DVI®† Image Compression - Second Generation

Stuart J. Golin

Multimedia Products Operation
Intel Corporation
Plainsboro, NJ, 08536

ABSTRACT

The goal of this compression algorithm is to exploit the i750®† video processor to achieve VCR quality video at CD-ROM data rates.

The algorithm uses motion compensation with fractional-pixel displacements to produce good predicted images. The residual errors are then encoded using pyramid and vector-quantization (VQ) techniques. A binary tree is used to efficiently encode large areas of the image with the same displacement vector, and large areas where the residual errors quantize to zero. The remaining quantities are statistically encoded for the programmable statistical decoder of the i750 video processor.

This second-generation algorithm has achieved its goal and is now incorporated in current DVI products.

1. INTRODUCTION

The off-line compression algorithm used by DVI technology is designed to produce high quality video on the i750 video processor at CD-ROM data rates. This video processor consists of a pixel processor and a display processor.

A block diagram of the pixel processor is shown in Fig. 1. Its most important feature is its programmability. Its on-chip cache holds 512 instructions of 48 bits each. The cache can be completely reloaded in about 0.4% frame times. Also there is on-chip memory for 1024 bytes of data. The instruction set has been optimized for pixel manipulation and decoding. An example of a very useful instruction is add-with-saturate. This instruction adds two pairs of 8-bit numbers, and clips the results to the range 0 to 255. This makes the application of VQ very fast. The pixel processor has a statistical decoder that supports efficient variable-length encoding. It has a pixel interpolator that supports motion displacements of a fraction of a pixel. It has VRAM registers that auto-increment, which is important for processing pixels in raster order.

The display processor, shown in Fig. 2, receives the decoded image from the pixel processor. The chrominance in that image is subsampled 4 to 1 both horizontally and vertically. The UV interpolator expands the color components of the image to full resolution for display.

2. OVERVIEW

Fig. 3 contains a block diagram of the steady-state encoding of an image sequence. The first step in encoding is the motion analysis. The current image is compared with the previous one, and displacement vectors are determined that approximate the motion between the two. These displacements are applied to the previous reconstructed image to predict the image. They are also transmitted to the channel, as indicated by the thick arrows. The difference between the original and predicted images is the error image, which is then encoded and transmitted to the channel. It is also decoded and added to the predicted image to get the reconstructed image, which is identical to the one the decoder will reconstruct. That image is saved and then used to predict the next image.

---

†DVI and i750 are registered trademarks of Intel Corporation.

 0-8194-0811-5/92/$4.00

Fig. 4 shows the decoder. The decoder gets the displacement vectors from the channel and applies them to the previous reconstructed image to predict the current image. It then decodes the (prediction) error image and adds it to the predicted image, yielding the reconstructed image, which it then displays. It saves this image for one frame time and then uses it to predict the next image. Note the similarity of the decoder with the right half of the encoder in Fig. 3.

Since the first frame of a sequence cannot be based on a previous image as above, it must be encoded as a still. It is also convenient to encode some images within a sequence as stills, as at scene changes. Except for the use of prediction images, the encoding of still images closely follows that of moving images.

The typical DVI image is 256H x 240V, and 30 are displayed per second. When the input is NTSC, alternate fields are selected, and the images are horizontally filtered and subsampled to a resolution of 256 pixels. The encoder is typically allowed an average of 4500 bytes per image.

3. STATISTICAL ENCODING

The i750 video processor has a very simple and powerful statistical decoder, which is defined by a table of code forms. The codes are composed of two parts, a header and a binary number. The headers are what is known as a comma code: there are 0, 1, 2, ... 1's followed by a 0, the "comma". Table I illustrates a table with three forms, where the x's represent the number of bits in the binary number, which is separated from the header for readability. It shows how the code forms are expanded into codes. The values represented by these codes are the non-negative numbers. (Typically they are table indices.) Here, the first entry contains one x, defining codes for two values: 0 and 1. The next entry has zero x's, defining the code for one value: 2. The last entry has two x's, defining codes for four values: 3, 4, 5, and 6.

Table I. Example of a table of the statistical decoder.

form #	form hdr #	code hdr #	value
1	0 x	0 0	0
		0 1	1
2	10	10	2
3	110 xx	110 00	3
		110 01	4
		110 10	5
		110 11	6

In the i750 statistical decoder, each code form may have from 0 to 6 x's; the numbers are chosen for encoding efficiency based on the statistics of the data. Good and even optimal codes can be calculated rapidly, as is described elsewhere.[1] This scheme is very efficient, typically within about 1% of Huffman coding.

Generally, small values are encoded with the fewest bits, so it is important that the probabilities of the values decrease approximately monotonically. If this in not the case, one must sort the data and transmit a translation table. However, perfect order is not necessary. Our compression scheme uses VQ, and the codebooks are not sorted for each image. The encoding of these vectors typically uses an additional 1 or 2% more bits for this reason; which is less that the cost of a translation table. Partial reordering, however, would probably be cost effective.

An advantage of this scheme is that a table can be described with very few bytes, an average of about 3 bytes for the current algorithm. Also important is its simplicity. Because of the complexity of true Huffman encoding, it is very common for codecs to use fixed tables.

4. MOTION COMPENSATION

Motion displacements are calculated for each 8x8 block, typically to a resolution of 1/8 pixel. While this is much more detailed than that used by the MPEG standard, which specifies a displacement resolution of 1/2 pixel for 16x16 blocks, motion compensation still uses only about 10% of the bits in the typical case.

The bit usage is controlled in two ways. First, each unique displacement is encoded only once, and a displacement index is encoded for each block. The displacements are sorted in descending order of their frequencies of occurrence, and the indices are statistically encoded. Secondly, adjacent blocks with the same displacement are often encoded as a group. The grouping is done with the binary tree.

The binary tree is convenient for segmenting an image into areas that are homogeneous in some way. In this approach the image is recursively split until it is sufficiently homogeneous, or until some minimum size is reached. The splitting is done in a general manner: there is no strict alternation of horizontal and vertical splitting, and the splits are not constrained to bisect the image into two equal parts. While a constrained split would be cheaper to encode, the general approach reduces the total number of splits needed to read homogeneity, and reduces the number of displacement indices that must be encoded. This results in a net savings of bits.

The use of the binary tree is illustrated with the table-tennis sequence used by the MPEG committee. Fig. 5 shows the displacements superimposed on the image, with solid white lines separating blocks with different displacements. Fig. 6 shows the binary-tree grouping of these displacements. The light rectangles contain blocks of a single displacement, and the small, dark rectangles contain more than one displacement. In the latter case, it was not felt worthwhile to further subdivide. The existence of large light rectangles demonstrate the utility of the method.

While Fig. 6 shows the end results of the binary-tree subdivision, the order of splitting is only approximately indicated. The first split, for example, must be one of the four vertical splits that cut the full image. This follows since these lines are unbroken, and because no other line cuts the full image. However, it is unclear which of the four is first.

The encoding of the displacements is most efficient when there are large areas with the same displacement. For this reason, the displacement field is smoothed: adjacent blocks with similar displacements are merged if the resulting mean-square-error does not increase too much.

5. ENCODING THE RESIDUAL IMAGE

It is known that the human visual system requires a different update rate for different types of imagery. For example, it will tolerate a relatively low update rate for high spatial frequency detail. To take advantage of this fact, the DVI algorithm accurately encodes, or "refreshes", only every third image, and minimally encodes the intermediate ones. This approach is also common in MPEG encoding.

Stills and refreshed images are encoded using pyramid[2] techniques. In the pyramid encoding of an image, the image is filtered and subsampled into images of successively lower resolution, termed the Gaussian[2] images, and designated G_i, $i \geq 0$. G_0 is the full-resolution image. The Gaussian image of lowest resolution is encoded, and then expanded and interpolated up to the next higher level of resolution. This approximates the image at that resolution; the error is termed the Laplacian image[2] and designated L_i, $i \geq 0$. The Laplacian image at that resolution is encoded, added back to the approximation to form the reconstructed Gaussian image at that level. The process is continued until the full-resolution Laplacian image is encoded. With one more step, the encoder has the reconstructed image: it decodes L_0 and adds it to the approximation to G_0. (Although Fig. 3 shows the encoder having a separate block for pyramid decoding, it is seen that the pyramid encoder does the decoding as a matter of course.) Fig. 7 shows the encoding of a 3-level pyramid. The thick arrows indicate data that is written to the channel.

The corresponding decoder is shown in Fig. 8; it is just the right half of the encoder in Fig. 7.

All pyramid levels are encoded with VQ, whereby n pixels are represented by a single code; n is the dimension

of the vector, and is any integer larger than zero. It is important for decoding speed that the codebook fit into the on-chip data memory of 1024 bytes, making it difficult to use vectors with a large dimension.

The largest dimension used is 4, whereby blocks of 2x2 pixels, or quads, are encoded as a unit. When there are not enough vectors to adequately encode the errors, 2-dimensional vectors, or dyads, are used. In the typical case, the full-resolution level, L_0, which is quantized the most coarsely, is encoded with quads, and the intermediate levels are encoded with dyads. The lowest-resolution level, G_2 in the above example, is typically encoded with 1-dimensional vectors, or scalars. This level uses a simple DPCM scheme, whereby each pixel is predicted by the one to its left.

Each resolution level of the residual image is divided into 8x8 blocks. Just as the binary tree is used to efficiently encode displacement vectors, it is used here to efficiently encode blocks whose pixels all quantize to zero.

In part for decoding speed, the non-refreshed images are usually encoded without the pyramid. They use VQ, typically quads, and use the binary tree. The latter is especially important because these images are minimally encoded.

6. BIT-RATE CONTROL

The bit rate is regulated via a modified proportional control. The quantization for image n+1 is calculated as follows:

$$q^2_{n+1} = q^2_n \, [1 + k(\langle b_n \rangle - R)] \, \langle C_{n+1} \rangle \, / \, \langle C_n \rangle \qquad (1)$$

where k is the strength of the control feedback, R is the desired bit rate, b_n is the bit rate of the nth image, and C_n is a measure of the complexity of compressing the image. The angle brackets indicate a weighted average, over recent images in the case of b, and over recent and future images in the case of C. (Obviously, the latter requires off-line compression.) The main reason for the averaging is to prevent erratic behavior. This is especially important because every third image tends to be much larger than the rest. Were the complexity terms absent, this would be simple proportional control.

The complexity is an estimate of the mean-square-error of the predicted image. It is calculated in an initial pass over the sequence, and uses only original images. The reason for including the complexity in the bit-rate control is the expectation that, for constant bit rate, C/q^2 should be a constant. To the extent that this is true, the feedback term is not needed. While certainly not exact, it reduces the dependence on feedback and improves stability.

7. CONCLUSIONS

The DVI compression algorithm demonstrates that VCR-quality video at CD-ROM data rates can be achieved with inexpensive, well-designed hardware. It also demonstrates that VQ is effective even at low dimensions.

8. ACKNOWLEDGEMENTS

Brian Astle was the first in our group to recognize the need for image compression, and to demonstrate feasibility. Michael Keith implemented most of the microcode, and helped figure out how to make real-time decoding possible. Adnan Alattar, Judith Goldstein, and Gerald Waxler contributed code or microcode. Too many people to mention contributed ideas, insight, and encouragement at critical times in the development of this algorithm and its predecessor. However, among these, Allen Simon must be acknowledged.

9. REFERENCES

1. S. J. Golin, "The DVI Statistical Decoder", submitted for publication.

2. P. J. Burt and E. H. Adelson, "The Laplacian Pyramid as a Compact Image Code", *IEEE Trans. Comm.*, COM-31, pp. 532-540, April 1983.

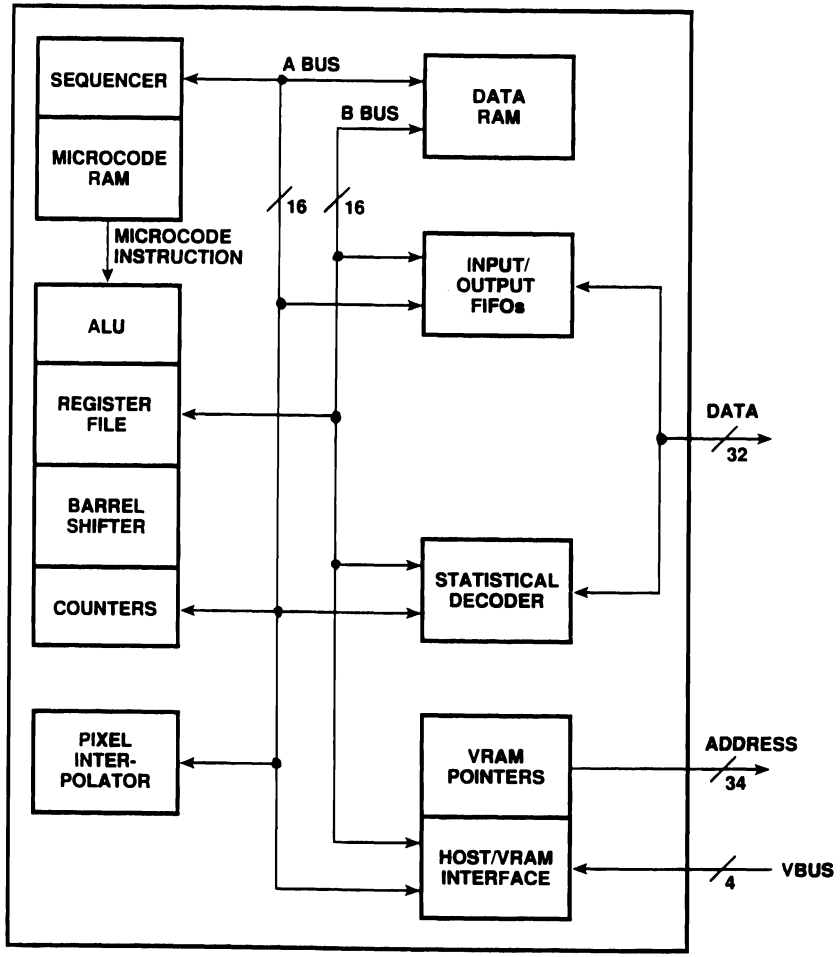

Fig. 1. Block diagram of the i750 pixel processor.

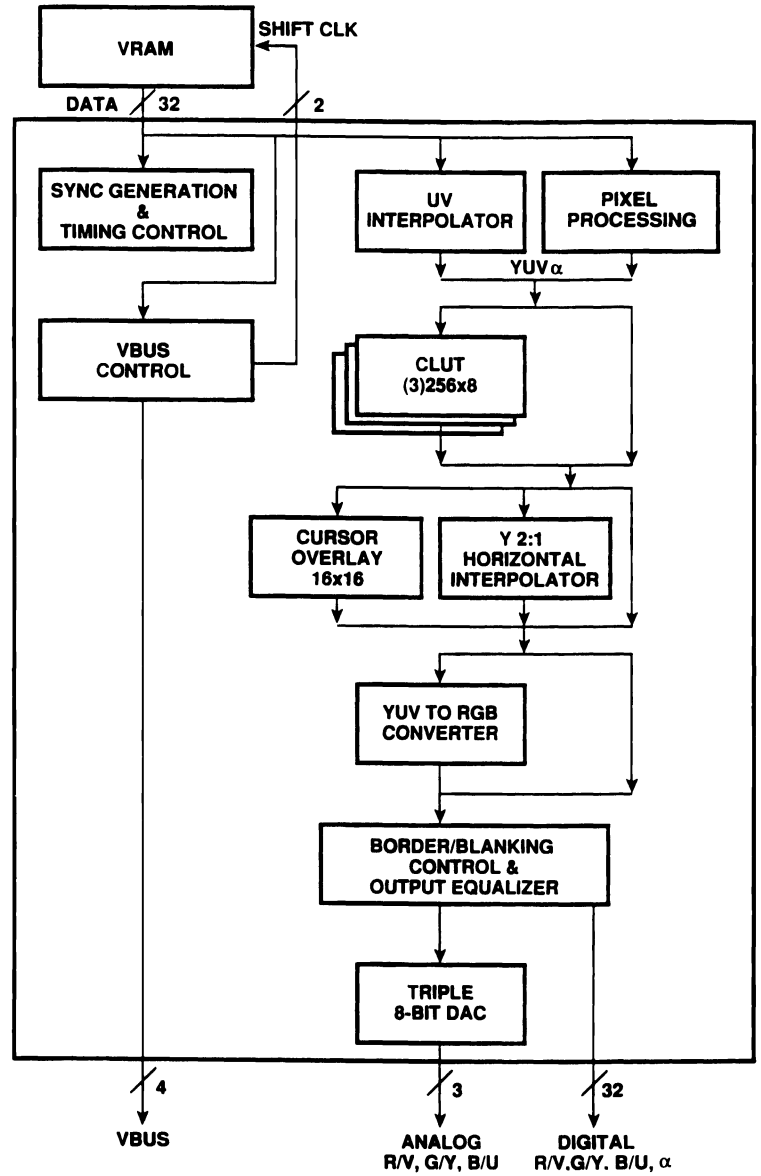

Fig. 2. Block diagram of the i750 display processor.

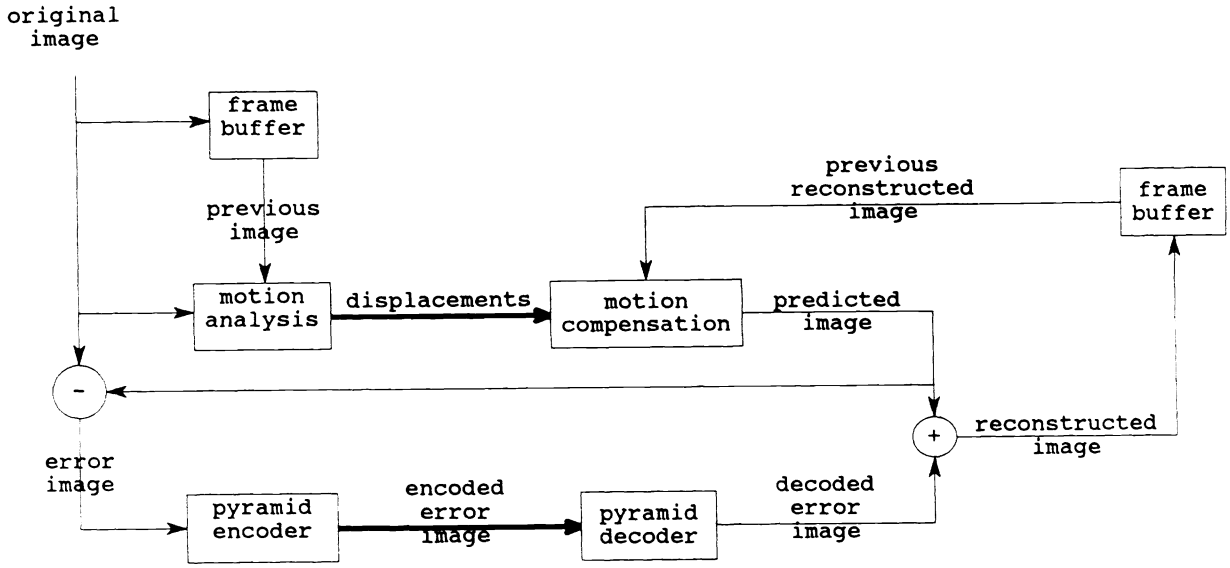

Fig. 3. Block diagram of the encoder in the steady-state.

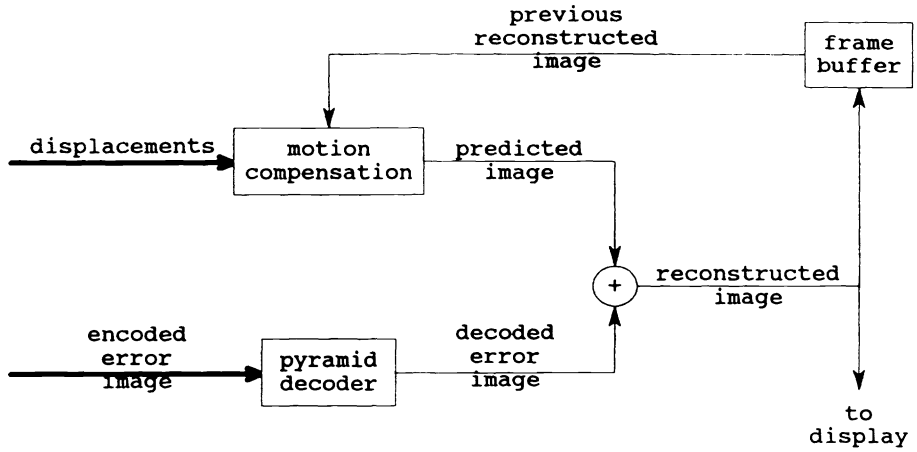

Fig. 4. Block diagram of the decoder in the steady-state.

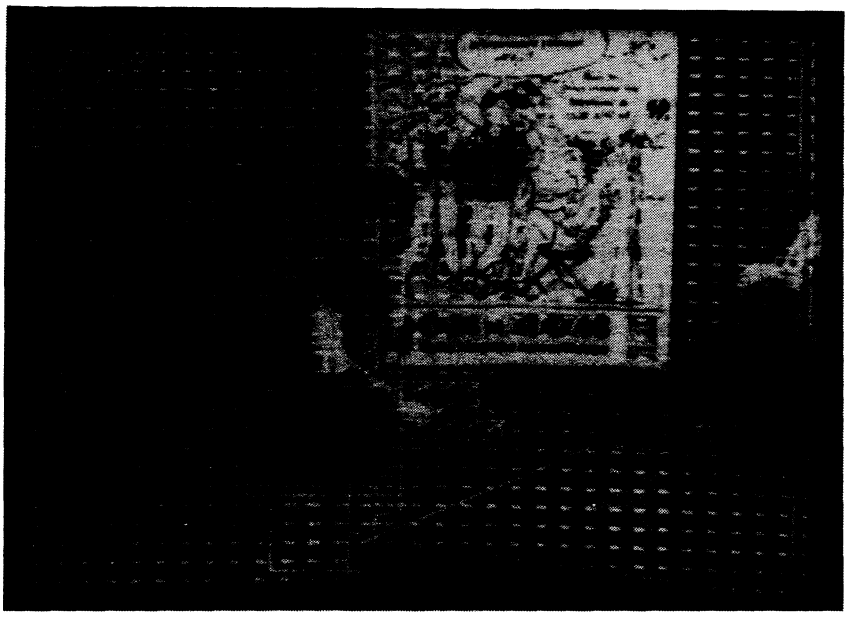

Fig. 5. Displacement vectors relating an image in the table-tennis sequence to the previous one. A solid white line separates blocks with different displacements.

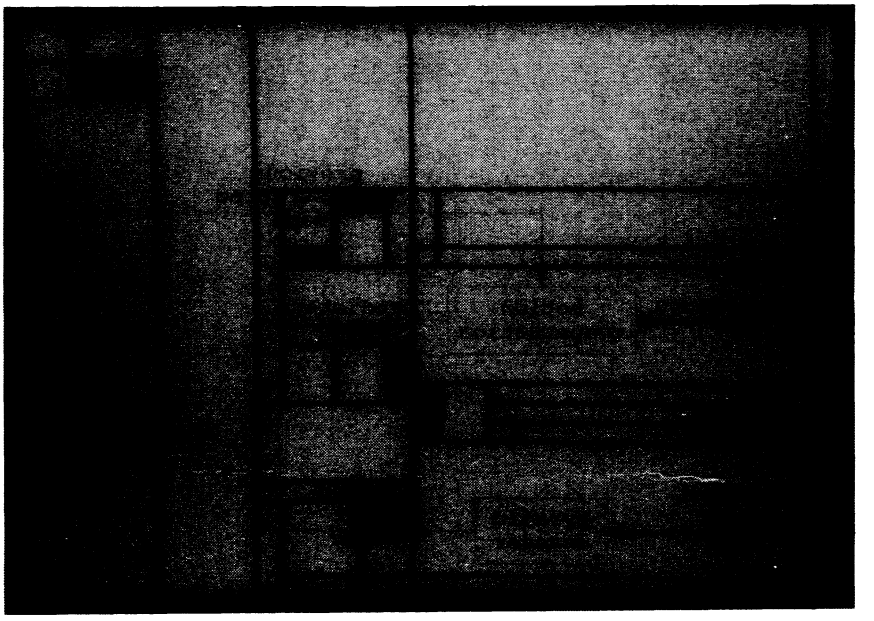

Fig. 6. The segmentation of the displacement field of Fig. 5. using the binary tree. The lines show the splitting. The white rectangles contain blocks of a single displacement, and the shaded rectangles contain blocks with more than one displacement.

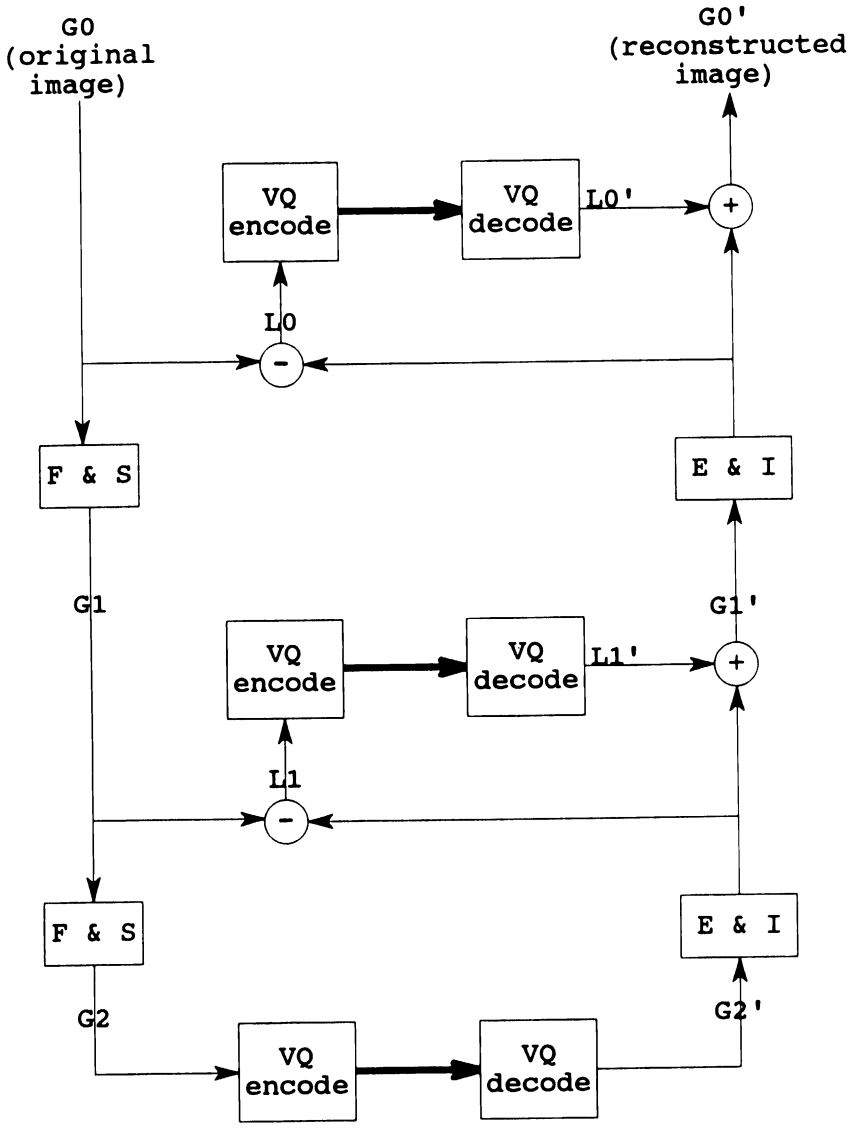

Fig. 7. Block diagram of a 3-level pyramid encoder. The primes indicate reconstructed images. "F & S" means filter and subsample. "E & I" means expand and interpolate.

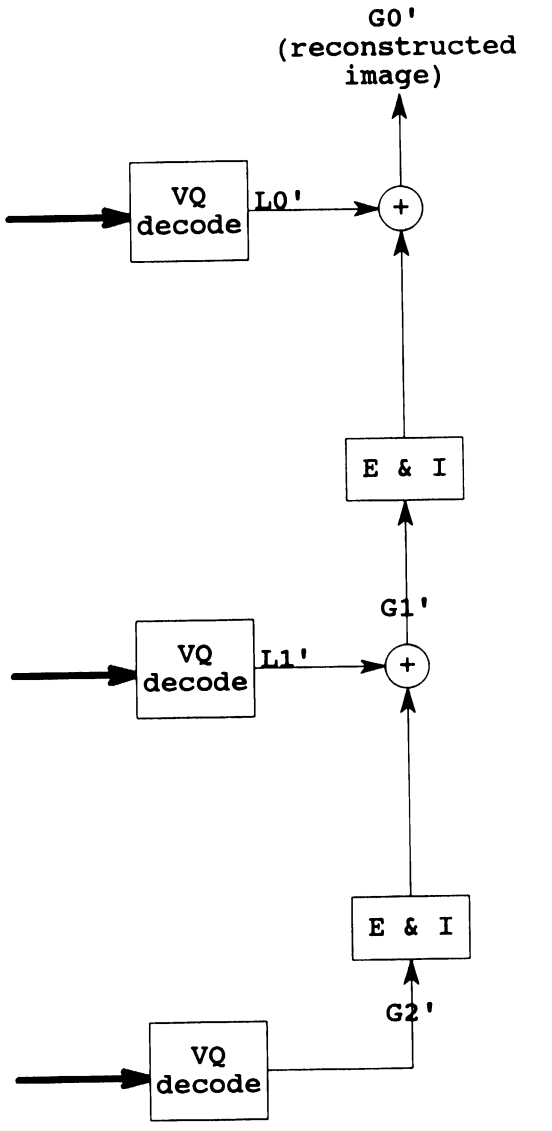

Fig. 8. Block diagram of a 3-level pyramid decoder.