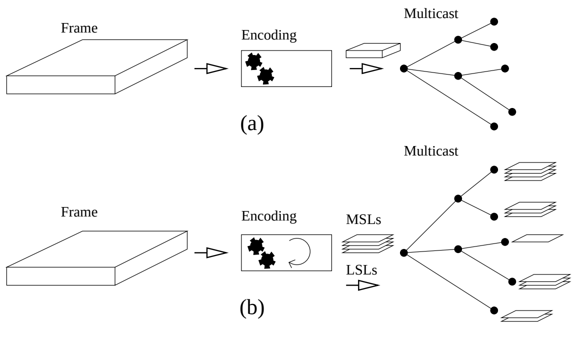


Figure 1 Conventional single bit-rate video coding (a) vs. layered coding (b).

Live video requires that video frames can be 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 requires significant flexibility. Fortunately, most modern CPU-architectures have been extended with Single Instruction Multiple Data (SIMD) instructions to speed up digital signal processing. Examples include VIS in Sun's UltraSPARC [17], MMX in Intel's Pentium(II) [12], 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 a reasonably sized test-video in real-time. We demonstrate how SIMD parallelism and careful considerations of super scalar processing can speedup image processing significantly. Our encoder implementation exists in both an optimized C-version and 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.

Our codec combines spatial and SNR layering, using a hybrid wavelet/DCT coding scheme. Spatial layering is performed using quadratic spline wavelets, and SNR layering is performed through repeated quantization. Furthermore, the lowest resolution wavelet layer is DCT coded before quantization for enhanced psycho-visual compression.

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 Codec Design

Three parameters influence the quality of a video stream and thus the bit-rate. The *frame rate* designates the number of frame updates per second. A higher frame rate results in smoother video display but also increases the amount of data (bits) needed to represent the video stream. The *resolution*, measured in (x $\times$ y) pixels, defines image resolution. Higher resolutions provide greater image detail and thus better quality but also require more bits. The last parameter is *bits-per-pixel*, which defines pixel precision. 8 bits correspond to $2^8 = 256$ levels of detail. The more levels, the higher image quality, and again more bits needed to represent the video stream.

Fixing these parameters essentially implies choosing visual video quality. Higher resolution means better image quality, thus increasing the *spatial* resolution of the image. Likewise, the bits-per-pixel parameter affects the pixel resolution, or *SNR* (signal-to-noise ratio) characteristics of an image, and frame rate decides the *temporal* resolution of the image. To each quality parameter we can associate a layering strategy:

Spatial layering splits the input into a number of layers of lower *resolution*. The base layer has the lowest resolution, and the resolution increases with additional layers.

SNR layering splits the incoming pixel values into several levels of significance. The result is a coarse version of the image (hence the name SNR) with few *bits-per-pixel* as the base layer, which is then progressively refined as more layers are added.

Temporal layering splits the video stream into layers of differing *frame rates*. The most significant layer has the lowest frame rate, which is then increased as the number of layers increases.

The primary goal of a layered codec is high *bandwidth diversity*, i.e., allow a wide range of receiver capabilities, both with respect to bandwidth and computational resources. Secondly, it should avoid data redundancy between layers in order to efficiently utilize available bandwidth, and thirdly, it should be computationally fast to permit real-time software solutions for both encoding and decoding. To decide whether these goals can be met, we design and implement a layered codec, and evaluate it through a series of performance measurements. To achieve these goals, our codec combines spatial and SNR layering.

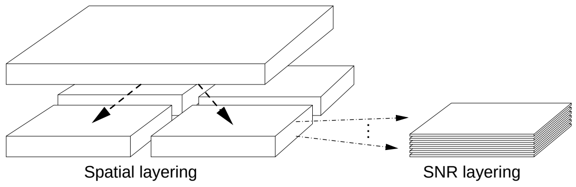


Figure 2 Spatial and SNR layering. Spatial layering splits the input frame by resolution, and SNR layering divides the spatial layers by information contents (bits-per-pixel).

Figure 2 shows the effects of spatial and SNR layering on an image. Spatial layering decomposes a frame into several smaller units of lower resolution, which is then divided into several "bit-planes" of decreasing information significance by SNR layering. Below we further describe spatial, SNR, and temporal layering, as well as our methods for their implementation using wavelets, DCT, and quantization. Finally, we develop a detailed codec model which describes the interaction and functionality of the chosen codec components.

2.1 Spatial Layering

Spatial layering decomposes the input into a number of layers of lower resolution. Assuming that decomposition is performed by reducing by a factor of 2 in each dimension, spatial layering immediately reduces required bit-rate at the base-layer to one quarter before further processing. This promising bit-rate reduction, as well as the option of transmitting and viewing the video stream at different spatial resolutions, inspired us to include spatial layering in the codec.

A simple spatial layering strategy would be to select only every nth scan line horizontally and vertically. While this certainly is efficient, it provides poor picture quality because details in the input image are easily lost, if they happen to lie outside the base layer scan lines. A filtering technique which uses information from a neighborhood of pixels in the input image is preferable because details are preserved better. Therefore we have chosen the wavelet transform for spatial layering, because it provides exactly this subsampling and filtering while still being computationally efficient ($O(n)$ [14, p. xiv]). It decomposes the input image into subbands of low and high frequencies and filters the output.

Typically, the transformation subsamples the image by a factor of 2 in each dimension, thus decomposing it into one low (L) and one high frequency (H) subband per dimension. Figure 3 shows the effect of subband decomposition of the image Lena.

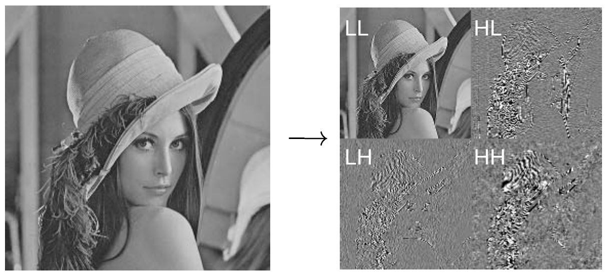


Figure 3 Wavelet transformation of the Lena image into LL, HL, LH, HH layers. The LL layer is a downsampled version of the original image and the HL, LH, and HH layers contain the information required to restore horizontal, vertical, and diagonal resolution, respectively.

The result is an LL subband which is a subsampled and filtered version of the original image, and three subbands, HL, LH, and HH, which 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.

There is no redundancy between subbands so they make perfect layers. Further, since the LL subband is essentially a subsampled version of the original image, it is possible to apply the familiar block-based DCT coding scheme known from other codecs on it for further compression, e.g., by using MPEG-II. The subband decomposition dictates an ordering between the wavelet layers; The LL layer is most significant since it contains the base image, and the HH layer is the least significant as typically it contains the least information.

2.1.1 The Wavelet Transform

The wavelet filter transforms a set of samples to the frequency domain. Unlike the Fourier transform, which is based on periodic, trigonometric functions, the wavelets are usually based on polynomials. The wavelets work on small, overlapping blocks, which in practice means that the blocking artifacts known from for example the discrete cosine transform are not present in images transformed using wavelets. It also means that the wavelets can be applied repeatedly for several levels of decomposition; This process is commonly referred to as *octave-band* decomposition. Both wavelet functions and wavelet transforms are fairly recent mathematical discoveries, but have quickly grown to become an entire mathematical discipline. Therefore it is out of the scope of this article to cover these in detail. For text book coverage, readers are encouraged to turn to [21], [14], or [7].

For the actual wavelet filter functions, we have chosen a 4-tap 1-3-3-1 spline wavelet [21, p. 136], because it has several advantages: It has *good pixel value approximation* because it is based on splines. Splines, which are piecewise polynomials with a smooth fit between the pieces, are excellent for interpolation. Also, it is invertible (biorthogonal) so it provides *perfect reconstruction*. Finally, it is *simple and efficient* because it is short and symmetric, meaning small pixel overlaps, hence requiring few operations per output pixel. As a side effect, it also simplifies border handling.

This wavelet tends to slightly emphasize high frequency data when constructing the LL layer, thereby producing a sharper downsampled image.

For analysis (subband decomposition), the 1-3-3-1 wavelet has the following form in the common z-transform notation:

$$\begin{aligned} H_L &= -1z^2 + 3z^1 + 3 - z^{-1} \\ H_H &= -1z^2 + 3z^1 - 3 + z^{-1} \end{aligned}$$

where H_L is the low frequency and H_H is the high frequency filter. It is a centered wavelet, meaning that it is centered around the z^0-term.

For synthesis (reconstruction), the wavelet has the following formulas:

$$\begin{aligned} G_L &= 1z^2 + 3z^1 + 3 + z^{-1} \\ G_H &= -1z^2 - 3z^1 + 3 + z^{-1} \end{aligned}$$

where G_L is low and G_H the high frequency part. Instead of using the z-transform notation, we represent the filters more compactly by using only their coefficients (represented by vectors, e.g. H_L = [-1 3 3 -1]) in the remainder of this paper.

2.1.2 The 2D Wavelet Filter

The above definitions apply to one dimension. The filter therefore needs to be applied twice to an image to produce the 4 desired subbands. Because this filter is (bi)orthogonal, there is no correlation between the horizontal and vertical dimension, so the horizontal and vertical filters can be applied in arbitrary order, or both dimensions simultaneously. The last form is potentially more efficient as it requires only one traversal of the input image instead of two.

This requires that the subband decomposition (and reconstruction) algorithm is capable of processing 2D data, as well as a set of 2D filters derived from the 1-3-3-1 wavelet above. By the convolution rule [14, p. 5], 2D filters, or 2-channel filters, are constructed as products of the H_L and H_H (or G_L and G_H) filters above. Let H_L^n denote the nth coefficient of H_L's coefficient vector [-1 3 3 -1], then the LL filter coefficient matrix, henceforth named H_{LL}, would be of the form

$$\begin{aligned} H_{LL}^{xy} &= H_L^x \cdot H_L^y = \\ &= \begin{bmatrix} H_L^0 \cdot H_L^0 & H_L^1 \cdot H_L^0 & H_L^2 \cdot H_L^0 & H_L^3 \cdot H_L^0 \\ H_L^0 \cdot H_L^1 & H_L^1 \cdot H_L^1 & H_L^2 \cdot H_L^1 & H_L^3 \cdot H_L^1 \\ H_L^0 \cdot H_L^2 & H_L^1 \cdot H_L^2 & H_L^2 \cdot H_L^2 & H_L^3 \cdot H_L^2 \\ H_L^0 \cdot H_L^3 & H_L^1 \cdot H_L^3 & H_L^2 \cdot H_L^3 & H_L^3 \cdot H_L^3 \end{bmatrix} \\ &= \begin{bmatrix} -1 \cdot -1 & 3 \cdot -1 & 3 \cdot -1 & -1 \cdot -1 \\ -1 \cdot 3 & 3 \cdot 3 & 3 \cdot 3 & -1 \cdot 3 \\ -1 \cdot 3 & 3 \cdot 3 & 3 \cdot 3 & -1 \cdot 3 \\ -1 \cdot -1 & 3 \cdot -1 & 3 \cdot -1 & -1 \cdot -1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & -3 & -3 & 1 \\ -3 & 9 & 9 & -3 \\ -3 & 9 & 9 & -3 \\ 1 & -3 & -3 & 1 \end{bmatrix} \end{aligned} \tag{1}$$

Using coefficient matrices, the resulting H_{LL} as well as the three enhancement layer filters, H_{HL}, H_{LH}, and H_{HH}, are stated below.

$$\begin{aligned} H_{LL} &= \begin{bmatrix} 1 & -3 & -3 & 1 \\ -3 & 9 & 9 & -3 \\ -3 & 9 & 9 & -3 \\ 1 & -3 & -3 & 1 \end{bmatrix} \quad & H_{HL} &= \begin{bmatrix} 1 & -3 & 3 & -1 \\ -3 & 9 & -9 & 3 \\ -3 & 9 & -9 & 3 \\ 1 & -3 & 3 & -1 \end{bmatrix} \\ H_{LH} &= \begin{bmatrix} 1 & -3 & -3 & 1 \\ -3 & 9 & 9 & -3 \\ 3 & -9 & -9 & 3 \\ -1 & 3 & 3 & -1 \end{bmatrix} \quad & H_{HH} &= \begin{bmatrix} 1 & -3 & 3 & -1 \\ -3 & 9 & -9 & 3 \\ 3 & -9 & 9 & -3 \\ -1 & 3 & -3 & 1 \end{bmatrix} \end{aligned} \tag{2}$$

The reconstruction matrices G_{LL}, G_{HL}, G_{LH}, and G_{HH}, are constructed similarly:

$$G_{LL} = \begin{bmatrix} 1 & 3 & 3 & 1 \\ 3 & 9 & 9 & 3 \\ 3 & 9 & 9 & 3 \\ 1 & 3 & 3 & 1 \end{bmatrix} \quad G_{HL} = \begin{bmatrix} 1 & 3 & -3 & -1 \\ 3 & 9 & -9 & -3 \\ 3 & 9 & -9 & -3 \\ 1 & 3 & -3 & -1 \end{bmatrix}$$
$$G_{LH} = \begin{bmatrix} 1 & 3 & 3 & 1 \\ 3 & 9 & 9 & 3 \\ -3 & -9 & -9 & -3 \\ -1 & -3 & -3 & -1 \end{bmatrix} \quad G_{HH} = \begin{bmatrix} 1 & 3 & -3 & -1 \\ 3 & 9 & -9 & -3 \\ -3 & -9 & 9 & 3 \\ -1 & -3 & 3 & 1 \end{bmatrix} \tag{3}$$

It is clear from these coefficient matrices that the filters are extremely symmetric. This calls for several optimizations, but we postpone the details until the implementation section.

There is one problem with this filter: It always needs data 2 pixels back from the current position, but these points do not exist at the image borders. The best method for solving this border problem is to mirror input data points [14, p. 263]. Suppose $x[n]$ is the input data, and you are filtering the first data points. Then, using the filters above, you would need $x[-2]$, $x[-1]$, $x[0]$, and $x[1]$ for filtering the first pixels in each scan line, where $x[-2]$ and $x[-1]$ are undefined. Mirroring input points solves this problem by defining $x[-(n+1)] = x[n]$. Then $x[-2] = x[1]$ and $x[-1] = x[0]$ and the filter can behave as normal.

2.2 SNR Layering

In SNR layering the goal is to divide the incoming coefficients into several levels of significance, such that 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. The layering explicitly defines the ordering between SNR layers after significance; decoding only low significance layers makes no sense without the most significant layers. Therefore, SNR layers are always decoded in order of significance.

The task of splitting input coefficients into SNR layers is performed by a quantizer. A quantizer reduces the number of symbols in a data stream by dividing the dynamic range of the coefficients into a number of decision levels, and outputting the decision level in which each input coefficient belongs. For example, with a step size of 8, a byte is divided into $256/8 = 32$ decision levels: Input coefficients values from 0 to 7 belong to level 0, values from 8 to 15 to level 1 and so forth. A quantizer as the above, with a constant step size across the dynamic range it covers, is called a *uniform* quantizer. A uniform quantizer is easy to implement: It corresponds to integer division followed by a rounding step. Therefore we use uniform quantizers in the codec.

2.2.1 Enhancement Layer Quantization

For the coefficients in the spatial enhancement layers, the HL, LH, and HH layers, we use a uniform scalar quantizer. The LL layer is treated differently as described below. The scalar refers to a single number, and hence that all coefficients are processed independently with the same decision levels. This is opposed to the matrix quantizer presented below which processes a block of coefficients at a time, each with potentially different coefficients.

The scalar quantizer traverses the layer from top-left to bottom-right by scan line. But a single traversal is insufficient. Input coefficients must be quantized repeatedly, in such a way that the remainder from one layer of quantization is used in the next layer for refinement. Naturally, it is not efficient to traverse data repeatedly. Rather, we read input data once, thus needing only one traversal of input data, and quantize repeatedly, outputting several layers one coefficient at a time. In fact, it is possible to quantize several coefficients at once using SIMD, but these optimization issues are again postponed until the implementation section.

2.2.2 Spatial Base Layer Quantization

The LL spatial layer is a downsampled version of the original, so we can apply the well-known Discrete Cosine Transform (DCT) on the image. The DCT has proven very effective in reducing spatial redundancy in images thus adding to the compression potential. The DCT operates on a block of pixels by transforming them from the spatial to frequency domain. It allows ordering information contents in image blocks by psycho-visual significance, which in turn allows layering by coefficient selection.

The upper left corner of the DCT matrix holds the lower frequency coefficients with higher frequencies distributed along the right and down directions. The frequency content of natural images is primarily of low frequency, thus the DCT compacts the image information towards the upper left corner. An example is given in Figure 4 which shows a block of pixel values on the left, and the same block after DCT in the middle. The compaction of data in the upper left

$$\begin{bmatrix} 175 & 173 & 177 & 180 & 171 & 168 & 165 & 147 \\ 182 & 180 & 177 & 174 & 174 & 173 & 160 & 148 \\ 181 & 179 & 181 & 182 & 177 & 169 & 155 & 159 \\ 187 & 185 & 189 & 191 & 184 & 176 & 160 & 163 \\ 190 & 192 & 200 & 201 & 182 & 165 & 164 & 160 \\ 200 & 210 & 211 & 197 & 173 & 159 & 160 & 151 \\ 201 & 198 & 185 & 172 & 153 & 146 & 142 & 133 \\ 178 & 163 & 146 & 139 & 132 & 116 & 98 & 107 \end{bmatrix} \longrightarrow \begin{bmatrix} 1358 & 119 & -23 & -4 & 3 & 4 & 1 & -3 \\ 45 & -54 & -18 & 9 & -4 & 3 & 2 & 1 \\ -84 & 14 & 14 & 16 & -2 & -1 & -1 & 0 \\ 55 & 5 & -6 & -13 & -12 & 12 & -8 & 3 \\ -27 & -9 & -5 & 2 & 10 & 1 & 1 & 0 \\ 15 & -1 & 3 & 0 & -3 & 2 & 0 & -1 \\ -7 & 1 & -3 & -7 & 3 & -1 & 0 & 1 \\ -2 & -3 & 0 & -2 & 3 & 3 & -2 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 169 & 7 & -1 & 0 & 0 & 0 & 0 & 0 \\ 2 & -3 & 0 & 0 & 0 & 0 & 0 & 0 \\ -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4 Discrete Cosine Transformation of an 8x8 pixel block, followed by quantization with the MPEG quantization matrix, equation (4).

corner is clearly visible in this example.

The right block in Figure 4 is the DCT block after quantization with the MPEG quantization matrix in equation (4), which is employed by the uniform matrix quantizer used in our codec. The uniform matrix quantizer takes a block of coefficients as input and quantizes them by dividing each element in the block by the corresponding scalar in the quantization matrix.

The advantage of using a matrix quantizer is that when it is applied to a block of DCT coefficients, it can be adjusted so that it minimizes error according to the human visual system. It does so by quantizing high frequency coefficients with a larger step size because, to the human eye, an error in a high frequency coefficient is less visible than an error of the same magnitude in a low frequency coefficient. This weighing in frequency is apparent in the quantization matrix used in MPEG, which is depicted in equation (4). The lowest frequency in the upper left corner has a small step size, and as the frequency grows along the down and right axes, so does the quantization step.

$$T(x,y) = \begin{bmatrix} 8 & 16 & 19 & 22 & 26 & 27 & 29 & 34 \\ 16 & 16 & 22 & 24 & 27 & 29 & 34 & 37 \\ 19 & 22 & 26 & 27 & 29 & 34 & 34 & 38 \\ 22 & 22 & 26 & 27 & 29 & 34 & 37 & 40 \\ 22 & 26 & 27 & 29 & 32 & 35 & 40 & 48 \\ 26 & 27 & 29 & 32 & 35 & 40 & 48 & 58 \\ 26 & 27 & 29 & 34 & 38 & 46 & 56 & 69 \\ 27 & 29 & 35 & 38 & 46 & 56 & 69 & 83 \end{bmatrix} \tag{4}$$

Using DCT prior to quantization allows two different types of psycho-visually enhanced SNR layering types: The matrix quantization described above, which works on all coefficients, and simple coefficient selection, which we call *band-pass* layering [6, pp. 32-34]. Band-pass layering implies selecting a range of adjacent coefficients which together constitute a layer. A few, significant coefficients may constitute a base layer, while larger groups of less significant coefficients may constitute enhancement layers. These two layering types may be combined to *hybrid* layering [6, p. 34] which provides more fine-grain control of quality and bit-rate. Details of these layering types and their implications on bandwidth reduction and image quality are also found in [6].

As the DCT coded layers belong to the LL wavelet layer, the ordering between spatial and SNR layers is implicitly defined. The DCT coded SNR layers must be decoded as the first before the spatial (wavelet) SNR enhancement layers.

2.3 Temporal Layering

The purpose of temporal layering is to enable receivers to view the same video stream at different frame rates such that frame updates increases with the receiver's bandwidth and processing capability. We here outline some possible design options.

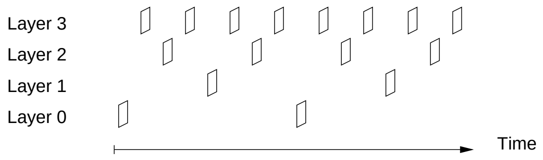


Figure 5 A temporal layering example: Selecting frames from a video stream for transmission. Layer 0 has the lowest frame rate and, for each added layer, the frame rate doubles.

Frame dropping provides layering by selectively transmitting frames: Fewer frames need less bandwidth. The temporal base layer has a low frame rate, and frame rate is increased by adding temporal layers, as shown in Figure 5. Each frame is treated as a separate unit.

Conditional block replenishment transmits only blocks with changes larger than certain thresholds from one frame to the next. Layering results by set-

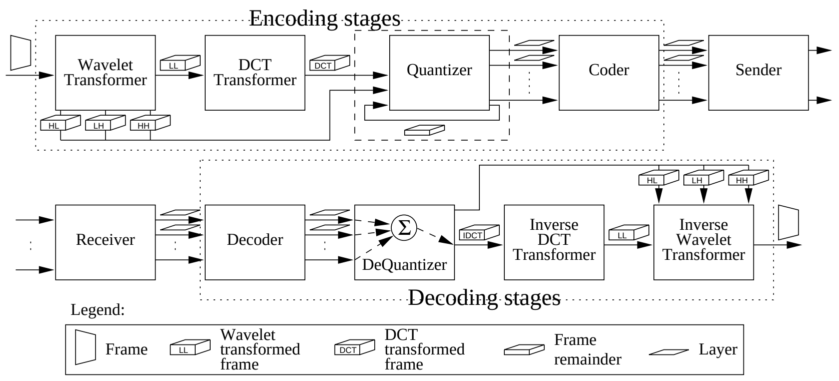


Figure 6 Our codec model. In the encoding stages, the wavelet transformer sends the LL layer to the DCT stage before all layers go through the quantizer and on to transmission. The quantizer reuses coefficient remainders, depicted by the arrow reentering the quantizer carrying the frame remainder. The decoding stages perform the inverse functions.

ting several thresholds, resulting in higher change sensitivity at higher layers. Transmitting fewer blocks reduces bandwidth requirements, but may produce image artifacts in case a block's changes remain below the specified threshold while the surrounding ones are changed.

Predictive coding does not in it self provide temporal layering. It does however reduce bandwidth requirements considerably by utilizing and removing redundancies between frames: Only blocks that change are coded, and the codec tries to predict whether the change is due to motion, and if so only transmit motion vectors instead of updating a whole block as above. This has a side effect on temporal layering, though: Separating frames for layered transmission becomes more difficult due to the possible dependencies introduced between the predictively coded frames. Temporal layering using predictive coding would need to consider this problem by minimizing this dependency between updates, or minimize the motion error introduced if predicted frames are dropped.

Our design does not offer temporal layering besides simply dropping frames. The most effective temporal compression schemes use predictive coding to compensate for motion, 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. Temporal layering and predictive coding are both active research areas, and interested readers are referred to [24], [18], [5], and [8]. Due to the amount research needed to produce an efficient temporal layering using predictive coding we postpone this to future research. Our codec does thus not include temporal layering besides the possibility of selectively sending frames.

2.4 The Codec Model

Our overall codec model is shown in Figure 6. The upper half of the figure shows the encoding stages, and the lower half the decoding stages of the codec. The sender and receiver stages are modules outside the actual codec which handle connection setup and data transmission. Also, before the encoding stages there is a colour space conversion from RGB to YUV (luminance/chrominance format) which enhances the psycho-visual compression potential of using DCT coding. Likewise, there is a reconversion after the decoding stages, but these stages are of less interest to the design.

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 Discrete Cosine Transform on the input blocks in the LL image from above. The resulting coefficient blocks are passed on to the quantization stage. The *Quantizer* performs SNR layering by splitting incoming coefficients into several layers of significance. The *Coder* takes the quantized coefficients from each separate layer and Huffman compresses them to produce a single bit-stream for each layer.

The decoding stages perform the inverse of the above using the following components: The *Decoder* uncompresses the Huffman codes in the received layers. The *Dequantizer* reconstructs coefficients based on the decoded layers; the Dequantizer works both as coefficient reconstructor and layer merger. As in 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 and thus reconstructs the lowest resolution spatial layer, LL. The LL layer is used in the *Inverse wavelet transformer*, which performs wavelet reconstruction using the reconstructed LL layers from the IDCT stage and reconstructed enhancement layer coefficients, if any.

The Sender and Receiver modules 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.

3 Implementation

The purpose of the implementation is to allow functional testing and performance measurements. Our focus is onan efficient implementation, and since the elements of digital signal processing in our codec design are well suited for SIMD processing, we have chosen to implement a prototype using VIS (visual instruction set) [17] on SUN UltraSPARC CPUs.

3.1 The Wavelet Transform

The presentation of our implementation of the 1-3-3-1 wavelet transform has two parts: First, we explore the optimizations possible by utilizing the symmetries in the filter matrices. Second, we describe the actual implementation of the filtering and reconstruction functions using VIS.

3.1.1 Utilizing the Filter Symmetries

The filters are highly symmetric. Therefore, the algorithms for both decomposition and reconstruction can be optimized accordingly so as to avoid redundant processing. The important observation is that the matrices are identical except for their signs. Extracting only the signs for clarity, results in the sign-matrices in (5) for decomposition and (6) for reconstruction.

$$\begin{aligned} H_{LL} &= \begin{bmatrix} + & - & - & + \\ - & + & + & - \\ - & + & + & - \\ + & - & - & + \end{bmatrix} & H_{LH} &= \begin{bmatrix} + & - & - & + \\ - & + & + & - \\ + & - & - & + \\ - & + & + & - \end{bmatrix} \\ H_{HL} &= \begin{bmatrix} + & - & + & - \\ - & + & - & + \\ - & + & - & + \\ + & - & + & - \end{bmatrix} & H_{HH} &= \begin{bmatrix} + & - & + & - \\ - & + & - & + \\ + & - & + & - \\ - & + & - & + \end{bmatrix} \end{aligned} \tag{5}$$

$$\begin{aligned} G_{LL} &= \begin{bmatrix} + & + & + & + \\ + & + & + & + \\ + & + & + & + \\ + & + & + & + \end{bmatrix} & G_{LH} &= \begin{bmatrix} + & + & + & + \\ + & + & + & + \\ - & - & - & - \\ - & - & - & - \end{bmatrix} \\ G_{HL} &= \begin{bmatrix} + & + & - & - \\ + & + & - & - \\ + & + & - & - \\ + & + & - & - \end{bmatrix} & G_{HH} &= \begin{bmatrix} + & + & - & - \\ + & + & - & - \\ - & - & + & + \\ - & - & + & + \end{bmatrix} \end{aligned} \tag{6}$$

The vertical high-resolution matrices H_{LH} and H_{HH} (or G_{LH} and G_{HH}) are exact copies of their low vertical resolution counterparts, only with the two lower rows negated. The same applies horizontally, where the high-resolution matrices are copies of the low-resolution ones with the two rightmost columns negated. As we use SIMD, which operates across data columns, it is most convenient to utilize the *vertical* symmetries.

These symmetries enable the processing required for the decomposition matrices to be halved for each block by only filtering with the vertical low-resolution matrices and computing the high-resolution matrices here-from. This involves filtering with H_{LL} and H_{HL}, and copying the upper two rows of the unchanged result and the lower two rows of the result negated, to H_{LH} and H_{HH}. In practice, however, this task is performed without copying matrices. Instead, after computing the LL and HL result matrices, the two upper rows of each matrix are summed to the intermediate results, r_L^{upper} and r_H^{upper} for LL and HL, respectively. Likewise, the lower two rows are summed into r_L^{lower} and r_H^{lower}. Then, all four results matrices, named R_{LL}, R_{HL}, R_{LH}, and R_{HH}, are constructed as either a sum or a difference between two of these intermediate results, as shown in (7).

$$\begin{aligned} R_{LL} &= r_L^{upper} + r_L^{lower} & R_{HL} &= r_L^{upper} + r_L^{lower} \\ R_{LH} &= r_L^{upper} - r_L^{lower} & R_{HH} &= r_L^{upper} - r_L^{lower} \end{aligned} \tag{7}$$

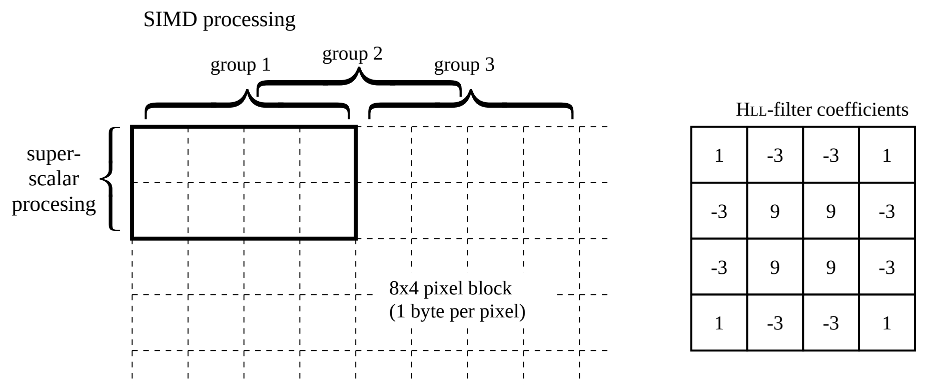


Figure 7 Utilizing SIMD and super scalar processing. SIMD instructions allow 4 elements in the same row to be multiplied with the corresponding filter coefficients row in parallel. Super scalar processing enables two rows in the same group to be computed in parallel. Thus all 8 pixels in the enclosed area are processed in parallel.

A similar processing reduction is done for reconstruction. Here each input coefficient is multiplied with the reconstruction matrix for the layer from which is originates, resulting in a 4×4 matrix result for each layer. As shown in (3) the 2 upper and 2 lower rows are equal, mirrored around the center, except for sign for vertical high-frequency as seen in (6). As all matrix elements are multiplied by the same source, only the upper 2 rows of each matrix need to be computed; the remaining are constructed by mirrored, possibly sign inverted, versions hereof. This effectively reduces computations by 50%.

This procedure is only possible because the low and high-frequency filters in the 1-3-3-1 wavelet differ only in their signs. Other filters, such as the 1-2-6-2-1/1-2-1 spline filter [21, p. 136] do not have this property.

3.1.2 Visual Instruction Set Implementation

Both decomposition and reconstruction functions are written entirely using VIS. The visual instruction set found on SUN's UltraSPARC CPU's is capable of processing 8×8 bit, 4×16 bit, or 2×32 bit partitioned data elements in parallel. In addition to the usual multiplication and addition instructions VIS contains various special instructions dedicated to video compression and manipulation of 2- and 3-dimensional data. Furthermore, it has two pipelines and is therefore able to execute pairs of VIS instructions in parallel [9].

The processor dependent optimizations fall in three categories: 1) using the Visual Instruction Set 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 optimizations principles are applied in all stages of the codec. Below we exemplify these on the layer decomposition stage.

The principle behind applying the decomposition filter is to multiply 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 7, 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

```
// value 1
m1    = vis_fpadd16(vis_fmul8x16(vis_read_hi(src0), matrix1L),  //  1: Calc r^upper_L
                    vis_fmul8x16(vis_read_hi(src1), matrix2L));
m2    = vis_fpadd16(vis_fmul8x16(vis_read_hi(src2), matrix2L),  //  2: Calc r^lower_L
                    vis_fmul8x16(vis_read_hi(src3), matrix1L));
m3    = vis_fpadd16(vis_fmul8x16(vis_read_hi(src0), matrix1H),  //  3: Calc r^upper_H
                    vis_fmul8x16(vis_read_hi(src1), matrix2H));
m4    = vis_fpadd16(vis_fmul8x16(vis_read_hi(src2), matrix2H),  //  4: Calc r^lower_H
                    vis_fmul8x16(vis_read_hi(src3), matrix1H));
a1    = vis_fpadd16(m1, m2);                                    //  5: a1 = R_LL
a2    = vis_fpsub16(m1, m2);                                    //  6: a2 = R_LH
a3    = vis_fpadd16(m3, m4);                                    //  7: a3 = R_HL
a4    = vis_fpsub16(m3, m4);                                    //  8: a4 = R_HH
ll1   = vis_fpadd16s(vis_read_hi(a1), vis_read_lo(a1));         //  9: Sum 2x2 LL columns
hl1   = vis_fpadd16s(signoffs,                                  // 10: Add sign offset
        vis_fpadd16s(vis_read_hi(a3), vis_read_lo(a3)));        //     to sum of 2x2 HL columns
dest0 = vis_fpack32(dest0,                                      // 11: Pack LL, HL result bytes
        vis_fpadd16(vis_freg_pair(ll1, hl1),                    //     after merging to 64-bit and
        vis_faligndata(vis_freg_pair(ll1, hl1), zero)));        //     adding result shifted 16-bits
lh1   = vis_fpadd16s(signoffs,                                  // 12: Add sign offset
        vis_fpadd16s(vis_read_hi(a2), vis_read_lo(a2)));        //     to sum of 2x2 LH columns
hh1   = vis_fpadd16s(signoffs,                                  // 13: Add sign offset
        vis_fpadd16s(vis_read_hi(a4), vis_read_lo(a4)));        //     to sum of 2x2 HH columns
dest1 = vis_fpack32(dest1,                                      // 14: Pack LH, HH result bytes
        vis_fpadd16(vis_freg_pair(lh1, hh1),                    //     after merging to 64-bit and
        vis_faligndata(vis_freg_pair(lh1, hh1), zero)));        //     adding result shifted 16-bits
```

Figure 8 Wavelet filter kernel implementation using VIS. This section is a simplified version of the actual kernel, which repeats this filtering section four times with different sources and includes loads/stores, as well as spilling and border handling.

multiplies four 16-bit values with four 8 bit values producing four 16 bit results, the four pixels in row 1 in group 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 computations by three quarters, and super scalar parallelism further reduces this number by half.

The VIS instructions are available to the application programmer through a set of macros which can be used from C-programs with reasonable effort, although it must be realized that even with these C-macros, programming VIS is essentially at the assembly level. The programmer is, however, alleviated from certain aspects of register allocation and instruction scheduling. To illustrate the use of VIS instructions in the implementation, part of the wavelet filtering kernel is depicted in Figure 8.

Following the design, lines 1-4 compute the temporary results r_L^{upper}, r_L^{lower}, r_L^{upper}, and r_L^{lower} from (7). Lines 5-8 compute LL, HL, LH, and HH as sums and differences of these Rs results by adding 64-bit values. In lines 9-10 and 12-13 the 64-bit results are summed across columns into 32-bit results by adding the high 32-bit with the lower 32-bit in each register. Lines 10, 12, and 13 also add a sign offset because VIS uses unsigned 8-bit values. All negative values would be clamped to zero in the enhancement layers if this offset was not added. This is compensated for in the reconstruction algorithm. Finally, in lines 11 and 14, the final pixel values are constructed. First, the results still span two 16-bit columns in each of the 32-bit `ll1`, `hl1`, `lh1`, and `hh1` registers. Therefore the registers are shifted 16 bits left by the `vis_faligndata` instruction and added to the non-shifted registers. To optimize, the registers are first gathered in pairs into 64-bit registers, and then shifted and added. The final result resides in the upper part of each 32-bit half in the 64-bit result registers, and the `vis_fpack32` packs the result into bytes by clamping each of the 32-bit values in the 64-bit source to 2 unsigned 8-bit values.

The implementation of the reconstruction filters follows the same optimizations principles as the decomposition filters. A significant difference is that there are 4 different reconstruction functions, one for each layer. The image data is thus traversed each time a layer is added. This is not the fastest option but it is the most flexible: It allows the receivers to select and decode any layer. We impose just one restriction: As the enhancement layers make little sense by themselves, we require that the LL layer be decoded as the first.

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.

3.2 DCT/IDCT Coding the LL Layer

We use the DCT-functions present in SUN's mediaLib [15] for DCT coding the lowest resolution spatial layer. mediaLib is a publicly available library, and exist both in C and VIS-accelerated versions. The DCT function in mediaLib takes an 8×8 block of 8-bit coefficients as input and outputs an 8×8 block of 11-bit DCT coefficients. The IDCT function does the inverse.

The output coefficients from the DCT function are not ordered by frequency which is preferable for psycho-visually enhanced compression. The extra compression potential comes from utilizing the fact that in natural images, the DCT compacts information towards the lower frequencies and many high-frequency coefficients are likely to be zero after quantization. Therefore, if the coefficients are ordered by frequency, one can stop coding each block earlier, meaning shorter blocks in the compressed bit-stream and subsequently better compression.

The resulting DCT coefficients are therefore reorganized using a technique referred to as *zigzagging*. In zigzagging, the DCT coefficients are reordered by selecting them according to a predefined pattern. The zigzagging pattern used in our codec is depicted in Figure 9. It is derived from the matrix used in MPEG, changed to obtain better visual quality with just 4 coefficients.

We have also taken advantage of the VIS 64-bit load/store capabilities for the zigzagging routine in our prototype. We do so by loading 4 16-bit values and packing them into one 64-bit register and storing. This saves 16.7% of the memory accesses required by a non-VIS implementation using 32-bit stores.

3.3 Quantization

Both the scalar quantization for wavelet coefficients and the uniform matrix quantization used for DCT coefficients are equivalent to integer division. But since division is an expensive operation measured in processor cycles, we prefer to compute the reciprocal of the quantization values at compile time and use multiplication for quantization instead. This comes with a cost however: rounding errors are bound to occur unless all quantization values are powers of two. These rounding error are small though (one half LSB), and in our implementation the errors do not propagate through iteration.

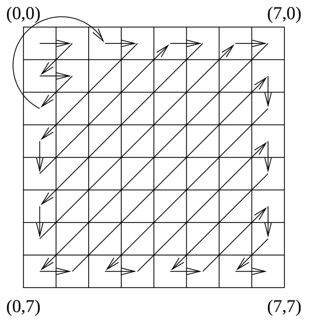


Figure 9 The zigzag pattern used in our codec. Compared to the MPEG-matrix, it improves approximation to Euclidean distance and hence visual quality when using the first four coefficients.

Another advantage is that it is possible to use VIS for quantization. Since the output from the DCT function is 16-bit integers (with 11 bits actually used), it is possible to quantize four coefficients simultaneously, reducing required multiplication operations to 1/4. Also, the number of memory accesses is halved due to 64-bit loads/stores.

Implementation details of the matrix quantization kernel can be found in [6, pp. 41-43].

3.4 Huffman Coding

A table based Huffman compressor encodes/decodes the quantized coefficients. A table based algorithm is considerably faster than sequentially processing each Huffman code using a state machine-like algorithm. But it also means that the Huffman codes cannot be fully expanded because the required decision tables would grow too large. For each bit added, the table would grow to twice the size. We therefore limit the maximum Huffman code length to 10 bits (corresponding to a table length of 1,024 bytes). While this representation is usually not optimal in terms of bandwidth, we have found it to be good in most cases.

In order to counteract the negative impact on bandwidth utilization by using static Huffman trees, the codec uses 4 different Huffman trees, optimized for different coefficient contents (different bit-per-pixel resolutions). For each layer, we can then choose Huffman tree most optimal for the information contents in that layer. For details of the implementation, see [6, pp. 43-45]

3.5 Current Status

The results of our implementation efforts are two applications: An encoder application and a decoder application. The encoder application takes raw video frames as input, encodes them, and transmits them to a specified set of receivers. The decoder application receives data, decodes it, and displays it in real-time on a connected X-windows compatible display device. For communication, the applications have modules that support ATM multicasting.

Both applications implement the layered codec with support for spatial and SNR layering for DCT layers. The applications have simple command line interfaces that provide control of vital behavior and settings, such as the number of layers used.

The final implementation takes up approximately 26,000 lines of code for the prototype in the form of C++ source code.

4 Performance Measurements

This section presents our performance measurements of bit-rates and CPU-usage. These measurements are essential for determining the usefulness of our layering concepts and chosen design. We required that the resulting bit-rates must be diverse and low for the lowest quality layers, while maintaining acceptable visual quality, and very good for high bandwidth streams. Also the codec must have low computational overhead, or it will be useless for real-time applications without hardware support.

4.1 Test Platform

The input for the measurements was a test video stream with 174 frames of 384×320 pixels at 24-bit colour, digitized at 18fps. The video stream is a typical "talking head" sequence, with one person talking and two persons entering and leaving the image as background movement.

CPU-usage was measured on a Sun Ultra-1 workstation 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.

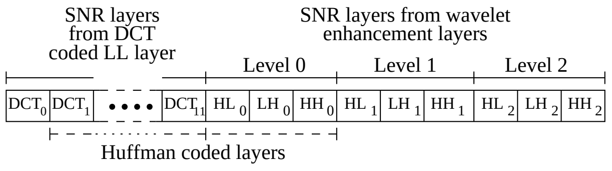
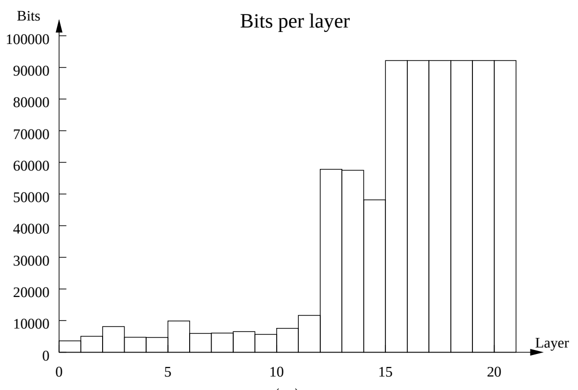
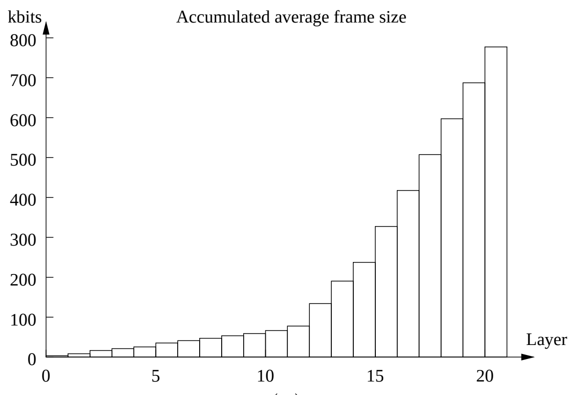


Figure 10 The layer definitions used for testing along with their significance ordering. DCT_0-DCT_{11} are layers constructed from the DCT coded LL layer. HL_0, LH_0, and HH_0 contain the most significant bits from the corresponding wavelet enhancement layers. Levels 1 and 2 are refinement layers to these.

Unless stated otherwise, our test codec uses one level of subband decomposition, and has a total of 21 layers as seen in Figure 10. The layers are depicted in their significance order; 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 hybrid 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 in Figure 10, consists 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



(a)



(b)

Figure 11 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 frame sizes are evenly distributed across the scale of bit-rates. The rate of change shifts at layer 12 where the wavelet enhancement layers are added.

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 distributions are very random and Huffman coding would add very little, if any, compression.

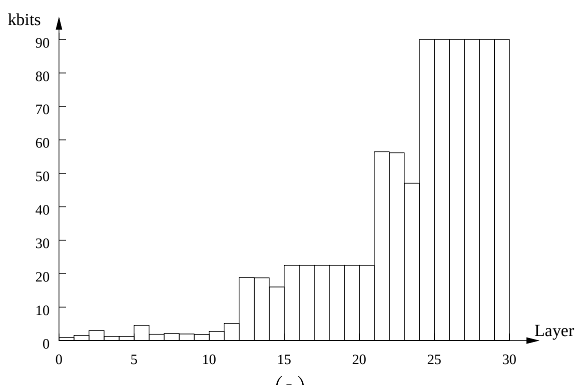
This layer distribution does not provide 100% reconstruction: The DCT coded layers reconstruct up to MPEG quality video (+1 bit for the DC coefficient) using all 12 DCT layers. The wavelet layer coefficients are rounded from 12-bit to 8-bit quantities to fit bytes, which is a necessity before DCT coding the LL layer, which needs bytes as input. Therefore, perfect reconstruction is also not possible from wavelet layers, but the error is small; Less than ± 1.5 in reconstructed pixel value using all wavelet enhancement layers from one level of decomposition. Note that the lack of perfect reconstruction is a prototype limitation, not a principal problem; our design allows perfect reconstruction except for possible rounding errors.

A number of sample images can be found on the World Wide Web: `http://www.cs.auc.dk/~bnielsen/codec/`

4.2 Bit-rate measurements

Figure 11(a) shows the average frame sizes per layer produced by our codec. The first 12 layers, which comprise the DCT coded LL layer, are the smallest, ranging from 3.5 to 11.6kbits. Then follow the three Huffman coded wavelet enhancement layers with sizes from 47kbits to 56.4kbits, and finally the 6 uncoded enhancement layers with fixed size 90kbits. Figure 11(b) shows the resulting accumulated average frame size obtained from gradually adding all 21 layers. The bit-rate scales nearly linearly with the number of layers, with a rate change at layer 12. Thus there are no sudden big gaps in the receiver capacities which can be supported. The lowest average frame size is 3.5kbits for one layer and the highest 777.3kbits with all 21 layers, yielding a difference factor of 222:1. We found the lowest useful layer count to be 3, which corresponds to an average frame size of 16.4kbits. This reduces the factor to 47.4:1, which is still, however, a large span. In consequence a receiver on a 128kbps ISDN line can receive 3 layers at 7.8fps which is just enough to be useful. A receiver with plenty of bandwidth can receive the entire video stream with 21 layers at 18fps, corresponding to 13.7Mbps. From the even distribution and the large span of accumulated bit-rates, we conclude that our codec provides good bandwidth versatility.

To determine the effect on the layer bandwidth distribution with more than on downsampling, we have performed the same layer size measurements on our codec configured to downsample the test stream twice. As Figure 12 shows, the the layers now falls in three sizes; the smallest layers (1-12) are the DCT-coded LLL layers. The next largest (13-21) is the enhancements layers of the second downsampling, and the largest (21-31) layers are the enhancement layers of the first downsampling. Again this produces a smooth, but accelerating, accumulated layer size distribution.



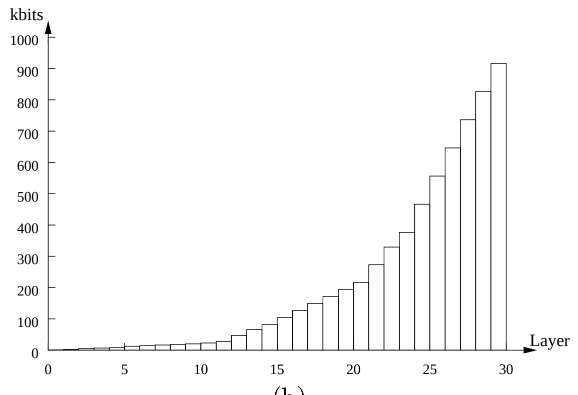


Figure 12 Bandwidth distribution on layers for a two-times downsampling codec. (a) Average layer size. (b) Accumulated layer size.

If the user is content with a very small version of the image (96×80 pixels) reconstructed from the 3 least significant DCT the bandwidth requirement falls to 5kbits per frame, corresponding to 25 fps on a 128 kbps ISDN connection. This experiment shows that repeated downsampling is an effective strategy for handling very large input images (e.g., high definition TV), or providing images to users on low-speed links.

Comparison with a non-downsampling codec

The bit-rates obtained with the spatially layering codec above are compared to an identical non-spatially layered one with the goal of checking the degree of orthogonality between spatial and SNR layering. Ideally, the bit-rate reduction for the DCT coded layers should be 4:1 per level of decomposition, but due to the information compaction resulting from image downsampling, the information contents per pixel block rises, and the relative compression rate declines. How much this declination amounts to the test video stream is the object of this section.

The non-spatially layered codec used for comparison has identical layer definitions apart from the absent wavelet layers, and therefore has 12 DCT coded hybrid layers. The difference between average frame sizes and their ratio in the two codecs is depicted in Figure 13.

The difference in compression ratio lies approximately between 2:1 and 4:1, except for layers 5 and 12 where it is approximately 1.2:1. Generally the ratio between the two codecs' compression rates decreases with higher layer numbers. This ratio difference stems from the spatial information compaction when downsampling, causing the image textures to vary more rapidly. This shows up as more high frequency data in the DCT coded blocks, which results in larger coefficients and longer blocks after quantization. The layers that carry high-frequency contents, layers 5 and 12 in particular, grow in size. These carry the highest frequency 34 coefficients in each block with layer 5 having the most significant information.

The difference in compression ratio decreases with the level of downsampling, due to increasing information compaction and the resulting rise of high-frequency information in the DCT blocks. Therefore, the smaller the image, the less gain in compression rate from downsampling. But as low bandwidth receivers are interested in the low frequency contents, where the difference ratio is still large, repeated downsampling is still meaningful. Very low bit-rate frames are also useful in video conferences with many participants. Here, the focus is typically on one or two persons at a time, and the remaining persons could be viewed at a very low resolution. This not only saves bandwidth but also visible screen area.

4.3 CPU usage measurements

The CPU-usage measurements show average encoding times (for both the C- and VIS-version), average decoding time (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

Table 1 Average encoding and decoding CPU-usage measurements. The Wavelet Transform stage produces all 4 wavelet layers, the DCT transforms the LL layer, and the Quant & Huff (DCT/wavelet) stages quantize and code the 9 output layers from the wavelet enhancement layers, and the 12 SNR layers from DCT, respectively. The DeQuant & UnHuff (DCT) stage decodes all 12 SNR layers for the IDCT stage. The 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. (†) The C and VIS versions of DCT and IDCT functions are from SUN's MediaLib graphics library[15]. (‡) Currently, only the C version exists of the (de)quantization and huffman (de)coding stages.

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

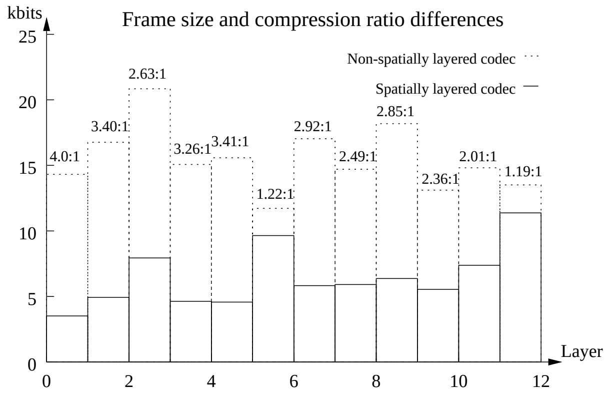


Figure 13 Difference in frame size between spatially and non-spatially layered codec.

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. Decoding is more expensive than encoding, because decoder is designed to permit separate decoding of each layer. This increased flexibility costs a separate iteration across the image per layer. Also two of the stages are not VIS-accelerated.

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 types of 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 [15] 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 coding is preferable to other methods such as the *lowest common denominator*-method, which transmits only one signal, tuned to the lowest bit-rate of any receiver. Of course, this is not acceptable for receivers with plenty of bandwidth to spare. Another method is *multi-rate coding* which repeatedly codes the video signal to several bit-rates. This is expensive, however. 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. Recently, another method, *router filtering* [23], does away with the bandwidth redundancy by coding to only one high bit-rate, and letting network routers perform the necessary bandwidth reductions underway. But this requires computationally powerful routers with semantic knowledge of the packets they process. Routers able to act on the semantic contents of packages are not likely to become common until the wide-spreading of active networks [16], [2].

Layered video coding and related network design issues are active and recent research areas. Related work on layered video codecs exist [4], [8], but we have focussed on practical and efficient implementation, with comprehensive and empirical tests to evaluate our codec's performance.

MPEG-2 was the first standard to incorporate a form of layering, called *scalability*, as part of the standard [3]. 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 spatial, SNR, and temporal scalabilities. While it provides better bit-rate adaption than the MPEG-1 standard, it is still inadequate for most networks with receivers on dial-up lines. 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 [22]. 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.

The work in [4] includes spatial and SNR layering, with strong focus on the spatial layering. It presents the design of a coding algorithm based on Laplacian pyramid image decomposition using centered cubic spline wavelets. This coding algorithm is incorporated into a codec which, in addition to image decomposition, uses predictive coding for temporal compression and conditional arithmetic coding for coding wavelet coefficients. This results in a codec with spatial and SNR layering, but no temporal layering. Both this codec and ours use wavelets for spatial layering, although of different degree (cubic vs. quadratic), and we use them differently. In our codec, they are used for subband coding, which is different from the pyramid approach used in their coding algorithm. The paper provides extensive descriptions of visual quality through PSNR (peak signal-to-noise ratio) graphs, and also provides information on bandwidth utilization. There is, however, no information on computational complexity or CPU-usage.

Compression performance of our codec may be improved by using other methods than Huffman compression. One of the most efficient methods is the embedded zero tree wavelet coding [13], but it is most effective when using several levels of decomposition. Another recent method is conditional arithmetic coding [10]. [4] above also relies on conditional arithmetic coding for extra compression performance. We have put more effort into codec versatility at these less significant layers than compression performance, because we assume that the receivers wanting many layers typically have enough band-width to spare.

In [8], a codec is developed which 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, however, allow for several levels of subband decomposition. They include temporal layering in the codec using conditional block replenishment. In the paper, 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 receivers adjust their reception rate by joining and leaving multi-cast groups, was developed to work in environments such as the MBone [1] using IP multi-casting. Their conditional block replenishment provides good error resilience, but lacks some compression performance compared to some of good predictive coding techniques. We have stressed the need for a fast software implementation, and empirical tests. Although suggestions for implementation optimizations are presented in the paper, they present very little information about run-time performance. Evaluation of quality is presented through PSNR graphs, but there are no indications of bit-rate distributions.

New predictive temporal coding techniques [11], [18] provide excellent compression rates, but typically target a fixed low bit-rate [24], [5] and discard the remaining image information. This is useful only for fixed, low bit-rate video distribution, and is therefore inadequate to satisfy the high-bandwidth receivers' quality requirements. Moreover, the predictive coding techniques are usually computationally complex, and generally do not run in real-time. New predictive coding schemes are required that allow temporal layering, and which can be implemented to run in real-time, if they are to be incorporated in interactive video applications. Here, a SIMD architecture helps considerably to achieve this performance goal.

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 propose a high performance implementation using the SIMD capabilities of the run-time platform for speeding up coding and decoding.

Bit-rate tests show that the codec is capable of delivering frame sizes evenly distributed across a large spectrum. The lowest recognizable visual quality is at 16.4kbits, meaning that the useful range of bit-rates is as versatile as 16.4kbits to 777kbits per frame, a difference factor of 47.4:1 from the lowest to the highest bandwidth requirement. These bit-rates were obtained without temporal compression, which could improve versatility even further.

Image quality ranges from recognizable to very good as the number of layers decoded, and thereby the bit-rate, increases. The codec allows viewing at two spatial resolutions by selecting an appropriate number of DCT and wavelet enhancement layers. This allows a tradeoff between resolution and sharpness.

Our CPU-usage measurements show that the codec is capable of real-time encoding a test movie at 39fps and decoding at 28fps. This level of efficiency is achieved by using SIMD computations. We found that the VIS accelerated stages in the codec was 3-4 times as fast as an otherwise identical, efficient C-implementation; we have therefore found 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 application programmers without years of image 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 has found its way into virtually all modern CPU-architectures, we think that developers should consider using it, given the speed-ups possible. Unfortunately, the SIMD engines are not compatible between CPU-architectures, but most of them implement the same features, which can be mapped to a distinct set of instructions.

References

[1] Hans Eriksson. MBONE: The Multicast Backbone. *Communications of the ACM*, 37(8), 1994.

[2] Heine Frifeld, Peter Lindstrøm, and Simon Nybroe. Minimizing the Static - Building an Active Node. Technical report, Aalborg University, January 1998.

[3] Barry G. Haskel, Atul Puri, and Arun N. Netravali. *Digital Video: An Introduction to MPEG-2.* Chapman and Hall, 1997.

[4] 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.

[5] Faouzi Kossentini et al. Predictive RD Optimized Motion Estimation for Very Low Bit-Rate Video Coding. *IEEE Journal on Selected Areas in Communications*, 15(9):1752–1763, December 1997.

[6] Martin H. Kristiansen and Morten V. Jensen. Scaling Video for Heterogeneous Networks. Technical report, Aalborg University, January 1998.

[7] A. K. Louis, P. Maaß, and A. Rieder. *Wavelets - Theory and Applications.* Wiley-Interscience, 1997.

[8] 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.

[9] Sun Microsystems. UltraSPARC and New-Media Support, 1995.

[10] F. Müller, K. Illgner, and B. Menser. Embedded Laplacian Pyramid Image Coding Using Conditional Arithmetic Coding. In *Proc. IEEE Int. Conf. Image Processing, ICIP'96*, volume I, pages 221–224, Lausanne, Switzerland, September 1996.

[11] Mutsumi Ohta and Satoshi Nogaki. Hybrid Picture Coding with Wavelet Transform and Overlapped Motion-Compensated Interframe Prediction Coding. *IEEE Transactions on Signal Processing*, 41(12):3416–3424, December 1993.

[12] Peter Peleg, Sam Wilkie, and Uri Weiser. Intel MMX for Multimedia PC's. *Communications of the ACM*, 40(1):25–38, January 1997.

[13] Jerome M. Shapiro. Embedded Image Coding Using Zerotrees of Wavelet Coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, December 1993.

[14] Gilbert Strang and Truong Nguyen. *Wavelets and Filter Banks.* Wellesley-Cambridge Press, 1996.

[15] Sun Microsystems. *mediaLib Users Guide*, June 1997. WWW: `http://www.sun.com/microelectronics/vis/mlib_guide.pdf`.

[16] David L. Tennenhouse et al. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[17] Marc Tremblay, J. Michael O'Connor, V. Narayanan, and Liang He. VIS Speeds New Media Processing. *IEEE Micro*, 16(4):10–20, August 1996.

[18] Dimitios Tzovaras, Stavros Vachtsevanos, and Michael G. Strintzis. Optimization of Quadtree Segmentation and Hybrid Two-Dimentional and Three-Dimensional Motion Estimation in a Rate-Distortion Framework. *IEEE Journal on Selected Areas in Communications*, 15(9):1726–1738, December 1997.

[19] Michael Unser, Akram Aldroulbi, and Murray Eden. B-Spline Signal Processing: Part I-Theory. *IEEE Transactions on Signal Processing*, 1993.

[20] Michael Unser, Akram Aldroulbi, and Murray Eden. B-Spline Signal Processing: Part II-Efficient Design and Applications. *IEEE Transactions on Signal Processing*, 41(2):834–848, February 1993.

[21] Martin Vetterli and Jelena Kovačević. *Wavelets and Subband Coding.* Prentice Hall, 1995.

[22] Qi Wang and Mohammed Ghanbari. Scalable Coding of Very High Resolution Video Using the Virtual Zerotree. *IEEE Transactions on Curcuits and Systems for Video Technology*, 7:719–727, October 1997.

[23] Nicholas Yeadon et al. Filters: QoS Support Mechanisms for Multipeer Communications. *IEEE Journal on Selected Areas in Communications*, 14(7):1245–1262, September 1996.

[24] Kui Zhang, Miroslaw Bober, and Josef Kittler. Image Sequence Coding Using Multiple-Level Segmentation and Affine Motion Estimation. *IEEE Journal on Selected Areas in Communications*, 15(9):1704–1713, December 1997.