Tree Triangular Coding Image Compression Algorithms

Article *in* International Journal of Image and Graphics · October 2001

DOI: 10.1142/S0219467801000414 · Source: DBLP

2 authors:

Munaga V N K Prasad
Institute for Development & Research in Banking Technology
110 PUBLICATIONS **401** CITATIONS

SEE PROFILE

Kaushal K Shukla
Indian Institute of Technology (Banaras Hindu University) Varanasi
123 PUBLICATIONS **751** CITATIONS

SEE PROFILE

Image Compression by B-Tree Triangular Coding

Riccardo Distasi, Michele Nappi, and Sergio Vitulano

***Abstract*— This paper describes an algorithm for still image compression called B-tree triangular coding (BTTC). The coding scheme is based on the recursive decomposition of the image domain into right-angled triangles arranged in a binary tree. The method is attractive because of its fast encoding, $O(n \log n)$, and decoding, $\Theta(n)$, where n is the number of pixels, and because it is easy to implement and to parallelize. Experimental studies indicate that BTTC produces images of satisfactory quality from a subjective and objective point of view. One advantage of BTTC over JPEG is its shorter execution time.**

I. Introduction

MANY compression techniques are symmetric with respect to the computing time required for encoding and decoding, for instance, the standard JPEG [1], [2] and wavelet-based codecs [3]. However, it is desirable to have algorithms that are asymptotically faster, at least in decompression, because rapid decoding is useful in a wide range of applications such as image retrieval and communication [4]–[7].

The coding scheme described in this paper is based on domain decomposition and linear interpolation. The decomposition scheme involves triangulating the domain recursively. Linear interpolation has been chosen since, while yielding an acceptable quality in the coded image, it has a quite convenient computational cost if compared to other methods [8]: the interpolation performed by B-tree triangular coding (BTTC) requires only four floating-point multiplications and one floating-point division per pixel. As a result, it runs much faster than the standard techniques based on transforms [1], [3], especially in decoding.

One of the ways to decompose the domain uses quadtrees [9], for which several asymptotic results are known [10]. However, quadtrees are only applicable to binary or multicolor images, where the term "multicolor" denotes images with large uniformly colored regions. Other advantages of binary over quaternary trees is the finer local control of multiresolution that can be attained, as well as the possibility of planar interpolation: it is, in general, impossible to find a plane comprising the four vertices of the rectangular subregions generated by quadtrees.

The paper is organized as follows. Section II explains the BTTC technique, the experimental results are discussed in Section III, and some concluding remarks are given in Section IV.

Paper approved by M. R. Civanlar, the Editor for Image Processing of the IEEE Communications Society. Manuscript received April 20, 1994.

R. Distasi is with the IRSIP/CNR, 80135 Napoli, Italy.

M. Nappi is with the Dipartimento di Informatica ed Applicazioni "R. M. Capocelli," Universitá di Salerno, 84081 Baronissi (SA), Italy.

S. Vitulano is with the Instituto di Medicina Interna "M. Aresu," Universitá di Cagliari, 09124 Cagliari, Italy.

Publisher Item Identifier S 0090-6778(97)06623-3.

II. The Coding Scheme

The image to be encoded can be regarded as a discrete surface, i.e., a finite set of points in three-dimensional (3-D) space, by considering a nonnegative discrete function of two discrete variables $F(x,y)$, and establishing a correspondence between the image and the surface $A = \{(x,y,c)|c = F(x,y)\}$, so that each point in A corresponds to a pixel in the image: the couple (x,y) gives the pixel's position in the XY plane, while c (the point's height) is the pixel's color. Fig. 1 shows the surface corresponding to the test image `lena`.

Our goal is to approximate A by a discrete surface $B = \{(x,y,d)|d = G(x,y)\}$, defined by means of a finite set of polyhedrons. Each polyhedron has a right-angled triangle (RAT) face on the XY plane and a RAT upper face approximating A. The surface B is made by the upper faces of the polyhedrons. In the following discussion, the term "PRAT" will denote a RAT on the XY plane, while "URAT" will stand for an upper face RAT included in B. Fig. 2 shows a region of the surface A being approximated by two URAT's.

To illustrate how this process works, let T be a generic PRAT of vertices

$$P_1 = (x_1, y_1), \quad P_2 = (x_2, y_2), \quad P_3 = (x_3, y_3), \tag{1}$$

and let

$$c_1 = F(x_1, y_1), \quad c_2 = F(x_2, y_2), \quad c_3 = F(x_3, y_3)$$

so that

$$(x_1, y_1, c_1), (x_2, y_2, c_2), (x_3, y_3, c_3) \in A. \tag{2}$$

The last three points define the URAT associated to T.

The restriction inside T of the approximating function G is given by the linear interpolation

$$G(x,y) = c_1 + \alpha(c_2 - c_1) + \beta(c_3 - c_1) \tag{3}$$

where α and β are defined by the two relations

$$\alpha = \frac{(x - x_1)(y_3 - y_1) - (y - y_1)(x_3 - x_1)}{(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)} \tag{4}$$

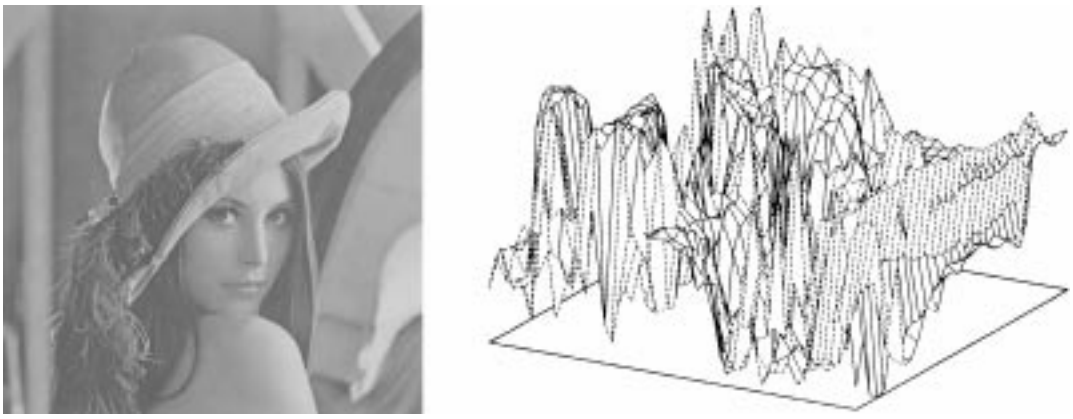
$$\beta = \frac{(x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1)}{(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)}. \tag{5}$$

From this definition, it can be seen that the values of F and G coincide on the vertices of T:

$$F(P_1) = G(P_1); \quad F(P_2) = G(P_2); \quad F(P_3) = G(P_3). \tag{6}$$

We now test our approximation function G by defining

$$\mathrm{err}(x,y) = |F(x,y) - G(x,y)|$$



(a) (b)

Fig. 1. Original image `lena` along with the corresponding surface.

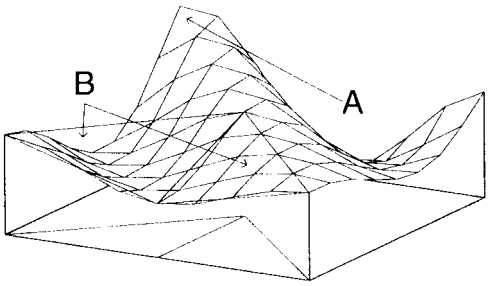


Fig. 2. Two URAT's in B approximate the surface A. The corresponding PRAT's are shown on the XY plane.

and checking for the condition

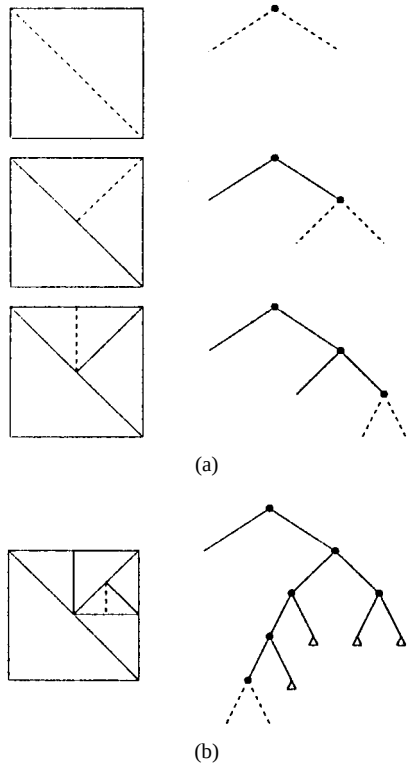
$$\mathrm{err}(x, y) \leq \epsilon \tag{7}$$

where $\epsilon > 0$ is an adjustable quality factor.

If the condition does not hold, the PRAT T is divided along its height relative to the hypotenuse. In this way, the resulting triangles are always right angled. If the subdivision procedure is reiterated indefinitely, we eventually obtain minimal PRAT's—comprising only three pixels, namely, their vertices—which surely satisfy condition (7) since the equalities in (6) ensure that $\mathrm{err}(x, y) = 0$ on each vertex.

The topological information relative to all subdivisions is stored in a hierarchical structure—a B tree; each PRAT is represented by a node whose position in the tree implicitly defines the vertices' coordinates. Fig. 3 shows how the partition process works: each time a PRAT is divided, the two resulting PRAT's become its children in the B-tree. As a consequence, each node has either zero or two children.

The pseudocode of BTTC is shown in Fig. 4. We assume without loss of generality that the image is square shaped and has side length m, where $m = 2^k+1$ for some integer $k \geq 1$; if this is not the case, the image is padded. With this assumption, all of the PRAT's generated by subsequent divisions will be isosceles, and their hypotenuse's length in pixels will still be of the form 2^h+1 for some $h \geq 1$. A RAT of vertices P_1, P_2, and P_3 is represented by the notation $\langle P_1 P_2 P_3 \rangle$. The first vertex corresponds to the right angle; the other two vertices follow clockwise. The X and Y coordinates vary from 1 to m. If the image has been padded, the PRAT's that contain no point of



(a)

(b)

Fig. 3. Example of the domain partition process. A leaf marked with a small triangle indicates that the corresponding PRAT satisfies the uniformity predicate. (a) The first three steps. (b) An intermediate step.

the original image rectangle are added to the list of leaves L with no processing.

The structure of the resulting B tree is stored in a binary string s obtained from a breadth-first visit of the tree: the value 0 represents a leaf node, the value 1 an intermediate node; the root is not stored at all. In this way, we need one bit for each node (with the exception of the root). Indeed, as illustrated in Fig. 5, it is possible to store only a part of s. In fact, if p is the minimum level at which a leaf appears, we can avoid storing all the leading 1's if we keep track of p. This expedient eliminates a substring—entirely composed of 1's—whose length is $\sum_{i=1}^{p-1} 2^i = 2^p - 2$.

Furthermore, if r is the B tree's maximum depth, we can dispense with storing all of the trailing 0's that represent leaves

BTTC Compression

1. [Initialize the set of "leaf" PRATs.]
   Set $L \leftarrow \emptyset$;
   [Initialize the first two PRATs.]
   Set $T_1 \leftarrow \langle (1,1)(1,m)(m,1) \rangle$, $T_2 \leftarrow \langle (m,m)(m,1)(1,m) \rangle$.
2. *Push* T_1 *and* T_2 *into the stack.*
3. *Pop the* PRAT T *from the stack;*
   Let $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, $P_3 = (x_3, y_3)$ *be its vertices;*
   [Determine the URAT corresponding to T.]
   Set $c_1 \leftarrow F(x_1, y_1)$, $c_2 \leftarrow F(x_2, y_2)$, $c_3 \leftarrow F(x_3, y_3)$;
4. [Test if the approximation for T is good enough.]
   For each point $(x, y) \in T$, *calculate* $G(x, y)$ *using (3), (4), and (5); If Condition (7) is satisfied for each point, then go to Step 6.*
5. [Divide T into two PRATs by drawing its height relative to the hypotenuse.]
   Set $P_M \leftarrow (P_2 + P_3)/2$;
   Set $T_1 \leftarrow \langle P_M P_1 P_2 \rangle$, $T_2 \leftarrow \langle P_M P_3 P_1 \rangle$;
   Go to Step 2.
6. *Insert* T *into* L.
7. *If the stack is empty then stop, otherwise go to Step 3.* ■

Fig. 4. B-tree triangular coding algorithm.

at level r, as the presence of these leaves can be inferred knowing which nonleaf nodes are at level $r-1$. In the case of a complete tree with $r+1$ levels numbered from 0 to r, we have that 2^r nodes out of $2^{r+1}-1$ can be spared—more than half of the total, but in the worst case, we only conserve the storage space for two leaves.

The heights of the URATs' vertices are memorized after the binary string s.

Decoding is analogous to encoding, with the only difference being that the quality of the approximation need not be tested. This yields a significant reduction of the operations required, also asymptotically.

The first step consists of reconstructing the B tree structure from the binary string s. To do so, the first two PRAT's (the biggest ones) are pushed into the stack, then s is scanned bit by bit. Whenever a 1 bit is encountered, the current PRAT is divided, and its two children are pushed into the stack; if a 0 bit is encountered, the current PRAT is added to the list of leaves L, and its vertex pixels in the reconstructed image are marked as "needing a height."

In the second step, the URAT's are generated from the information just gathered. First, all of the marked pixels in the reconstructed image receive their height (color); this defines F at the PRAT's vertices. Finally, (3) is computed inside each PRAT in L, thus reconstructing the surface B.

A. Computing Time

The BTTC method of compression has proven to be time efficient. For encoding, it requires a time proportional to an, where n is the number of pixels and a expresses the average number of subdivisions per pixel. An upper bound for a is therefore $O(\log n)$, yielding a worst case time $O(n \log n)$. The encoding process, having a high degree of inherent parallelism, would greatly benefit by a parallel implementation. A study in this direction is currently underway.

As for the decompression phase, the first and second steps are performed in $\Theta(t)$ time, where t is the number of nodes in the tree. The final step, requiring a calculation of (3)–(5) for each pixel, can be completed in $\Theta(n)$ time. Since $t < n$, the total time for decompression is $\Theta(n)$.

III. Experimental Results

In order to evaluate the performance of BTTC, several 8 bit test images have been used. This section will show some of the results obtained on one such image, 512×512 `lena`, depicted in Fig. 1.

The quality of the reconstructed images has been assessed by considering the bits per pixel (bpp), the root-mean-squared signal-to-noise ratio (RSNR), as well as the mean-squared error (MSE) and the peak signal-to-noise ratio (PSNR).

Since the BTTC algorithm ensures that $\text{err}(x, y) \leq \epsilon$, we obtain an upper bound on the MSE:

$$\text{MSE} \leq \epsilon^2. \tag{8}$$

From this inequality, we can deduce the obvious lower bound for the PSNR. This means, for example, that PSNR ≥ 24.04 dB for $\epsilon = 16$.

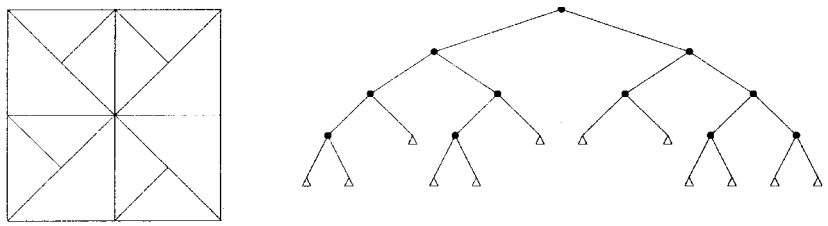
The visual results of the algorithm on `lena` compressed at different levels of quality are illustrated in Fig. 6, while Fig. 7 shows a sample outcome of domain partition.

Tables I and II show the statistics obtained for the images in Fig. 6. In Table I, the first column specifies the global bit rate of the compressed image, while the following two columns specify the size of the unabbreviated tree. The "heights" column tells how many values of F are present in the compressed data, while the following two columns give two separate bit rates, one for the B tree structure (the significant substring of s) and one for the heights. The latter undergo Huffman compression using a zigzag ordering similar to JPEG's, while the B tree structure is passed on unchanged. The last column reports the number of FLOP's (additions and multiplications/divisions) required for encoding by BTTC. In Table II, the parameter ϵ is related to the final quality of the compressed image.

From the analysis of the data in Table I, it can be seen that the number of leaves in the B trees is roughly half the total number of nodes. Furthermore, the number of heights that need to be stored is approximately equal to half the number of leaves. Finally, the bit rate needed for the heights is about twice the bit rate for the B tree structure.

A comparison with the standard compression techniques is in order. In particular, JPEG's bit rates are much better than BTTC's for a fixed SNR. However, for the same SNR, the subjective quality of BTTC-encoded images is better because the scheme proposed is sensitive to the local features of the image, thus yielding fewer distracting artifacts. Fig. 8 compares JPEG and BTTC for PSNR values around 37 dB. From the image, it can be seen that the eye details are clearer with BTTC coding, and JPEG induces a more visible blockiness on the shoulder and the background.

As happens with JPEG, the multiresolution scheme utilized by BTTC can be exploited to allow for progressive decoding.



111111∗10100011∗00000000

Fig. 5. Possible domain partition with its B tree and the resulting binary string s. Only the substring between asterisks has to be stored.

Fig. 6. Image `lena` compressed at different bit rates. From top left: 1.79, 1.20, 1.00, 0.68, 0.45, 0.35 bpp.

As can be seen in Table II, JPEG offers significantly better quality than BTTC. However, there is a price to be paid in terms of efficiency: the number of operations performed by JPEG during encoding is independent of the bit rate. In addition, the same number of operations is required for decoding. In contrast, BTTC'encoding has a smaller computing load that

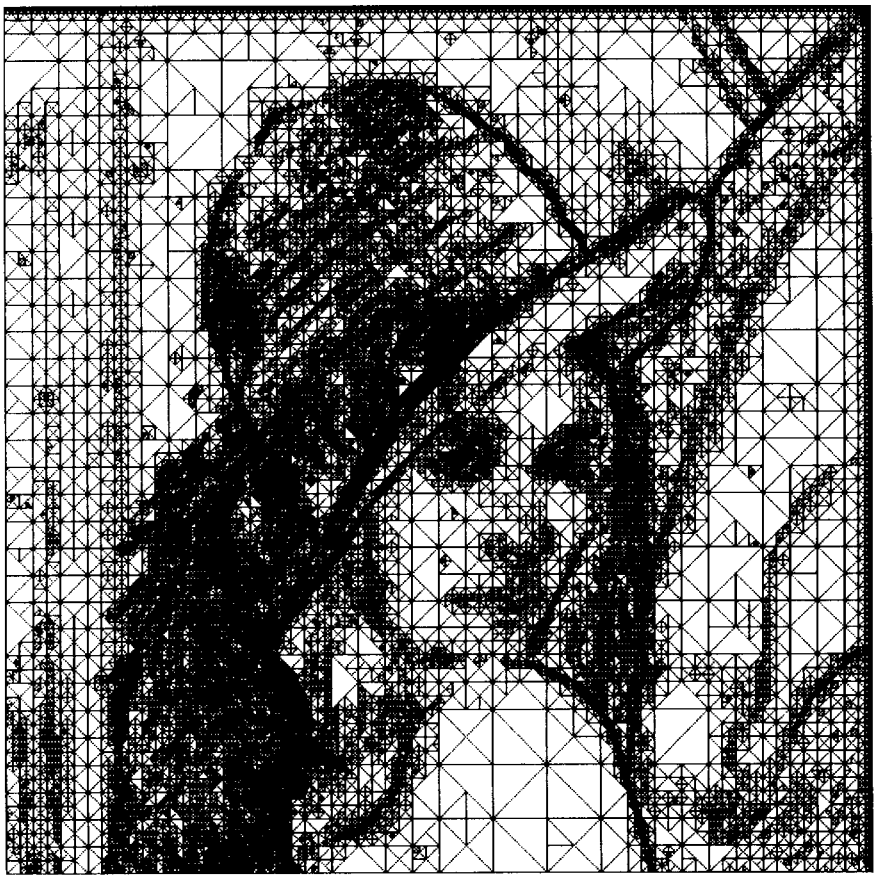


Fig. 7. Domain triangulation for `lena` at 1.20 bpp ($\epsilon = 12$).

(a) (b)

Fig. 8. Image `lena`: encoded (a) with JPEG at 0.68 bpp, 37.14 dB; (b) with BTTC at 1.43 bpp, 36.96 dB,

grows with the bit rate (see Table I), while decoding requires only 1.8 million FLOP's. For comparison purposes, JPEG requires about 5 million FLOP's (3 million additions and 2 million multiplications) for encoding or decoding a 512 $\times$ 512 $\times$ 8 image [4].

IV. Conclusions

This paper has presented a new lossy algorithm for image compression. The algorithm is based on recursive domain triangulation and planar linear interpolation between the vertices of the spatial triangles. The asymptotic time complexity is $O(n \log n)$ for coding and $\Theta(n)$ for decoding. It is simple to implement and highly parallel.

Experimental tests show that the quality achieved is still acceptable for bit rates around 0.5 bpp. Moreover, BTTC outperforms JPEG with regard to execution time, although the bit rates are higher by a factor of about 2.

Current research on BTTC is concentrating on the following topics: a parallel implementation, which is a natural idea given the inherent parallelism of the compression phase; a more

TABLE I
COMPRESSION STATISTICS FOR IMAGE lena

bpp	Total nodes	Leaves	Heights	bpp (tree)	bpp (heights)	FLOPs ($\times 10^6$)
0.35	35703	17853	9191	0.13	0.22	4.16
0.45	45273	22638	11688	0.16	0.29	4.24
0.68	67317	33660	17358	0.23	0.45	4.42
0.88	88105	44054	22757	0.30	0.58	4.57
1.00	103043	51523	26568	0.34	0.66	4.65
1.20	122829	61416	31641	0.40	0.80	4.82
1.43	149447	74725	38460	0.48	0.95	5.01
1.79	190793	95398	49010	0.60	1.19	5.45

TABLE II
DISTORTION MEASURES FOR lena

bpp	ϵ	PSNR (BTTC)	RSNR (BTTC)	MSE (BTTC)	PSNR (JPEG)
0.35	26.0	29.51	23.57	72.71	34.03
0.45	24.0	30.42	24.28	58.99	35.36
0.68	20.0	32.30	26.36	38.29	37.14
0.88	16.0	33.82	27.88	27.00	38.42
1.00	14.0	34.77	28.82	21.70	38.91
1.20	12.0	35.79	29.85	17.13	39.44
1.43	10.0	36.96	30.97	13.10	39.98
1.79	8.2	38.46	32.52	9.26	42.22

efficient coding of the B tree structure; the use of nonlinear approximation; and a scheme to obtain adaptive modifications of the parameter ϵ.

REFERENCES

[1] G. K. Wallace, "The JPEG still picture compression standard," *Commun. ACM*, vol. 34, Apr. 1991.

[2] A. Wright, "Image compression: The DCT approach," *Image Processing*, Winter 1990.

[3] B. A. D. Vore, B. Jawerth, and B. J. Lucien, "Image compression through wavelet transform coding," *IEEE Trans. Inform. Theory*, vol. 38, pp. 719–746, Mar. 1992.

[4] L. J. Dimento and S. Y. Berkovich, "The compression effects of the binary tree overlapping method on digital imagery," *IEEE Trans. Commun.*, vol. 38, Aug. 1990.

[5] A. E. Jacquin, "Image coding based on a fractal theory of iterated contractive image transformations," *IEEE Trans. Image Processing*, vol. 1, pp. 18–30, Jan. 1992.

[6] N. M. Nasrabadi, C. Y. Choo, and Y. Feng, "Dynamic finite-state vector quantization of digital images," *IEEE Trans. Commun.*, vol. 42, pp. 2145–2154, May 1994.

[7] B. Zeng and Y. Neuvo, "Interpolative BTC image coding with vector quantization," *IEEE Trans. Commun.*, vol. 41, pp. 1436–1438, Oct. 1993.

[8] R. Szelinski, "Fast surface interpolation using hierarchical basis functions," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, pp. 513–528, June 1990.

[9] H. Samet, "The quadtree and related hierarchical structures," *ACM Comput. Surv.*, 1984.

[10] M. Tamminen, "Encoding pixel trees," *Computer Vision, Graphics, Image Processing*, vol. 28, pp. 44–57, 1984.

Riccardo Distasi was born in Naples, Italy, in 1966. He received the laurea degree in computer science from the University of Salerno in 1992 and has been a CNR (Italian National Research Council) grantee since 1994.

Presently, he is assistant professor of Computer Science at the Second University of Naples. His research interests include signal processing and parallel computing. He is a member of IAPR since 1996.

Michele Nappi was born in Castellammare di Stabia, Naples, Italy, in 1965. He received the laurea degree in computer science from the University of Salerno in 1991 and is currently working toward the the M.Sc. degree in information and communication technology and Ph.D. degree in applied mathematics and computer science at the University of Naples "Federico II."

His research interests include pattern recognition, image processing and indexing.

Mr. Nappi is a member of IAPR since 1991.

Sergio Vitulano was born in Milan in 1940. He graduated in nuclear physics at the University of Naples "Federico II" in 1974 and received the Ph.D. in Cybernetics from the University of Salerno in 1976, where he taught image processing until 1992.

Presently, he is Associate Professor of Computer Science at the University of Cagliari, he is interested in image processing, pattern recognition and medical imaging. He is a member of IAPR and IASTED.