Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 49(63), Fascicola 1, 2004

A highly scalable video coding technique based on the Wavelet Transform

Dragoş N. Vizireanu[1], Radu O. Preda[2], Radu M. Udrea[3]

Abstract – We want to create a video codec, that can compress a video stream at a single bitrate and decompress it at different bitrates. Our implementation of the codec uses wavelets as its base. The encoder reads the uncompressed video stream and does a 2D Wavelet Transform on every frame. Then the SPIHT (Set Partitioning in Hierarchical Tree) algorithm is used to store the wavelet coefficients in an embedded way. The decoder can decode the compressed video stream at different bitrates, achieving variable quality. A comparison with JPEG and MPEG shows that our wavelet codec has comparable results.

Keywords: rate scalability, video compression, Wavelet Transform

I. INTRODUCTION

The objective of a video compression algorithm is to exploit both the spatial and temporal redundancy of a video sequence such that fewer bits can be used to represent the video sequence at an acceptable visual distortion. For example, it is frequently desired to transmit video over standard telephone lines, where data rates are typically restricted to 56,000 bps. A video sequence with frame size of 176_144 pixels (a size commonly used for this application) at 30 frames per second (fps) and 3 bytes per pixel, would require 18.25 Mbps, making impractical the transmission of video without compression. For different applications, different resolutions, visual quality, and therefore, different data rates, are required.

The available bandwidth of most computer networks almost always poses a problem when video is to be delivered. A user may request a video sequence at a specific data rate. However, the variety of requests and the diversity of the network may make it difficult for an image or a video server to predict, at the time the video is encoded and stored on the server, the video quality and data rate it will provide to a particular user at a given time.

Meeting bandwidth requirements and maintaining acceptable image quality simultaneously is a challenge. Rate scalable compression that allows the decoded data rate to be dynamically changed, is appealing for many applications, such as video streaming and multicasting on the Internet, video conferencing, video libraries and databases, and wireless communication. In these applications, the bandwidth available cannot be guaranteed due to variations in network load. When a video sequence is transmitted over a heterogeneous network, network congestion may occur, decreasing the quality observed by the user.

Consider the following example: A media provider digitizes, compresses, and stores news clips in a digital video library using MPEG-2 at 6 Mbits/sec (Mbps), and makes them available to the public. Most current video compression techniques and standards require that parameters, such as data rate, be set at the time of encoding. A problem exists if the server receives a request for the video sequence not at 6 Mbps, but at 4 Mbps. A solution to this problem would be to have the video server transcode the compressed bit stream. However, this is a computationally intensive task.

In general, a media producer faces the difficult task of providing content at different resolution (temporal, spatial, and/or rate) levels depending on the receivers' capability as well as possibly users' choice. One solution to this problem is to compress and store a video sequence at different data rates. The server will then be able to deliver the requested video at the proper data rate, given the network load and the specific user request. There are two problems with this approach:

- The need to store a sequence at various data rates introduces the added overhead of storage, duplicity and management of different sequences.
- For real-time applications, it is impractical to have several encoders compressing the sequence at the same time.

An alternative solution to this problem is to use a video coder that is capable of dynamically selecting the decoded data rate. This is a very attractive solution

[1] Electronics and Telecommunications Faculty, Department of Communications, Bd. Iuliu Maniu Nr. 1-3, 061071 Bucharest, e-mail nae@comm.pub.ro
[2] Electronics and Telecommunications Faculty, Department of Communications, Bd. Iuliu Maniu Nr. 1-3, 061071 Bucharest, e-mail radu@comm.pub.ro
[3] Electronics and Telecommunications Faculty, Department of Communications, Bd. Iuliu Maniu Nr. 1-3, 061071 Bucharest, e-mail mihnea@comm.pub.ro

for the flexibility it introduces to the system. This is known as "video scalability".

Video scalability is different from the concept of scalability used in networking. In this article, when we refer to scalability, we mean "video scalability". Most current compression techniques require that parameters, such as data rate, frame rate, and frame size, be set at the time of encoding, and are not easily changed. Video compression techniques that allow one to encode the sequence once and then decode it at multiple data rates, frame rates, spatial resolutions, and video quality are known as "scalable". The goal of scalable video compression is to encode a video sequence once, and decode it on any platform fed by any data pipe.

II. CODING SCHEME

Depicted in Fig. 1 is a principal sketch of our video codec. Not all steps are required for a video codec and some steps were omitted (for example, the entropy coding block was omitted). Only one frame at a time is compressed. First a color space transform is applied from the RGB color space (red, green, blue) to the YC_bC_r color space, we do a Discrete wavelet Transform and then the SPIHT (Set Partitioning In Hierarchical Tree) algorithm is used to save the wavelet coefficients in an embedded way. The video stream coded using our coder has an absolutely scalable bitrate and can be decoded at any user specific bitrate.

To uncompress an image or a video, the steps in Fig. 1 are traversed in reverse order with inverse transforms of all steps. The decoder side of the application can decode the encoded video stream at multiple bitrates, given by the available bandwidth and the users' preferences.

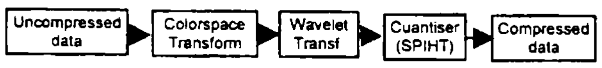


Fig. 1 Bloc diagram of the encoder

Colorspace transform

We will be using the YC_bC_r (ITU-R BT.601) format, which is a scaled and offset version of the YUV colorspace. To convert from RGB to YC_bC_r format the following equations can be used:

$$\begin{aligned} Y &= 0.257R + 0.504G + 0.098B + 16 \\ C_b &= -0.148R - 0.291G + 0.439B + 128 \\ C_r &= 0.439R + 0.368G - 0.071B + 128 \end{aligned} \qquad (1)$$

and to convert back:

$$\begin{aligned} R &= 1.164(Y-16) + 1.596(C_r-128) \\ G &= 1.164(Y-16) - 0.813(C_r-128) - \\ &\quad - 0.391(C_b-128) \\ B &= 1.164(Y-16) + 2.018(C_b-128) \end{aligned} \qquad (2)$$

In the RGB format each of the three values represents the amount of red, green and blue. In YC_bC_r format, the Y represents the amount of luminance (brightness), C_b and C_r represent the chrominance (color). In YC_bC_r format the chroma components of an image are often subsampled to a quarter of their original size. We chose not to subsample the chroma channels, instead we later compress those channels much harder.

Bi-dimensional Wavelet Transform

The easiest method to apply a bi-dimensional transform is to consider the image as rows of one-dimensional signals and to transform these rows. Then we transform them in the other direction as well. So the solution is to apply the Discrete Wavelet Transform first on the rows and then on the columns of the image, as shown in Fig. 2.

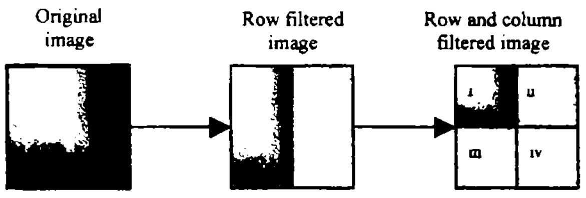


Fig. 2. Applying the Wavelet Transform on the rows and columns of an image

The four subimages are:

1) **I** – low-pass filtered on rows and columns, also known as **LL** (low subbands for row and column filtering)
2) **II** – high-pass filtered on rows and low-pass filtered on columns, also known as **HL** (high subband for row filtering and low subband for column filtering)
3) **III** – low-pass filtered on rows and high-pass filtered on columns, also known as **LH** (low subbands for row filtering and high subbands column filtering)
4) **IV** – high-pass filtered on rows and columns, also known as **HH** (high subbands for row and column filtering)

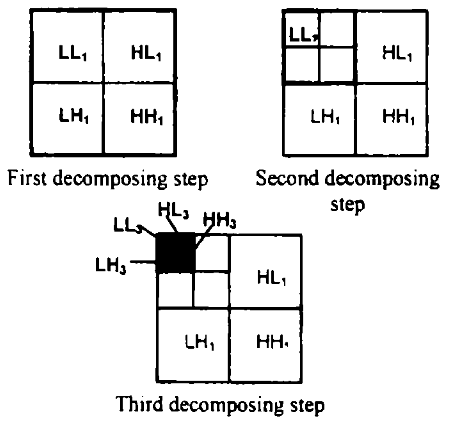


Fig. 3. Applying the 2D Wavelet Transform on tree resolution levels

After a one-dimensional transform we will have only half of the scaling coefficients s. The bi-dimensional transform is applied on columns, that contain both scaling coefficients s and Wavelet coefficients d, but on the columns there are only new scaling coefficients (obtained after the row transform), which are then used in the column transform. So, after the bi-dimensional transform we will have only 1/4 of the initial data and the following steps of transformation will need less computation time.

Applying the Inverse Wavelet Transform on the coefficients from subband LL we will obtain the the correspondent of the original image, but at one unit smaller resolution level.

At the second step we apply the Wavelet Tran^form on the subband LL. These steps can be successively repeated until we reach the wanted or the smallest permitted resolution level. Fig. 3 shows three steps in applying the Wavelet Transform and the corresponding subbands.

For example, Fig. 4 shows the position of the coefficients in the subbands after a tree resolution level decomposition of an 8x8 pixel image:

$s^3_{1,1}$	$d^{3h}_{1,1}$	$d^{2h}_{1,1}$	$d^{2h}_{2,1}$	$d^{1h}_{1,1}$	$d^{1h}_{2,1}$	$d^{1h}_{3,1}$	$d^{1h}_{4,1}$
$d^{3v}_{1,1}$	$d^{3d}_{1,1}$	$d^{2h}_{1,2}$	$d^{2h}_{2,2}$	$d^{1h}_{1,2}$	$d^{1h}_{2,2}$	$d^{1h}_{3,2}$	$d^{1h}_{4,2}$
$d^{2v}_{1,1}$	$d^{2v}_{2,1}$	$d^{2d}_{1,1}$	$d^{2d}_{2,1}$	$d^{1h}_{1,3}$	$d^{1h}_{2,3}$	$d^{1h}_{3,3}$	$d^{1h}_{4,3}$
$d^{2v}_{2,1}$	$d^{2v}_{2,2}$	$d^{2d}_{1,2}$	$d^{2d}_{2,2}$	$d^{1h}_{1,4}$	$d^{1h}_{2,4}$	$d^{1h}_{3,4}$	$d^{1h}_{4,4}$
$d^{1v}_{1,1}$	$d^{1v}_{2,1}$	$d^{1v}_{3,1}$	$d^{1v}_{4,1}$	$d^{1d}_{1,1}$	$d^{1d}_{2,1}$	$d^{1d}_{3,1}$	$d^{1d}_{4,1}$
$d^{1v}_{1,2}$	$d^{1v}_{2,2}$	$d^{1v}_{3,2}$	$d^{1v}_{4,2}$	$d^{1d}_{1,2}$	$d^{1d}_{2,2}$	$d^{1d}_{3,2}$	$d^{1d}_{4,2}$
$d^{1v}_{1,3}$	$d^{1v}_{2,3}$	$d^{1v}_{3,3}$	$d^{1v}_{4,3}$	$d^{1d}_{1,3}$	$d^{1d}_{2,3}$	$d^{1d}_{3,3}$	$d^{1d}_{4,3}$
$d^{1v}_{1,4}$	$d^{1v}_{2,4}$	$d^{1v}_{3,4}$	$d^{1v}_{4,4}$	$d^{1d}_{1,4}$	$d^{1d}_{2,4}$	$d^{1d}_{3,4}$	$d^{1d}_{4,4}$

Fig. 4. Distribution of the scaling and Wavelet coefficients for an 8x8 pixel image decomposed on 3 resolution levels

where $d^{jh}_{x,y}$ is the Wavelet coefficient at resolution level j and position (x,y) from subband LH, $d^{jv}_{x,y}$ is the Wavelet coefficient at resolution level j and position (x,y) from subband HL and $d^{jd}_{x,y}$ is the Wavelet coefficient at resolution level j and position (x,y) from subband HH. Note that we have only one scaling coefficient $s^3_{1,1}$ on poziția (1,1) in subband LL_3. This coefficient is an approximation of the original image at resolution level 3 (1x1 pixels). The Wavelet coefficients d hold the details of the image at for corresponding resolution level. For lower resolution levels (higher resolution) the details contained in the Wavelet coefficients are finer.

For the reconstruction (synthesis) of the original image, we apply the inverse algorithm. From the scaling coefficient $s^3_{1,1}$ we get the approximation of the original image at resolution level 3 (1x1 pixel resolution). Using the scaling coefficient $s^3_{1,1}$ and the level 3 Wavelet coefficients $d^3_{x,y}$ we get the image at resolution level 2 (2x2 pixel resolution). Using also the level 2 coefficients $d^2_{x,y}$, we obtain the image at resolution level 1 (4x4 ixel resolution), and, finally, using the coefficients $d^1_{x,y}$ we can reconstruct the the original image at resolution 8x8 pixels. Fig. 5 shows a two level decomposition of the image "Lena" of resolution 256x256 pixels. The image in the upper left corner is a approximation of the original image at resolution level 2 (64x64 pixel resolution). The other images were obtained only from the Wavelet (detail) coefficients from the corresponding subband and they are the detail images of the corresponding subbands.

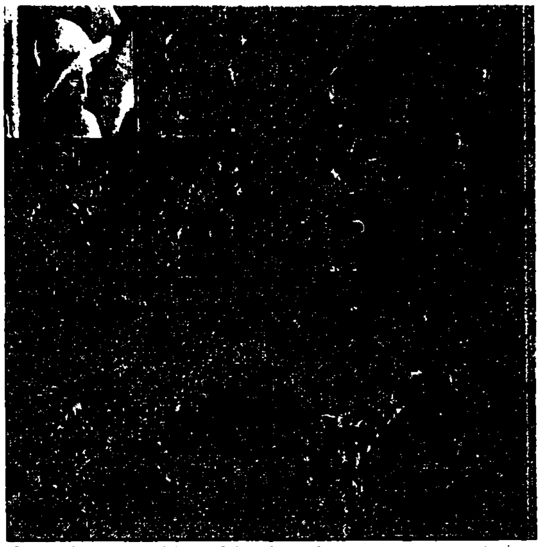


Fig. 5 Decomposition of the "Lena" image on two resolution levels

Cuantisation

So far, as our video codec is concerned, no information has been lost, and no actual compression has taken place.

Usually the most important coefficients will be grouped in the top left corner (see Fig. 3), with importance decreasing for each sub band, and a lot of the coefficients in the lower sub bands will have a value close to zero. Since small values represent small changes in the original image, they can often be discarded without visible loss of quality. Discarding low valued coefficients will increase compression for a modest degradation in quality. However, doing so will lead to a problem: the coordinates of the coefficients. A lot of coefficients will be zero, and storing them gives no compression. On the other hand, not storing them leads to the problem with coordinates: what coefficient should be at what coordinate? One solution is to define a scan order to follow. Two such scan orders are depicted in Fig . 6 for a eight by eight matrix.

The coefficients are simply stored in the order they are found according to scan order. When the desired compression ratio is achieved, storing is stopped. With this technique some coefficients that are stored are close to zero or zero, thus lowering the compression ratio. There exist a number of algorithms that address this problem. The EZW (Embedded Zero Tree) algorithm by Shapiro [6] is a way to both quantize and store the coordinates of the coefficients. The SPIHT (Set Partitioning in Hierarchical Trees) algorithm by Said and Pearlman [5] can be described as a more advanced version of the EZW algorithm. Both make use of "spatial orientation trees" (SOT). Spatial orientation trees are structures that use quad-tree representations of sets of wavelet coefficients that belong to different subbands, but have the same spatial location. These structures, which can be efficiently represented by one symbol, have been used extensively in rate scalable image and video compression.

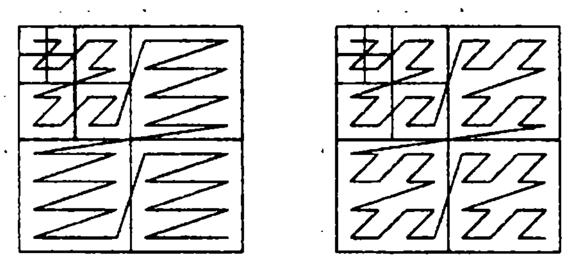


Fig 6. Coefficient scanning order: a)Raster scan order b)Morton scan order

The SPIHT algorithm

Said and Pearlman [5] investigated different tree structures that improved the quality of the decomposition. The SPIHT (Set Partitioning in Hierarchical Tree) algorithm is also an algorithm that stores the most important coefficients in an embedded way, along with information for the coordinates. Quantization is done implicitly by starting with a high threshold that is successively lowered as the algorithm progresses.

Examine Fig. 5 and note the similarity between the different subbands. A tree like structure can be built from this observation, where each node have four branches except the top node which is a special case (refer to Fig. 7). A closer inspection of Fig. 5 shows that usually a bright patch in the lower subbands seems to propagate into the higher sub bands. In this figure, white stands for a low value and black stand for a high value. Low values stand for small changes in the original image, and are thus of lesser importance.

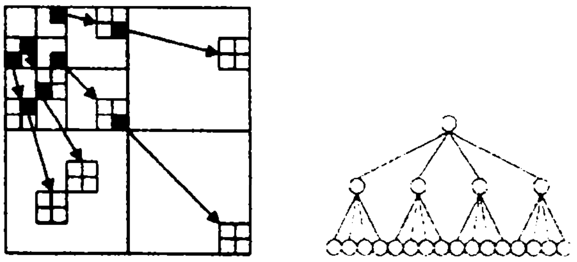


Fig. 7. An illustration how the parents expand to four clustered children in a filter bank set of coefficients.

What we want to do is to store the values of all the coefficients above a certain threshold along with information about the coordinates. Since we have this tree like structure, we can represent all the coordinates of a branch with no coefficients over a certain threshold with just one bit.

The Spiht algorithm uses tree lists to keep track of the nodes. The lists are:

- *list of insignificant pixels (LIP):* contains nodes, that will be checked for significance
- *list of insignificant sets (LIS):* contains nodes whose descendants will be checked for significance
- *list of significant pixels (LSP):* consists of nodes that have been found signi_cant earlier. The nodes in this list can be of two kinds:
  - *normal nodes (type A)*: are nodes, that have insignificant children
  - *multi-noduri (de tip B)*: are nodes, that have significant children

The SPIHT algorithm can be summarized as follows:

1. *Initialisation:*
   - Compute $n = \left|\log_2\left(\max_{(i,j)}\{|c_{i,j}|\}\right)\right|$, where $c_{i,j}$ are the coefficients and initialize the treshold $T = 2^n$.
   - Move the four highest nodes of the tree-like structure in the LIP and the tree direct descendents of the first node in the Lis, marked as normal nodes. The LSP is empty.
2. *Sorting pass*
   - Traverse the LIP testing the magnitude of its elements against the current threshold and representing its significance by 0 or 1. If a coefficient is found to be significant, it is moved from LIP to LSP and a "1" bit is written at the output, showing that the coefficient was significant; then another bit is written at the output, showing the sign of the coefficient (0 for positive, 1 for negative). If the coefficient is insignificant, a bit "0" is written at the output.
   - For every node in the LIS, determine if it is a normal node or a multinode.
     - If it is a multimode, its children are checked for significant descendents and a bir is written at the output. If significant descendents are discovered, all four children are added to the LIS marked as normal nodes and the current node is removed from the LIS.
     - If it is a normal mode, it is checked for significant descendents in respect to the current threshold and a bit is written at the output. If it has significant descendents with significant descendents (grandchildren), it is moved to the end

of the LIS marked as multimode; else it is removed and the sorting pass with a temporal LIP list (that contain the children of the nodes) is executed. The children, that haven't been moved to the LSP during the sorting pass are moved to the actual LIP list.

3. *Refinement pass*
   - Compare all nodes in the LSP with the current threshold T, except those added during the sorting pass and write the corresponding bits at the output.
4. Set T=T/2 and go to step 2.

The process is repeated until the target data rate is achieved. It is important to note that the location of the coefficients being refined or classified is never explicitly transmitted. This is because all branching decisions made by the encoder as it searches hroughou he coe ficien s are appended o he bi stream.
Decoding is done in a similar manner: all output parts are exchanged for input and all comparisons with threshold are exchanged for updates to the coefficients. When updating the coefficients, their initial value is set to 1.5 times the current threshold, beca se he hreshol ells s that he val is a leas threshold but less than two times threshold. So a good guess might be that the value will be somewhere in between.
The performance of SPIHT is better than EZW, at a modest increase in the computational complexity [5]. Th[illegible] hy [illegible] SPIHT [illegible]

III. RESULTS

For evaluating the performance of our codec we are using the Peak Signal to Noise Ratio (PSNR), a measurement commonly used when comparing images between each other. The PSNR is calculated as:

$$PSNR = 10\log\frac{255^2 NM}{\sum_{i,j}(x(i,j)-\tilde{x}(i,j))^2} \quad (3)$$

Here the 255 comes from the maximum value we can have anywhere on one bitplane of the picture, multiplying it with N, which is the size of a bitplane, in our case 128 x 128. The transformed and retransformed image elements are named $\tilde{x}(i,j)$ and they are of course compared to the original image elements $x(i,j)$. Based on our own observations we have noticed that a picture lying above 30 is of tolerable quality, and above 40 is close to flawless.

Results for scalable coding of images
First we show a comparison in PSNR between JPEG an our co ec in ig. 8. The pic ure we use in comparison is the classical color picture "Lena", often used in image processing. The resolution of the image is 256x256 pixels. For processing the JPEG pictures we used the ImageMagick program. For de wavelet decomposition and reconstruction we used two types of filters: the Haar filter with modest performence and the biorthogonal filter "bior4.4" of order 4 with much higher performance. The graphs show that JPEG is slightly superior to our wavelet coder, except for very low bit rates. JPEG's decline probably comes from the fact that the image is divided into 8x8 pixels large blocks before the Discrete Cosinus Transform (DCT) is applied, and the overhead becomes significant at low bit rates. When comparing the biorthogonal filter with the Haar filter, the difference in PSNR is about 2 dB. Comparing JPEG with the "bior4.4" filter, we end up just 1 dB lower in PSNR. Note that our codec does no entropy encoding after the SPIHT algorithm, while JPEG uses a Huffman coding, which gives an extra gain in PSNR.

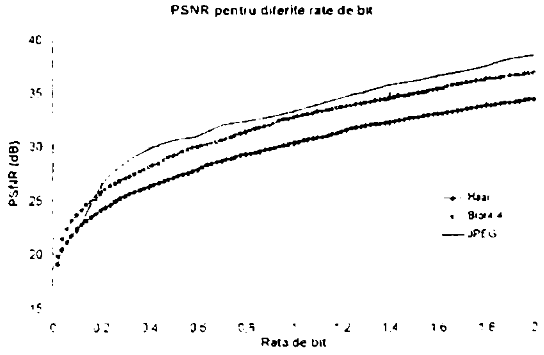
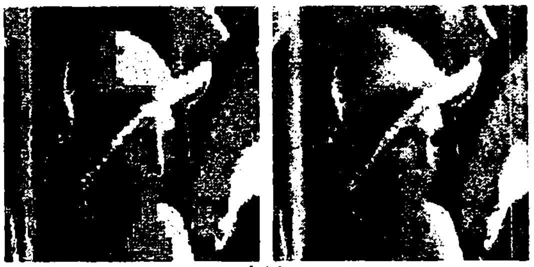
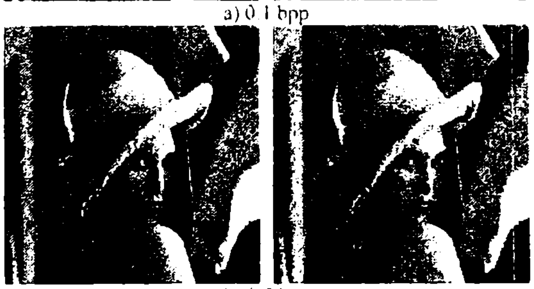


Fig 8 Comparison between JPEG and our wavelet coder for the "Lena" picture (256x256 pixels)


a) 0.1 bpp


b) 0.5 bpp

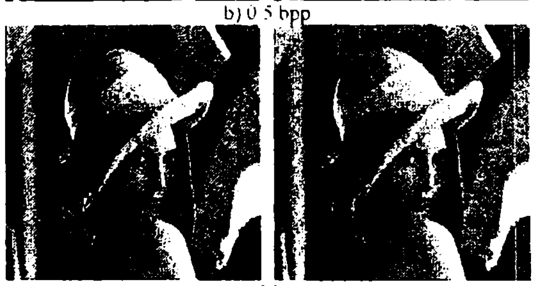

c) 1 bpp

Fig 9. Visual results obtained for the color image "Lena" with JPEG (left) and with our codec (right) for different bitrates

Fig. 9 shows the JPEG images and the images obtained with our coder ("bior4.4" filter) for different bitrates. We can see a better visual quality in case of our coder.

Results for scalable video coding

Fig. 10 and 11 show a comparison between MPEG-1 and our codec. The graph shows PSNR for the 128 first frames of the "foreman" video (which was cropped from its original size of 176x144 to 128x128). The MPEG-1 encoding was made with a program called "TMPGEnc" with default parameters for all options. Our coding application compresses the input video stream at a high data rate (5 bpp) so that clients with high as well as with low bandwidth to be able to achieve the best quality at their given bandwidth after decoding. The decoder side of the application can decompress the coded bitstream at different data rates.

In Fig. 10 the wavelet decoder works at a data rate of 1 bpp. We can see that the PSNR for MPEG-1 is about 4 dB better than our encoder. This advantage is obtained because MPEG uses an inter-frame coding scheme, which gives the extra gain in PSNR and is not used in our codec.

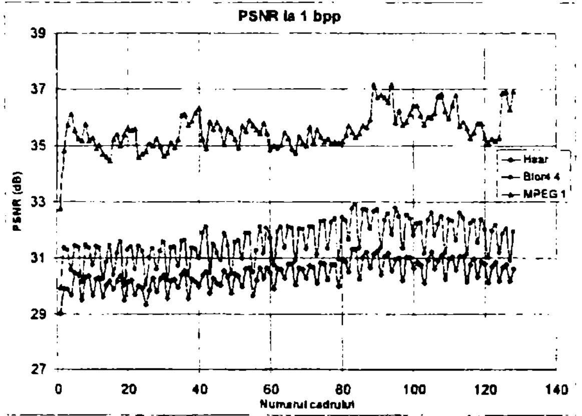


Fig. 10. Comparison between MPEG-1 and our wavelet coder for the "forman" video (resolution 128x128 pixels, 128 frames) at 1 bpp

In Fig. 11 our decoder works at a bitrate of 0.1 bpp. You can notice that our application has equal results with MPEG at very low bitrates.

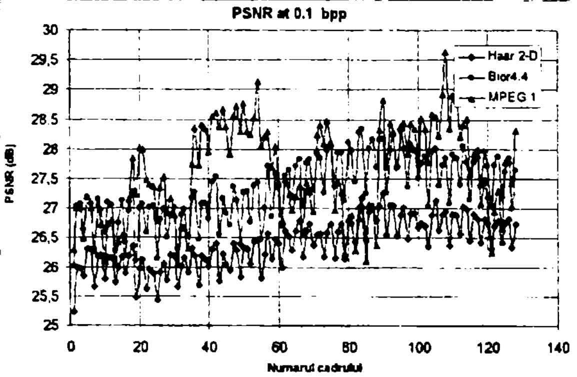


Fig. 11. Comparison between MPEG-1 and our wavelet coder for the "forman" video at 0.1 bpp

IV. CONCLUSIONS

The Wavelet Transform combined with the SPIHT algorithm is a very effective way of building a scalable video codec. The motion compensation techniques used in MPEG formats may be of some help in compressing the video stream and increasing the quality, but then the scalability is harder to achieve. For a gain in performance and preservation of scalability we should switch from 2D Wavelet Transform and SPIHT to their 3D versions. This should work by compressing several frames at once. The advantage is that the redundancy between consecutive frames is exploited and we can achieve higher compression ratios. Another way to improve our codec is to optimize the code. An entropy coder would improve things further, but not by very much. With these improvements we could equal and even beat the performance of MPEG.

REFERENCES

[1] T. Akiyama, T. Takahashi and K. Takahashi, *Adaptive three-dimensional transform coding for moving pictures*, Proceedings of the Picture Coding Symposium, Cambridge, MA, 1990.

[2] I. Hontsch and L. Kara, *Locally adaptive perceptual image coding*, IEEE Transactions on Image Processing, vol. 9, September 2000.

[3] C. Parisot, M. Antonini and M. Barlaud, *3D scan based wavelet transform for video coding*, Proceedings of the IEEEWorkshop on Multimedia Signal Processing, Cannes, France, 2001, pages 403-408.

[4] M. Saenz, P. Salama, K. Shen, and E. J. Delp, *An evaluation of* [illegible] *w* [illegible] *im g* [illegible] *p* [illegible] *h q* [illegible], [illegible] I Conference on Visual Communications and Image Processing '99, pages 282-293, San Jose, California, January 25-27, 1999.

[5] A. Said and W. A. Pearlman, *A new, fast, and efficient image codec based on set partitioning in hierarchical trees*, IEEE Transactions on Circuits and Systems for Video Technology, vol. 6, no. 3, pages 243-250, June 1996.

[6] J. M. Shapiro, *Embedded image coding using zerotrees of wavelet coefficients*, IEEE Transactions on Signal Processing, vol. 41, pages 3445-3462, December 1993.

[7] K. Shen and E. J. Delp, *Wavelet based rate scalable video compression*, IEEE Transactions on Circuits and Systems for Video Technology, vol. 9, no. 1, pp. 109-122, February 1999.

[8] Y. Sun, H. Zhang and G. Hu, *Real-Time Implementation of a New Low-Memory SPIHT Image Coding Algorithm Using DSP Chip*, IEEE Transactions on Image Processing, Vol. 11, No. 9, September 2002.

[9] Y.Wang, J.Ostermann, Y.Q.Zhang, *Video Processing and Communications*, Prentice Hall, 2002.