アルゴリズム 12－17
(1989. 11. 21)

Error-Free Image Compression with Gray Scale Quadtrees and its Optimization

Martin J. Dürst and Tosiyasu L. Kunii

Department of Information Science, Faculty of Science
The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, 113 Tokyo, Japan

Abstract

GDF, the Gray scale Depth First expression, is a new method for the error-free compression of gray scale images. It combines spatial hierarchical subdivision (quadtrees) with gray scale hierarchical subdivision, and achieves better compression than any previous hierarchical method.

GDF is not limited to quadtrees. The structure of both the spatial and the gray scale hierarchy are not restricted in any way. This leads to an algorithm that finds the optimal gray scale hierarchy for any given picture in polynominal time, resulting in significant additional compression.

1 Introduction

In image processing and computer vision, image compression is an important application and a basic tool for other, higher level tasks. As a tool, compression can reduce the amount of data to be processed at higher levels like enhancement, matching, recognition, etc. As an independent application, it is very useful in reducing bandwidth on communication channels and storage requirements for mass storage.

Image compression methods can basically be divided into two categories, approximate compression and error-free compression. Approximate compression reduces the amount of data as much as possible with a tolerable quality reduction. Error-free compression encodes the image so that it can be reproduced exactly. Exact reproduction is very important for image databases and fields such as law, and medicine.

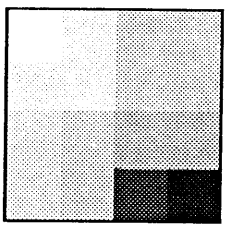
Error-free compression rates are not very high. There is no general type of image, and so each compression method is appropriate for some class of images, but hardly compresses or even expands others. Also, the less significant bit planes of most images contain a large amount of random noise, which is difficult to compress in any way. However, for large archives even a small improvement can result in large savings in space and money.

In this paper we present a new way of gray scale image compression based on quadtrees. Section 2 gives a short overview on quadtrees, focusing on gray scale quadtrees. In section 3, we introduce the bitwise condensed quadtree (BCQ) and the gray scale depth first expression (GDF), and give an outline of the compression and decompression algorithms. Section 4 shows how the gray scale hierarchy can be optimized to improve compression. Finally, section 5 contains experimental results.

2 Quadtrees for Gray Scale Images

An image of resolution r, size $2^r \cdot 2^r$, and b bits per pixel can be represented by a quadtree [Sam84] by recursively subdividing the image square into four subquadrants, thereby building a tree of outdegree four where the root represents the whole image and the leaves represent single pixels. This tree is called full quadtree. If all the pixels in one quadrant have the same value, the quadtree is condensed, replacing each interior node whose children are all of the same value by a leaf node.

For the image in fig. 1, with a resolution of r=2 and b=3 bits per pixel, the (condensed) quadtree is shown in fig. 2. Quadrants are ordered from left to right and top down.



a)

000	001	011	011
001	001	011	011
010	011	100	100
010	011	101	110

b)

Figure 1. a) Gray scale image
b) Corresponding binary array.

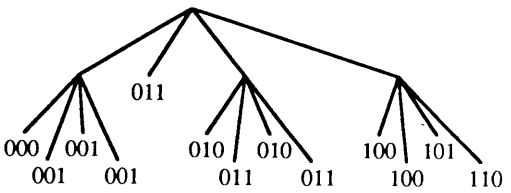


Figure 2. Quadtree for the image of fig. 1.

There exist many ways to represent a quadtree [Sam84]. The most compact of them (cf. [Tam84a]) is the DF-expression (Depth First-expression, hereafter simply called DF) [Kaw80], a sequence of symbols resulting from a preorder depth first traversal of the quadtree.

Some of the early works on quadtrees were mainly concerned with gray scale images (e.g. [Kli76]). However, for gray scale images, it is rare that even four neighboring pixels all have the same color. Condensation is much higher for binary quadtrees, and thus the research on quadtrees concentrated on this field.

There were several attempts to solve this problem. Klinger et al. [Kli76] tried to allow condensation based on statistical attributes like the standard deviation. Oliver et al. [Oli83] placed average values in interior nodes. Kawaguchi et al. [Kaw83], after converting pixel values to a Gray code, coded each bit plane separately. Based on bintrees [Kno80], Tamminen [Tam84b] developed a coding suited for so called 'maps', images that consist of rather large areas of unique color, but where colors of adjacent areas are not related. Woodwark [Woo84] proposed a similar scheme based on quadtrees. First proposed in [Kaw80,p.30-31], Oliver et al. [Oli83] and Kunii et al. [Kun86] presented slightly different variants of an extension of DF to gray scale images. In [Kun86], a family of quadtrees called G-quadtrees was also introduced. Each G-quadtree uses a different number of most significant bits (MSB), changing the amount of condensation based on the needs of the application. This work was extended to 3D and refined in [Mao87].

3 Gray Scale Depth First Expression (GDF)

3.1 Definition

As mentioned earlier, for a gray scale image it is rare that even in a small square of four pixels all pixels have the same color. However, for a wide range of images, adjacent pixels mostly have values close to each other. Seen at a low gray scale resolution, an image seems to consist of large areas of the same color. As the gray scale resolution is increased, we have to increase the spatial resolution, too, although for each (part of an) image the relation between gray scale resolution and spatial resolution is different.

To integrate gray scale hierarchical subdivision and spatial hierarchical subdivision for image compression, we introduce a new form of quadtree, called bitwise condensed quadtree (BCQ). Starting from the MSB, whenever all the first bits of all the children of a node are the same, these bits are removed and a corresponding bit added to the end of the entry of this node. When we remove the last bit from a leaf, we of course also remove the leaf itself. For our example, the result is depicted in fig. 3. Parentheses denote the end of an interior node, and indices the bit position of the corresponding symbol.

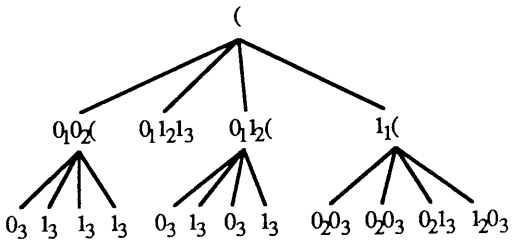


Figure 3. BCQ for the image of fig 1.

Now we can define GDF as the sequence of the three symbols **0**, **1**, and **(**, formed on a preorder depth first traversal of the BCQ, where bits 0 and 1 are represented by the symbols **0** and **1**, and a **(** is added after the last bit of each interior node. The following is GDF for our example, with blanks added for legibility:

(00(0 1 1 1 011 01(0 1 0 1 1(00 00 01 10

3.2 Proof of Error-free Decodability

We use the BCQ to prove that an image encoded with GDF can be decoded without errors. First, we will show how to reconstruct the BCQ from GDF, and then how to obtain the image from the BCQ. The key to decoding GDF is that every **0** or **1** can be indexed by its bit position, as shown in fig. 3.

We note that every entry in a leaf node has to end with $\mathbf{0}_b$ or $\mathbf{1}_b$, where b is the number of bits per pixel, and every entry of an interior node ends with **(**, and these are the only places where $\mathbf{0}_b$, $\mathbf{1}_b$, or **(** can appear. Also, when going from the root of the tree to any of its leaves, the bit position indices of the symbols are passed in sequence, from 1 to b.

Thus we can always assign the correct indices to the symbols being decoded, as we only need the information of the already decoded parent nodes. Also, we are always able to detect the last symbol of a node and decide on the next node that has to be processed. We can therefore reconstruct the tree exactly, and also detect the end of the code string.

To reconstruct the original image from the BCQ, we propagate the entries from each interior node to all its children, concatenating bits and eliminating parentheses, until we get the complete quadtree. As bits have only been condensed when they are the same for all children, the original uncondensed quadtree is reproduced exactly. Each leaf corresponds to a pixel of the original image, and so the original image can be reconstructed completely and without errors from GDF.

3.3 Binary Coding of the Three Symbols

Kawaguchi et al. [Kaw80, p.32] propose several ways for the binary coding of the three symbols **0**, **1**, and **(**. On the lowest level of spatial subdivision, there are no **(**, and so **0** and **1** are each coded with one bit. On higher levels, however, it is not clear which of the three symbols is the most frequent and therefore should have a one-bit code. Contrary to Kawaguchi et al. but as Tamminen [Tam84b], we code **(** with one bit, preserving the symmetry between **0** and **1**. This makes the maximal length of a 'bad' image the shortest, namely $2^{2r}\cdot(b+1/3)$ bits for an image of $2^{2r}\cdot b$ bits. An alternative is arithmetic coding [Wit87]. A coding model would take into account the probabilities of different symbols at different levels of spatial and gray scale subdivision.

Adapting 10 (or 0 on the lowest level) for **0**, 11 (or 1) for **1**, and 0 for **(**, and coding the GDF of our example image gives the following binary sequence. For comparison, we also give GDF again:

(0 0 (01110 1 1 0 1 (0101 1 (00000110
0101000111101111101100101111000000110
Over all, for this example we used 36 bits instead of 48, a saving of 25%.

3.4 Decoding Algorithm

We explain the decoding algorithm for GDF before the coding algorithm because it is much simpler. It basically follows the proof in section 3.3, but combines the analysis of the input and the expansion of the tree, without building the BCQ.

The algorithm is easier to formulate with a recursive procedure, but more efficient when implemented with a stack. In both cases, each level of the recursion or the stack corresponds to one level in the spatial hierarchy. Each level keeps track of the position of the upper left corner and the size of the square corresponding to the present node, the number of the subquadrant presently processed, and the level in the gray scale hierarchy (=the number of significant bits).

For each node of the quadtree, in depth first sequence, symbols are read until a parenthesis is received or the number of significant bits reaches b, and then the four subnodes are processed in turn. For a node at the pixel level, the accumulated bits are written to the corresponding position of the frame buffer. For a full recursive formulation of the algorithm and an outline of a hardware implementation , the reader is referred to [Dür88].

3.5 Encoding Algorithm

Encoding an image to GDF is more complicated than decoding, and cannot be compared directly with the encoding of DF. The very first symbol of GDF depends on the MSB of all pixels of the image. If all of them are the same, the first symbol is **0** or **1**, otherwise, it is **(**. So in principle, we have to scan the whole image before we can output anything (compare, however, section 4.4).

This makes it necessary to use two passes, working bottom up in the first pass to decide on the number of significant bits for each node, and top down in the second pass to output the symbols. To store the number of significant bits in all the levels but the lowest one, a data structure similar to the explicit quadtree of [Woo82] is best used. The number of significant bits in the lowest level is always b. Otherwise, the comments of section 3.4 apply here, too.

4 Optimizing the Gray Scale Hierarchy

4.1 Basics

For the sake of simplicity, we presented GDF based on quadtrees. However, it can easily be adapted to trees with outdegrees other than four. Of particular interest for image processing are bintrees [Kno80] and octrees [Sam88]. In general, any tree structure is possible, as long as the encoder and the decoder agree on it. The same applies to the gray scale hierarchy. Also it is possible to deviate from a pure depth first traversal. This variations and their various applications such as progressive transmission and a fast hardware encoder, are outlined in [Dür88]. Here we concentrate on increasing compression by optimizing the gray scale hierarchy.

We present an algorithm that, with respect to a given image and a given binary coding of the symbols of GDF, optimizes the gray scale hierarchy. It calculates, in time $O(2^{2r}+g^3)$, the gray scale hierarchy that leads to the highest compression. 2^{2r} is the number of pixels in the image, and $g=2^b$ is the number of gray levels. The example image we will use, and its unoptimized gray scale hierarchy, are shown in fig. 4.

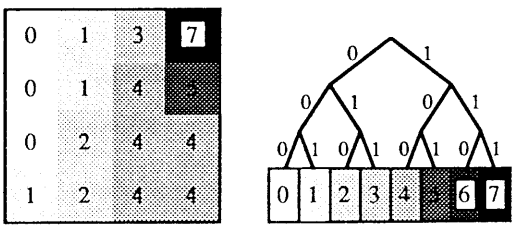


Figure 4. Example image for optimization (a) and its gray scale hierarchy (b).

At first, it may seem difficult to find an efficient algorithm for the problem at hand because of the complicated relationships between the spatial and the gray scale hierarchy. The key idea is to consider all possible gray scale intervals. First, the quadtree is traversed and for each node, the gray scale interval it covers is calculated, and the number of nodes of each interval is counted. Next, nonexistent gray levels are eliminated and the number of independent trees for

each interval is calculated. The actual optimization then decides the best splitting point for each interval working bottom-up.

In the next two subsections, the formal base of the algorithm is presented, along with the actual values for the example image of fig. 4. Due to space constraints, there is no program code, and almost no comments for the example, but we trust that the readers can work this out for themselves.

4.2 Interval Statistics

To formalize the algorithm, we first note that the number of parentheses is independent of the gray scale coding and therefore parentheses can be ignored in the optimization algorithm. Then we denote by L_i the number of leaves of the (full) quadtree with gray level i, and by N_i^j $(i \le j)$ the number of nodes whose childrens' smallest gray level is i and whose largest gray level is j. N can be represented as a triangular matrix.

Inspecting L, gray levels that are not used at all can be eliminated in time $O(g)$. As the number of actually used gray levels is bounded by g, we will continue to use g for simplicity.

Now we can calculate the number T_i^j of independent trees in the quadtree that contain leaves in the interval form i to j. Each leaf in such a tree can be considered originally as an independent tree, and each internal node connects four subtrees to one tree and thus reduces the number of trees by three. Therefore,

$$T_i^j = \sum_{k=i}^{j} L_k \; - \; 3 \cdot \sum_{k=i}^{j} \sum_{l=k}^{j} N_l^k \qquad (1).$$

T_i^j is also the number of root nodes of independent trees with the corresponding interval and thus the number of nodes with that interval whose parent does not fit in that interval.

Actually, L and N can be integrated from the start into N^* so that

$$N^{*j}_i = L_i \; - 3 \cdot N_i^j \qquad \text{and}$$

$$N^{*j}_i = - 3 \cdot N_i^j \quad (i<j) \qquad (2) \text{ and so}$$

$$T_i^j = \sum_{k=i}^{j} \sum_{l=k}^{j} N^{*k}_l \qquad (3).$$

N^* needs $O(g^2)$ storage and can be obtained in a simple first-depth traversal of the quadtree using $O(2^{2r})$ time and $O(r)$ storage. For each quadtree node, the corresponding N^{*j}_i is incremented by 1 for leaf nodes and by -3 for internal nodes. These values can be adjusted for trees or nodes with outdegrees other than four. T can be calculated from N^* in time $O(g^2)$. New space is not needed as the values of T can overwrite those of N^*. Fig. 5 shows the values of N^* and T for the example image of fig. 4, together with some values introduced in the next subsection. There is no pixel with gray value 6, and so it has been eliminated.

i \ j	0	1	2	3	4	5	7
0	3 3 - 3	-3 3 0 9	-3 2 1 13	0 3 2 17	0 5 2* 24	0 6 3 29	0 1 2 (25)
1		3 3 - 13	0 5 1 10	0 6 1 15	0 8 1* 24	0 9 1* 30	0 7 1 28
2			2 2 - 2	0 3 2 6	0 5 2* 13	0 6 3 18	0 4 2 18
3				1 1 - 1	0 3 3 6	0 4 3* 13	-3 2 4 12
4		N* T D R			2 2 - 2	0 3 4 6	0 4 4 10
5						1 1 - 1	0 2 5 4
7							1 1 - 1

Figure 5. Values of N^*, T, D, and R.

4.3 Finding the Optimum

The number of symbols (excluding parentheses) needed to code all trees in a given interval, S_i^j, can now be expressed as follows. For an interval with only one gray level, there is no information necessary to distinguish this gray level from others, and so

$$S_i^i = 0 \qquad (4).$$

For larger intervals, S_i^j depends on the division point D_i^j $(i \le D_i^j < j)$ of the interval, defined as the top gray value of the lower subinterval. Setting $k = D_i^j$, S_i^j can be calculated as

$$S_i^j = S_i^k + S_{k+1}^j + T_i^k + T_{k+1}^j \qquad (5)$$

where the first two terms represent the number of symbols to code all the trees of the two subintervals independently, and the later two terms correspond to the number of symbols, one for each independent tree of the subintervals, to code the subinterval the tree belongs to.

Optimization is carried out by selecting D_i^j so that S_i^j is minimized. The optimum values of S_i^j are calculated for small intervals first, so that they are available when deciding the splitting values of larger intervals. The number of symbols needed for the whole tree is obviously S_0^{g-1}. The optimal gray level hierarchy can be constructed from a subset of D, starting with D_0^{g-1}. The number of additions in (5) can be reduced by introducing

$R_i^j = S_i^j + T_i^j$ so that

$$S_i^j = R_i^k + R_{k+1}^j \qquad (6).$$

Again, the R values can overwrite the T values, but separate space is needed for D. The time for this step is dominated by the number of the evaluations of (6), which is of the form $\sum_{k=1}^{g} k\cdot(g-k)$ and thus $O(g^3)$, so that the whole algorithm optimizing the gray scale hierarchy uses time $O(2^{2r}+g^3)$. D and R are shown in fig. 5. A star means that there are other division values that lead to the same optimal coding for the interval. The value in parentheses is S_0^7, the number of symbols (without parentheses) needed for the optimal coding of the example image, and not R_0^7, which has no meaning. The final gray scale hierarchy is shown in fig. 6.

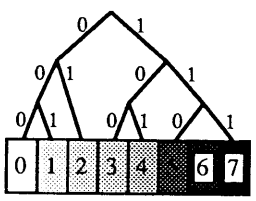


Figure 6. Optimal gray scale hierarchy.

Below are the GDF based on conventional (upper row) and optimal (lower row) gray scale hierarchy. For this example, we saved five symbols. As we also have to transmit the gray levels not used (g bits) and the structure of the optimized gray scale hierarchy ($<2g$ bits), optimization may not pay of here. As we will show in section 5, this is different for actual images.

(00(0101 (011 111 100 101 0(00 10 01 10 100
(00(0101 1(00 11 01 10 0(00 1 01 1 101

4.4 Considering Binary Coding

Above, only the number of symbols was considered, regardless of the fact that a symbol, depending on its position, may be coded with one or two bits (see section 3.3). The algorithm can be modified to take into account the number of bits per symbol as follows. First, imagine that separate statistics are collected for nodes at each level in the spatial hierarchy, and that (5) is extended, summing over the levels, each level weighted appropriately. The weighting factors can then be propagated to the first step similar to (2). The resulting increments for N^* become 1 for pixel nodes, -2 for the next higher level, and -6 for all other nodes. In this way, the later steps of the algorithm, and thus its complexity, remain the same.

The gray level hierarchy is not limited to a binary tree. Of particular interest are ternary trees because they permit the four symbols **0**, **1**, **2**, and **(** to be coded with exactly two bits. Finding the optimal gray level hierarchy in this case is still possible in time $O(g^3)$. First, D is calculated for the binary subdivision. When deciding on the division of an interval into three subintervals, extensive search is necessary only for one division point. The optimal second division point is identical to the binary division point for the remaining interval, as there are just two terms more in (5) or one more in (6).

Generalizing this, it is possible to find the optimal n-ary gray scale coding with $O(g^3 \cdot \log n)$ time and $O(g^2 \cdot \log n)$ space, and all gray scale codes up to n-ary with $O(g^3 \cdot n)$ time and $O(g^2 \cdot n)$ space. It is also possible to combine binary (for the lowest level of the spatial hierarchy) and ternary (for the upper levels) trees. We have to leave the details of this to the reader.

A fractional number of bits for each symbol, as in the case of arithmetic coding, is also possible. However, the algorithm is limited in the sense that each binary coding has to be optimized separately, and all symbols except **(** have to be coded with the same number of bits.

4.5 Related Work

Optimizing the gray level hierarchy can be compared to the normalized or optimal quadtree of [Li82] and [Gro83] on the spatial side. However, there only the *position* of the quadtree grid is optimized, which will not give substantial savings if there are no big rectangular blocks, whereas we are optimizing the *structure* of the gray scale hierarchy.

Trying to optimize the structure of the spatial hierarchy will lead to something like the k-d-tree [Ben75]. But for usual images, the resolution r and the number of bits per pixel b are of about the same order, and so the gray scale hierarchy ($g = 2^b$ leaves) is much smaller than the spatial hierarchy (2^{2r} leaves). Thus to improve compression, changes to the

gray scale hierarchy are more promising than changes to the spatial hierarchy.

The optimization algorithm presented above also can lead to a new approach to segmentation. The first bit of the optimized gray scale divides the image into two parts, a lighter and a darker one, in such a way that the dividing line between the two parts is short and the the two parts themselves are as homogeneous as possible. However, there is still much work to be done before we will know for what kind of applications this approach is successful.

5 Experiments

To evaluate the compression ratio of GDF, and the improvement obtained by the gray scale optimization, we compared it with several other gray scale image compression algorithms. The programs were implemented in the programming language C on a VAX-750™ under Unix™. The other methods tested were the bitwise DF of [Kaw83], LZW [Wel84], and predictive Huffman coding [Ros82,pp.181-188].

LZW was used as implemented in the UNIX™ commands *compress* and *uncompress*. Pixels were reordered according to the Morton sequence to satisfy the requirements of [Lem86,p.8].

For comparability, the approach of Kawaguchi et al. [Kaw83], which transforms pixels to a Gray code and then uses DF for every bit plane separately, was modified to allow error-free compression.

Predictive Huffman coding was implemented in its simplest form [Ros82,p.182,(172)]. Whereas the other methods are based on the principle of spatial hierarchical subdivision, this method is scan line oriented and serves as a reference point.

The results of the experiments are shown in table 1. The images used are taken from version 1 of the Standard Image Data Base (SIDBA) [Ono79], which includes many well known images. Therefore, we refrain from including the images in this paper. For the girl and the couple, see e.g. [Hun79], for the moon surface, [Ros82,p.160], and for the aerial, [Lee80,fig.1.a]. In table 1, Nº refers to SIDBA version 1, and GDF, Gopt, DF, LZW, and Huff denote the algorithms as explained above. For all images, r=8 and b=8. Compression is given as

$$\frac{\text{size of compressed image}}{\text{size of full image}} \cdot 100\%,$$

so that smaller values indicate better compression and values above 100% indicate that the image is expanded rather than compressed. In the case of Gopt, the size of the compressed image includes the data necessary to transmit the gray scale hierarchy.

Nº	Name	GDF	Gopt	DF	LZW	Huff
1	Girl	73.3	59.5	77.2	76.7	64.0
5	Couple	64.8	57.2	68.5	74.3	61.6
10	Moon	82.0	71.8	83.1	94.1	69.9
11	Aerial	88.6	82.3	96.6	106.7	77.8

Table 1. Compression rate (units: %).

From table 1, we see that GDF performs clearly better than previous methods based on spatial hierarchical subdivision, and the gray scale optimization considerably improves GDF. This is especially obvious if we compare saved space instead of compression ratios. The other quadtree based methods mentioned in section 2 are either not suited for continuous tone images or clearly less efficient in any case.

Predictive Huffman coding, though in its simplest version, performs more or less equal to the optimized GDF. However, as a scan line oriented method, it is not usable in cases such as progressive transmission. Its better performance is mainly due to its efficiency for textures, while GDF is oriented towards smooth images. LZW is also suited for textures, but is inefficient because every gray value is treated independently.

For all the methods, fine tuning of the parameters and changing from algorithms working with bits as their smallest units to arithmetic coding [Wit87] could give better results, but it is doubtful whether this would change the overall ranking.

6 Conclusions

In this paper, we presented GDF, the Gray scale Depth First expression, a new way of error-free image compression combining the concepts of spatial and gray scale hierarchical subdivision. An algorithm for the optimization of the gray scale hierarchy further improved the compression rate and proved the validity of our approach of combining both hierarchies. For future research, we intend to concentrate on applications, like progressive transmission, segmentation, and implementation in hardware.

Acknowledgements

We would like to thank Prof. Satoru Kawai of the College of Arts and Sciences of the University of Tokyo, Issei Fujishiro of the University of Tsukuba, and Xiaoyang Mao and Yoshihisa Shinagawa of the Kunii Laboratory for interesting discussions and comments, and Prof. Masao Sakauchi of the Multidimensional Image Processing Center, Institute of Industrial Science, University of Tokyo, for providing the image data for our experiments (cf. [Ono79]).

References

[Ben75] Bentley, J.L. Multidimensional Binary Search Trees Used for Associative Searching. Comm. ACM 18, 9 (Sept. 1975), 509-517.

[Dür88] Dürst, M.J., and Kunii, T.L. Error-Free Image Compression using Gray Scale Quadtrees. Technical Report 88-024, Department of Information Science, Faculty of Science, University of Tokyo, Dec. 1988.

[Gro83] Grosky, W.I., and Jain, R. Optimal Quadtrees for Image Segments. IEEE PAMI 5, 1 (Jan. 1983), 77-83.

[Hun79] Hung, St.H. A Generalization of DPCM for Digital Image Compression. IEEE PAMI 1, 1 (Jan. 1979), 100-109.

[Kaw80] Kawaguchi, E., and Endo, T. On a Method of Binary-Picture Representation and Its Application to Data Compression. IEEE PAMI 2, 1 (Jan. 1980), 27-35.

[Kaw83] Kawaguchi, E., Endo, T., and Matsunaga, J. Depth-first expression viewed from digital picture processing. IEEE PAMI 5, 4 (July 1983), 373-384.

[Kli76] Klinger, A., and Dyer, C.R. Experiments on picture representation using regular decomposition. Computer Graphics and Image Processing 5 (1976), 68-105.

[Kno80] Knowlton, K. Progressive Transmission of Grey-Scale and Binary Pictures by Simple, Efficient, and Lossless Encoding Schemes. Proc. IEEE 68, 7 (July 1980), 885-895.

[Kun86] Kunii, T.L., Fujishiro, I., and Mao, X. G-quadtree: A Hierarchical Representation of Gray-Scale Digital Images. The Visual Computer 2, 4 (Aug. 1986), 219-226.

[Lee80] Lee, J.-S. Digital Image Enhancement and Noise Filtering by Use of Local Statistics. IEEE PAMI 2, 2 (March 1980), 165-168.

[Lem86] Lempel, A., and Ziv, Jacob. Compression of Two-Dimensional Data. IEEE Trans. on Information Theory 32, 1 (Jan. 1986), 2-8.

[Li82] Li, M., Grosky, W.I., and Jain, R. Normalized Quadtrees with Respect to Translations. Computer Graphics and Image Processing 20, 1 (Sept. 1982), 72-81.

[Mao87] Mao, X., Kunii, T.L., Fujishiro, I., and Noma, T. Hierarchical Representations of 2D/3D Gray-Scale Images and Their 2D/3D Two-Way Conversion. IEEE CG&A 7, 12 (Dec. 1987), 37-44.

[Oli83] Oliver, M.A., and Wiseman, N.E. Operations on Quadtree Encoded Images. The Computer Journal 26, 1 (Jan. 1983), 83-91.

[Ono79] Onoe, M., Sakauchi, M., and Inamoto, Y. SIDBA Standard Image Data Base. MIPC Report 79-1, Multidimensional Image Processing Center, Institute of Industrial Science, University of Tokyo (March 1979).

[Ros82] Rosenfeld, A., and Kak, A.C. Digital Picture Processing, Second Edition, Volume 1, Academic Press, New York, 1982.

[Sam84] Samet, H. The Quadtree and Related Hierarchical Data Structures. ACM Computing Surveys 16, 2 (June 1984), 187-260.

[Sam88] Samet, H. and Webber, R.E. Hierarchical Data Structures and Algorithms for Computer Graphics, Part I: Fundamentals, and Part II: Applications. IEEE CG&A 8, 3 (May 1988), 48-68, and 4 (July 1988), 59-75.

[Tam84a] Tamminen, M. Comment on Quad- and Octtrees. Comm. ACM 27, 3 (March 1984), 248-249.

[Tam84b] Tamminen, M. Encoding Pixel Trees. Computer Vision, Graphics, and Image Processing 28 (1984), 44-57.

[Wel84] Welch, T.A. A Technique for High-Performance Data Compression. IEEE Computer 17, 6 (June 1984), 8-19.

[Wit87] Witten, I.H., Neal, R.M., and Cleary, J.G. Arithmetic Coding for Data Compression. Comm. ACM 30, 6 (June 1987), 520-540.

[Woo82] Woodwark, J.R. The Explicit Quad Tree as a Structure for Computer Graphics. The Computer Journal 25, 2 (March 1982), 235-238.

[Woo84] Woodwark, J.R. Compressed Quad Trees. The Computer Journal 27, 3 (May 1984), 225-229.