GRAPHICAL MODELS AND IMAGE PROCESSING
Vol. 59, No. 2, March, pp. 61–72, 1997
ARTICLE NO. IP960417

Use of Neural Networks to Estimate the Number of Nodes of an Edge Quadtree*

F. A. SCHREIBER AND R. CALVO WOLFLER†

Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci, 32, 20133 Milan, Italy

Received May 16, 1995; revised December 2, 1996; accepted December 19, 1996

The number of nodes of an edge quadtree is the measure of its space complexity. This number depends on the figure's shape, its resolution and its precision. The goal of this work is to prove that a relation exists between the number of nodes of an edge-quadtree and these three parameters. To reach this goal an experimental approach has been used. A unique value to represent both the resolution and the precision is used. To measure the shape of the image we use the fractal dimension. A methodology to calculate the fractal dimension and the fractal measure is proposed. These three parameters being given, we use a neural network to approximate the sought function. The computational results show the effectiveness of this approach.

1. INTRODUCTION

Quadtrees are well known hierarchical data structures, based on a regular and recursive decomposition of space, used to represent spatial data. This methodology reduces storage by aggregating data having identical or similar values and it improves data manipulation. In particular, the edge-quadtree is used to transform a bitmap, representing an image composed of lines and curves, into a quadtree. The nodes of the edge-quadtree contain the coordinates of the segments which approximate the curves.

The total number of nodes in a quadtree representation of a scene is referred to as its complexity. Many papers in the literature have dealt with the space efficiency of quadtrees, but so far no comparable results are available for the edge-quadtree. The number of nodes in the edge-quadtree depends on several factors: the first is the irregularity of the image (shape), the other two are the precision and the resolution, which are used by the algorithm to decide when to stop the subdivision process.

Aim of this paper is to evaluate the number of nodes of an edge-quadtree as a function of the shape, the precision and the resolution desired. The precision and the resolution are defined by one input value, while the shape is calculated using the fractal dimension. The proposed methodology uses the fractal measure also as an input value, because this gives more information on how the image occupies the space. An experimental approximation of the function is obtained by a suitable training of a neural network.

The second section of this paper describes what an edge-quadtree is and contains the definition of precision and resolution. A measure of the precision and of the resolution is given in the third section. In this section some classical methodologies for the definition of the shape are also surveyed, including fractals and box counting techniques. Modifications of these techniques are also considered, in view of their application to this problem. The problem of function approximation is treated in the fourth section, which also covers the topic of neural networks. The computational results are given in the fifth section, in which the kind of data employed and the way they have been generated are also explained. Some conclusions in the sixth section complete the paper.

2. THE EDGE QUADTREE

As the quadtree data structure evolved from work in different fields, it is natural that different versions of it can be found for each kind of spatial data. Its development has been motivated to a large extent by the desire to save storage by aggregating data having identical or similar values. According to Samet [16, 17], quadtrees can be differentiated on the following bases:

- *The type of data they represent*. Currently they are used for point data, curve, spaces and volumes.
- *The principle guiding the decomposition process*. A decomposition into equal parts at each level is termed a *regular decomposition*. On the other hand, the decomposition may adapt itself to the shape of the input image, resulting in an irregular tree (*irregular decomposition*) [22].

* This research is partially supported by the CSISEI-CNR and MURST.
† E-mail: {Wolfler, Schreiber}@elet.polimi.it.

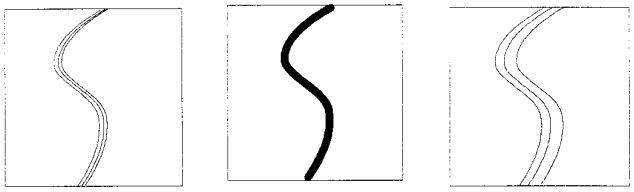
1077-3169/97 $25.00

• *The resolution* is a parameter that defines the number of times the decomposition process is applied. It may be fixed beforehand (i.e., input value) or it may be governed by properties of the input data. From an application viewpoint, the resolution defines the ability to discriminate between two objects (points, lines, etc.) near to each other.

The edge-quadtree [19] is useful when the image is composed of lines and curves [17]; therefore the edge-quadtree finds its relevance in applications such as medical information systems—where ECG and EEG have to be efficiently stored and retrieved [18]—and geographical information systems—where different types of information on maps are of monodimensional nature (e.g., contour plots). Other quadtree structures have been defined for this purpose, notably the PM-quadtree family. According to Samet [16, 17], the PM quadtrees differ from edge quadtrees as to how they treat the vertices (intersection of more lines), but they are still based on a recursive regular decomposition and they are influenced by the shape of the image. As the differences among these data structures lie only in the decomposition algorithm, our approach can be extended to PM-quadtrees as well.

In an edge quadtree, a region containing linear feature, or part thereof, is subdivided into four squares repeatedly until a square is obtained that contains a single curve that can be approximated by a single straight line. The image is inscribed in a box of size $2^n \times 2^n$ pixels and the space is recursively subdivided into four equal boxes until the stop condition is verified. The boxes correspond to the nodes of the quadtree. In the worst case, to represent a figure of $2^n \times 2^n$ pixels, it is necessary to use a quadtree n levels deep. The nodes of the kth level represent the blocks of $2^k \times 2^k$ pixels; the root is at level n and the nodes at level 0 represent single pixels. With this type of image the algorithm has to test two conditions before stopping the decomposition process: one, depending on the dimension of the box and on the number of lines and curves; the other, depending on the irregularity of the arc of curve included in the box. These conditions relate to the resolution and the precision of an edge-quadtree.

• *The resolution* is defined in the edge-quadtree when the maximal dimension of the box to explore is specified. The minimal resolution will be the dimension of the box into which the initial image is inscribed, the maximal resolution is the pixel. When the resolution is fixed, the decomposition process will stop in the case it meets a box where there are only two intersections between the box itself and the curve or when the process arrives at the box whose dimension corresponds to the maximal resolution. In the second case, if there is only an approximation line in the box, the algorithm approximates it with the segment, while when there are more lines inside the box it stores a node



SCHEME 1

with as many segments as necessary to approximate the picture. See Scheme 1.

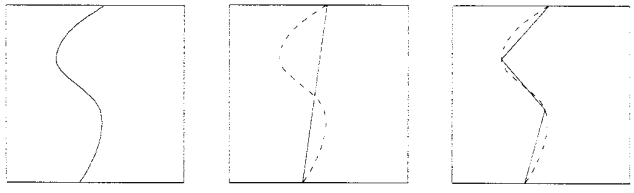
• *The precision,* expressing the fidelity with which a shape is represented, is measured by the value of the maximal distance between the points of the curve and the segment approximating the curve, measured on the perpendicular to the segment,

$$p = \max\left(\frac{|ax + by + c|}{\sqrt{a^2 + b^2}}\right) \tag{1}$$

The maximal precision is obtained when the segment and the curve are coincident and it is expressed by setting to zero the maximal accepted distance between the curve and the segment. The minimal precision is the diagonal of the box into which the line to be approximated is inscribed. See Scheme 2.

The technique used to store the tree coming from the decomposition process is the linear quadtree, proposed by Abel and Smith [1] and Gargantini independently [5]. This methodology is more efficient than that which uses pointers. In fact, even if it seems natural for the algorithm to use pointers to store a hierarchical data structure, the linear quadtree, based on locational codes, uses less memory. See, for instance, [21].

While the algorithm to convert the image into the quadtree is theoretically well defined, in our case we have to find all the possible limit configurations. Also, we have to prevent the algorithm from stopping in a box which contains a particular case; since we want to decompose an



SCHEME 2

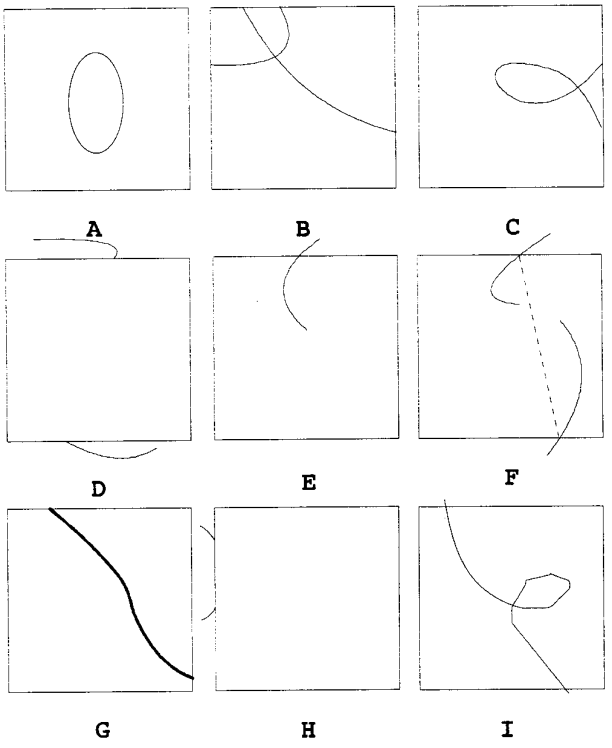


FIG. 1. The nine particular cases.

image according to general criteria. Figure 1 illustrates the cases; the pseudocode follows [18].

We define

- Q, the rectangular box
- x_0, the x coordinate at the right top of the box
- y_0, the y coordinate at the right top of the box
- k, the dimension of the box (edge-quadtree level)
- *inters*, the number of intersection points between the image and the sides of the box
- NW, NE, SW, SE, the parameters of the four boxes of the next level respectively

then

```
PROGRAM EDGE_QUADTREE[Q(x0, y0, k)]
begin
    if ¬∃ a black pixel ∈ Q
    then exit; {there are no lines into the box}
    else begin {examine the intersections with the box edges}
        if inters > 2 {there is more than one line}
          ∨ inters = 0 {all the lines are inside the box
          (Fig.1A)}
          ∨ inters with the same side ≥ 2
                {it is impossible to approximate the line
                (Figs.1B, 1C)}
          ∨ inters = 2 with two different sides ∧¬∃ inter-
          nal pixel {(Fig.1D)}
          ∨ inters = 1 {the line starts inside Q (Fig. 1E)}
        then begin {it is necessary to subdivide further
        the box}
            EDGE_QUADTREE(NW)
            EDGE_QUADTREE(NE)
            EDGE_QUADTREE(SW)
            EDGE_QUADTREE(SE)
            end;
        else if inters > 2 on the same side, they adjoin
        and no points fall inside the box
            {the line is tangential to one side (Fig. 1H)}
            then approximate the line with the segment
            else begin{there are two intersections on differ-
            ent sides}
                calculate the equation of the straight line
                which connects the intersection points;
                ∀ black pixel ∈ Q calculate the distance
                from the line;
            if max(distances) > fixed precision value
            {(Fig.1I)}
                ∨ the number of black pixel < 2/3 length of
                the segment
                    {the line is partioned (Fig.1F)}
            then begin {we have to subdivide further}
                EDGE_QUADTREE(NW)
                EDGE_QUADTREE(NE)
                EDGE_QUADTREE(SW)
                EDGE_QUADTREE(SE)
                    end;
                else approximate the line with the segment
            end;
        end;
    end;
end;
```

At the end of the creation process, each item of the data structure contains:

- the locational code of the box;
- its decomposition level;
- the coordinates of the intersection points between the curve and the box edges.

A detailed description of the algorithm can be found in [18].

3. RESOLUTION, PRECISION AND SHAPE

The complexity in space is defined as the total number of nodes constituting the quadtree. An a priori knowledge of this quantity is of considerable interest in problems involving tree search, because time and space complexities are interrelated [4, 11].

In this section we examine the problems related to the evaluation of the space complexity of an edge-quadtree as a function of shape, resolution and precision.

The same value can be used to specify the resolution and the precision, as the diagonal of the minimal box also represents the maximal distance between the points of the curve and any point of the segment which approximates the curve.

As to the shape, two approaches have been explored. The first one—a raster to vector conversion—is to open any closed line and to transform it in $f(x, y)$ or $f(\rho, \theta)$. A similar transformation has been used in pattern recognition in the case of non-self-intersecting closed curve [14].

However, since the goal of this work is to find a procedure applicable even to images formed of several lines, an alternative, more fruitful approach has been adopted: it consists of using the fractal dimension to define the input shape. This technique can be directly applied to the array of pixels as well as to fractal curves. The next sections explain the methodology.

3.1. *Box Counting*

To understand the fractal dimension it is necessary to shortly point out the definition of Hausdorff's measure and of measures derived from it. A deeper presentation of this topic can be found in [2], [10], [13].

The fractal dimension is related to the Hausdorff measure defined as

$$H^s = \lim_{\delta \to 0} H^s_\delta(E) \begin{cases} 0 & s < \bar{s} \\ \infty & s > \bar{s}, \end{cases} \tag{2}$$

where E is the set to cover, δ is the diameter of the subsets which cover E and s is the dimension of E. It is possible to demonstrate that there is a value $\bar{s}$ of s for which the limit exists and its value is finite. This value is the dimension of the set E.

Box counting is an easy alternative to the Hausdorff's distance. Given a set E, the idea is to use only regular boxes to cover E and to measure its dimension. Let a_δ be a uniform division of space with boxes whose diameter is δ. If $\{a_i;\ i = 1, \ldots, N\}$ is the set of boxes which has an intersection with E, we can write:

$$E = \bigcup_{i=1}^{N} a_i \tag{3}$$

$$H^s_\delta(E_\delta) = N(\delta) \cdot \delta^s \tag{4}$$

$$N(\delta) = c \cdot \delta^{-s}. \tag{5}$$

Equation (5) is a sufficient condition in order to have a finite value for $H^s(E_\delta)$, which is the limit of $H^s_\delta(E_\delta)$ when δ goes to 0.

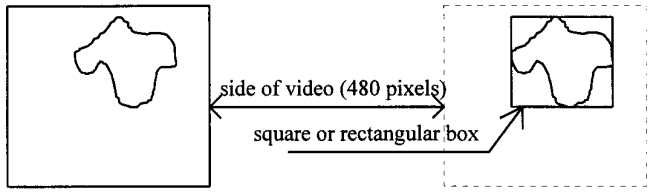


FIG. 2. The (x, y)-extent of an image.

We can say that s is the dimension of E and c its measure. Using a logarithmic operator (Eq. (6)), we get

$$\log N(\delta) = -s \log \delta + \log c \tag{6}$$

$$s = \frac{\log N}{\log (1/\delta)} - \frac{\log c}{\log (1/\delta)} \tag{7}$$

When δ becomes very small the last term in Eq. (7) goes to 0 and we may write Eq. (8), which gives the expression normally used to calculate the fractal dimension. With Eq. (9) we also define the logarithmic value of the fractal measure:

$$s = \lim_{\delta \to 0} \left(\frac{\log N}{\log (1/\delta)} - \frac{\log c}{\log (1/\delta)} \right) = \lim_{\delta \to 0} \frac{\log N}{\log (1/\delta)} \tag{8}$$

$$\log c = \log N(\delta) - s \log \frac{1}{\delta} \tag{9}$$

3.2. *Implementation Problems*

There is an obvious difficulty in using Eq. (8): it is not possible to evaluate the limit $\delta \to 0$. The lines in an image are made of pixels and δ cannot decrease as much as in the case the line thickness is null: we have to stop at the box whose dimension is the pixel. Moreover we never have an image made of fractal curves and so we look for the finite measure of the dimension of our lines. For these reasons while we can not decrease δ too much—it would not be useful either—we may decrease it enough to apply Eq. (8).

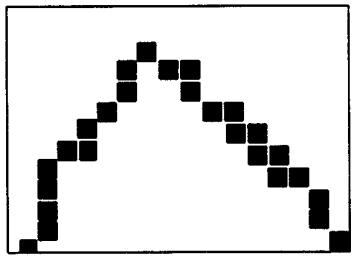


FIG. 3. A line made by pixels.

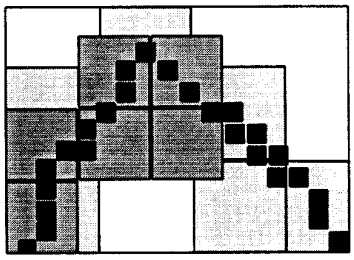


FIG. 4. A line and the boxes that cover it.

In order to obtain a stabilized and consistent fractal dimension, we had to solve some problems. The first one is related to the fact that the image may be smaller than the maximal area we considered. The area we use is the monitor screen (480 pixel × 480 pixel). If we apply box counting at the monitor surface when the figure is smaller, we make a mistake and we obtain a wrong measure. To solve this problem we inscribe the image into its (x, y) − *extent* (Fig. 2).

For every value of δ we have to compute the number of boxes $N(\delta)$ which have an intersection with the image, i.e., the boxes which contain at least one pixel of the curves. Then we plot $\log N(\delta)$ as a function of $\log(1/\delta)$. The slope of the straight line calculated using a linear regression gives the fractal dimension s.

The calculation is affected by the presence of noise: either "real" noise in experimental data or round-off noise in numerical computations [8]. The real noise concerns the computation of $N(\delta)$. Lines and curves are made by a number of pixels which is variable with the slope of the line (Fig. 3). For this reason, for some values of δ, the pixels of the image could intersect a number of boxes greater than expected—thus resulting in a value of $N(\delta)$ greater than expected (Fig. 4).

The points with a value of $N(\delta)$ affected by noise modify the slope of the linear regression. To minimize this effect we want the greatest number of points in the log/log plane. This means that the algorithm searches the greatest common divisor between the two sides of the (x, y)-*extent*.

When $1/\delta$ is large, the value of s is near 2, because $N(\delta)$ increases as if we had a surface. However, when $1/\delta$ decreases too much ($s \to 1$ or less), then the problem is to define a range for $1/\delta^n$—the latter varying with each picture.

Given $\{(x_i, y_i);\ i = 1, \ldots, n\}$, the set of points in the log/log plane, we calculate the slope of the linear regression using only two points according to equation (10), thus obtaining the upper and the lower bound for $1/\delta$.

$$a_j = \frac{2\sum_{j=k}^{k+1} x_j y_j - \sum_{j=k}^{k+1} x_j \sum_{j=k}^{k+1} y_j}{2\sum_{j=k}^{k+1} x_j^2 - \sum_{j=k}^{k+1} x_j \sum_{j=k}^{k+1} x_j} \tag{10}$$

This new set of values $\{a_j;\ j = 1, \ldots, n-1\}$ gives information on the original points in the log/log plane.

The value of the slope is always greater than or equal to 1.

The lower bound is computed by looking for the last value of $a_j \geq 1$. (I.e., if $a_{n-1} < 1$ the lower bound of $1/\delta$ is set to a_{n-2}, etc. until we find the first value of $a_j \geq 1$.)

The upper bound is defined in two different modes: one based on empirical considerations; the other based on a statistical operator. The computational results show that, for some pictures, there is a great difference between the first values of a_j (i.e., a_1, a_2, etc.) and all the other values (Fig. 5). This means that, at the first subdivision of the space, the algorithm does not distinguish between single lines and the background, but it "sees" the image as a full picture. One upper bound is based on the inequalities

$$a_j - a_{j-1} \geq 0.2$$
$$a_j - a_{j-2} \geq 0.2 \text{ for } (j = 1, \ldots, i-1)$$
$$a_j - a_{j-3} \geq 0.2$$

with $1 \leq a_j \leq 2$. If the three inequalities are simultaneously verified the algorithm calculates the linear regression using the set of points (x_i, y_i), with $i = j, \ldots, n$.

Let

- y_k be the initial values;
- $\hat{y}_k$ be the value obtained from the linear regression with the x_k values as input;
- $\bar{y}$ be the mean of the y_k;

then

$$R^2 = \frac{\sum_{k=1}^{n} (\hat{y}_k - \bar{y})^2}{\sum_{k=1}^{n} (y_k - \bar{y})^2}, \tag{11}$$

called the coefficient of determination, can be used to

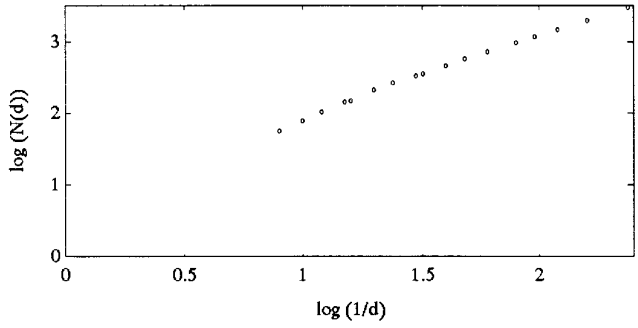


FIG. 5. Points in the log–log plane.

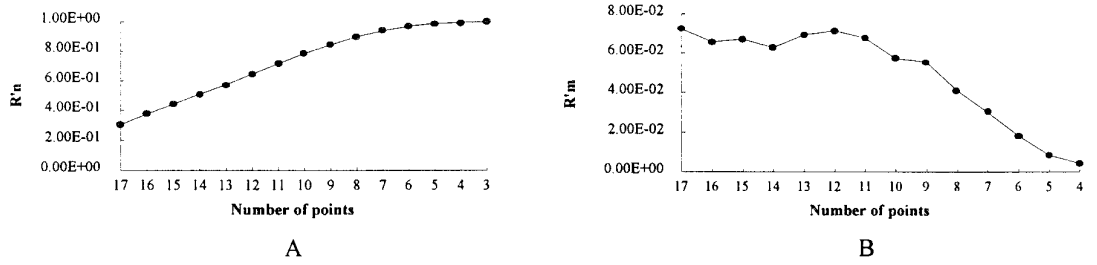


FIG. 6. The plot of R'_n (number of points) and R'_m (number of points).

estimate the adequacy of the regression model [9]. It is clear that $0 \leq R^2 \leq 1$.

By calculating R_n^2 with $n = 3, \ldots, i-1$ and plotting it as a function of n, we obtain the graph of Fig. 6A. The sought function of R_n^2 has a point of maximum, but for every picture we obtain a decreasing monotonic function (Fig. 6A). This means that we can not obtain the value of the upper bound from R_n^2. As it is necessary to evaluate the amount of the increments of R_n^2, we calculate $R'^2_m = R^2_{n+1} - R^2_n$ with $m = 1, \ldots, n-1$, thus obtaining the graph of Fig. 6B.

For some pictures this function has a point of maximum and, in this case, we use the value of m (i.e., $n-1$) to set the upper bound of $1/\delta$. The values of δ rejected by both (empirical and statistical) correspond to boxes less than 10^D, where D is the fractal measure plus one for the noise.

This result is in accordance with the work of Theiler [20] which sets to 10^D the minimum number of boxes required for δ to be a "good" value.

There is not a biunivocal correspondence between the fractal dimension and the number of nodes. For the same resolution and precision there are pictures which have the same fractal dimension, but a different quadtree. The number of nodes depends on the shape as well as on the image spatial distribution. In fact if the algorithm has to store a picture made by a closed line whose fractal dimension is $\bar{s}$, it uses a number of nodes smaller than that used to store an image made by two or more closed linear lines with the same fractal dimension $\bar{s}$. Moreover, as the resolution increases, the number of nodes increases more in the second case than in the first case. To solve this problem we need a parameter representative of how much space the picture occupies: the fractal measure. In fact, from Eq. (5) we have that $c = N(\delta) \cdot \delta^s$, where N is the number of boxes having an intersection with the picture and $\bar{s}$ is the box dimension.

Equation (8) gives the fractal measure:

$$\log c = q = y - mx = \frac{\sum_{i=1}^{n} y_i - s \sum_{i=1}^{n} x_i}{N} = \frac{\sum_{i=1}^{n} \log N_i(\delta) - \sum_{i=1}^{n} \log 1/\delta_i}{N}. \tag{12}$$

Two different fractal measures are needed: one to represent how an image "sits" in the (x, y) − *extent*, the other for the entire space (i.e., in our case the video), the equation being the same with two different values for δ. In the first case δ_0 is the side of the (x, y) − *extent*, with $\delta_1, \delta_2, \ldots, \delta_i$ the fractional values of δ_0. In the second δ_0 is the side of the video and as $\delta_1, \delta_2, \ldots, \delta_i$ are the fractional values of δ_0.

We call relative fractal measure (r.f.m.) the first one and absolute fractal measure (a.f.m.) the second one.

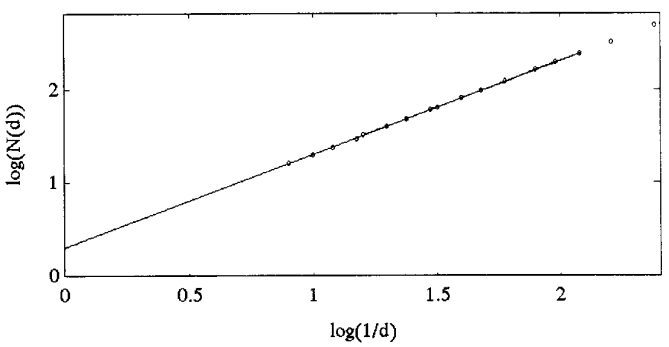


FIG. 7. Points in the log–log plane (straight line).

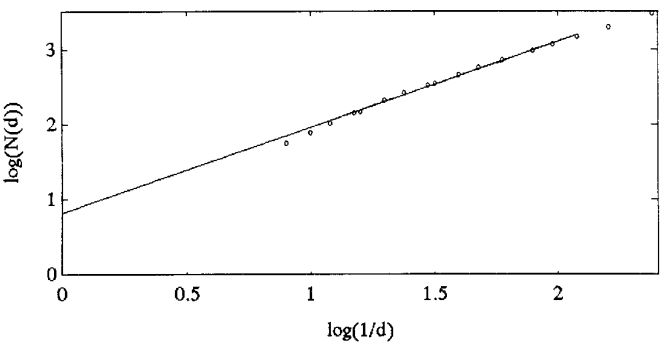


FIG. 8. Points in the log–log plane (with noise).

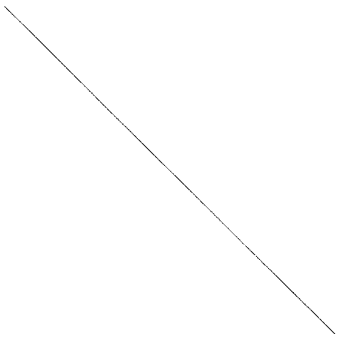


FIG. 9. Input values: f.d. = 1, r.f.m. = 0, a.f.m. = 0.

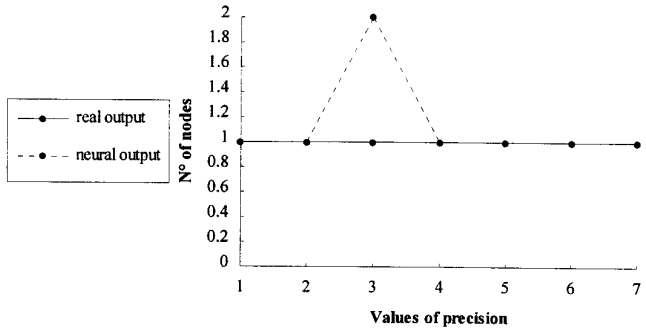


FIG. 10. The real output and the neural output of Fig. 9.

We have identified three different situations. The first one corresponds to points located on a straight line (Fig. 7). This means that the image is a square surface or a straight line (i.e., $s = 1$ or $s = 2$). For the second case, corresponding to a new set $\{a_j\}$ decreasing in value, it is possible to identify an asymptotic value. In the third case the points are affected by noise and very different values for $\{a_j\}$ may arise (Fig. 8).

4. NEURAL NETWORKS

Finally we ought to find a scalar function that approximates the function we are looking for, which links the space complexity of an edge quadtree to the four input parameters. The neural networks provide the answer. Given a continuous and multivariable function $f(x)$ defined on the set X, an approximation error $\varepsilon > 0$, and an approximation function $\hat{f}(x, w)$, which depends continuously on the parameter vector w and on x, the approximation problem consists of choosing vector $\hat{w}$, so that

$$\rho[f(x), \hat{f}(x, \hat{w})] \leq \varepsilon \quad \forall w \in W, \forall x \in X,$$

where W denotes the set of acceptable parameters and ρ the distance between two functions (L^2 norm, for instance).

For the sake of simplicity, we will consider the approximation of scalar functions only.

The existence of a good approximation depends on the

TABLE 1
Neural Network Configuration and Computational Results

N. layer	Normalization	N. neurons	N. steps	Learning error	% of right
1	$\frac{X - \min}{\max - \min}$	20	11,000	0.050	90
1	$\frac{\ln X}{\ln \max}$	30	11,520	0.052	92
1	$\frac{\ln X}{\log (1/0.0001)}$	20	11,680	0.052	94
2	$\frac{\ln x}{\log (1/0.0001)}$	15–7	9,000	0.045	96
2	$\frac{\ln X}{\ln \max}$	20–10	9,500	0.049	93
2	$\frac{\ln X}{\log \max}$	15–7	8,000	0.045	94.5
2	no one	15–7	10,000	0.049	94
2	$\log X$	15–7	11,000	0.049	94
2	$\frac{\ln X}{\log (1/0.0001)}$	15–7	10,000	0.056	90
2	$\frac{\ln X}{\log (1/0.0001)}$	15–7	10,000	0.055	91

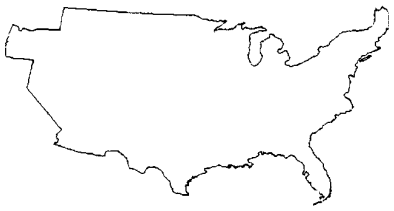


FIG. 11. Input values: f.d. = 1.1, r.f.m. = 1.31, a.f.m. = 2.5.

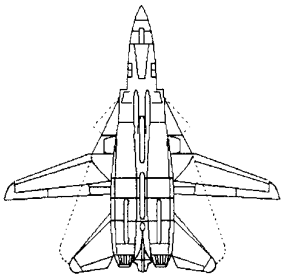


FIG. 13. Input values: f.d. = 1.4, r.f.m. = 0.91, a.f.m. = 1.42.

class of functions $\hat{f}(x, w)$ belongs to. Function $\hat{f}(x, w)$ is expressed as

$$\hat{f}(x) = \sum_{i=1}^{n} w_i \phi_i(x)$$

The function $\phi_i(x) : R^m \rightarrow R$ must be chosen according to the following conditions:

- $\phi_i(x)$ must be a *basis* for the class of functions $f(x)$ to be approximated;
- a "good" approximation has to be obtained with a finite number of terms;
- unnecessary functions have to be easily eliminated from the linear combination.

For the first condition the following equation must be verified:

$$f(x) = \sum_{i=1}^{\infty} w_i \phi_i(x). \tag{13}$$

A good approximation is obtained when the norm decreases rapidly with n:

$$\left\| \hat{f}(x) = \sum_{i=1}^{n} w_i \phi_i(x) \right\|$$

Many possible bases for a continuous $f(x)$ have been used, among them the spline function with fixed nodes, the basis of polynomials, etc. In [6, 7] the properties of neural networks as a basis have been studied. In particular, it has been proved that a feedforward neural network with one hidden layer, composed of neurons with sigmoidal activation function, and a linear output, can approximate any continuous function to the desired accuracy: in other words, it is termed a "universal approximator."

There are two reasons to consider neural networks a good choice as a basis: if we choose a function g with a local nonlinear property, we obtain a good approximation with a sum of a finite number of functions belonging to the basis. In addition we can easily eliminate the unnecessary functions from the basis by stopping the learning process before it arrives to the absolute minimum or by introducing in the objective function a term which penalizes the unnecessary parameters.

The proof that the neural network is a good approximation does not explain why it has been preferred to the polynomial basis, since both have the Weierstrass property. The difference between the neural network and the polynomial basis lies in the imposed constraints.

The polynomial basis has a number of possible terms that increases rapidly with the dimension of the input vector. A neural network, on the other hand, uses a limited number of iterations since it learns to select the right combination of inputs. When we increase the number of neurons we

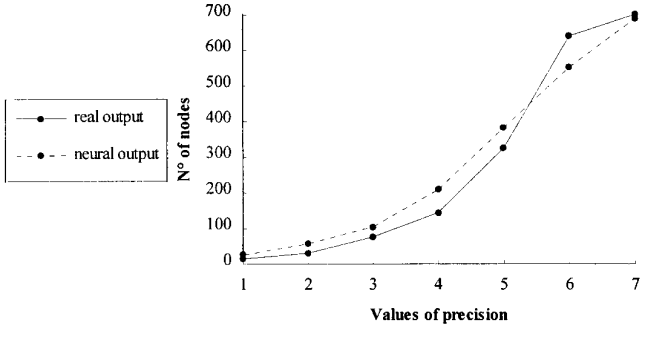


FIG. 12. The real output and the neural output of Fig. 11.

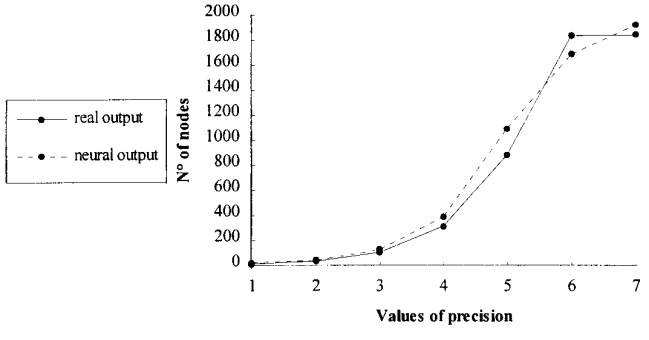


FIG. 14. The real output and the neural output of Fig. 13.

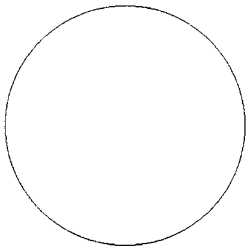


FIG. 15. Input values: f.d. = 1.03, r.f.m. = 1.22, a.f.m. = 1.97.

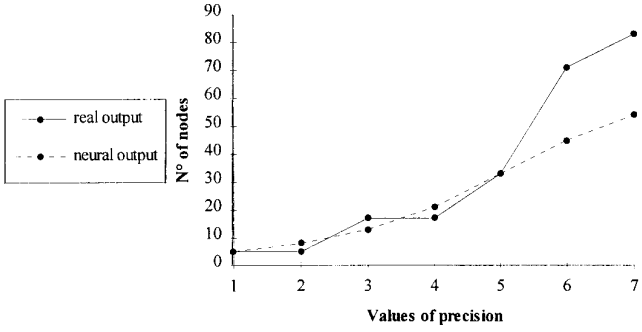


FIG. 16. The real output and the neural output of Fig. 15.

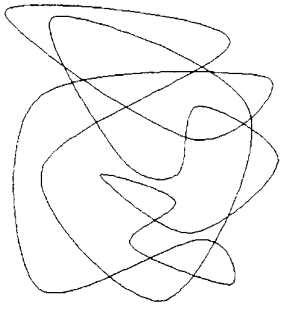


FIG. 17. Input values: f.d. = 1.08, r.f.m. = 2.12, a.f.m. = 3.99.

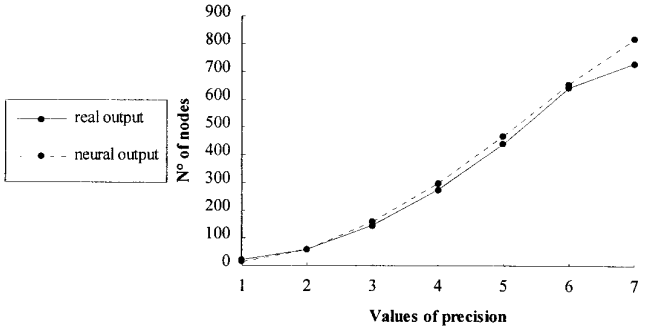


FIG. 18. The real output and the neural output of Fig. 17.

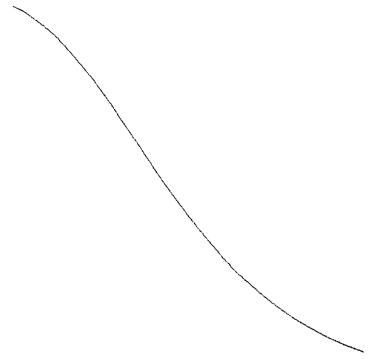


FIG. 19. Input values: f.d. = 1, r.f.m. = 0.69, a.f.m. = 1.38.

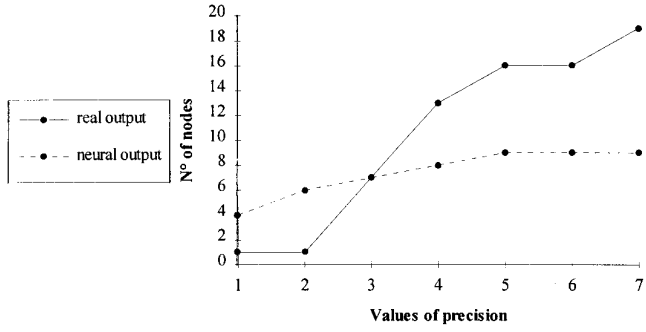


FIG. 20. The real output and the neural output of Fig. 19.

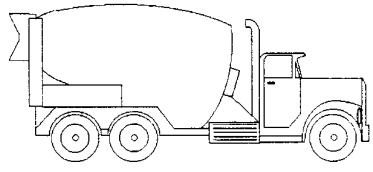


FIG. 21. Input values: f.d. = 1.17, r.f.m. = 1.73, a.f.m. = 3.19.

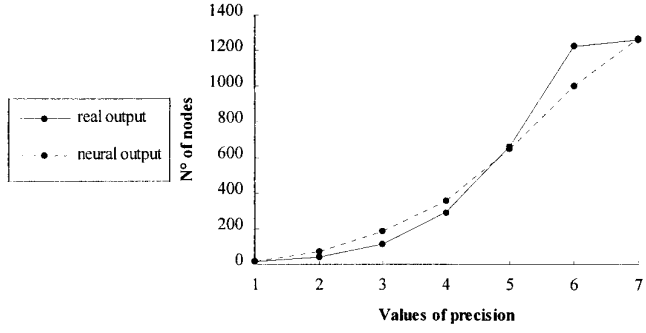


FIG. 22. The real output and the neural output of Fig. 21.

increase its approximation capability, not the degree of the polynomial regression.

The last problem is how to configure the network. There are many hypotheses on the "optimal" number of neurons and hidden layers, but none about the rules to apply. In the literature there are only some indication on the upper bound; the problem is still undefined and matter of research [7, 12]. In particular, the hidden layers may be one or two, depending on the particular application. There are no definite results, but it seems that to solve our problem it is better to use a network with two hidden layers [6]. Our experiments confirm this hypothesis.

5. COMPUTATIONAL RESULTS

We used a feedforward neural network with sigmoidal non linearities based on a commercial package (Neuralist version 1.3 EPIC systems Corp.). In Table 1 we report the result of the experiments with different configurations. In the first column we indicate the number of hidden layers; the network with two hidden layers performs the best. The second column reports the type of normalization used. We normalized input and output data in the range [0, 1]. The input data are easy to normalize, whereas we employed three different types of normalization for the output data. The best one, as in Table 1, is $(\ln X_i)/(\log (1/0.0001))$.

We also varied the number of neurons with little influence on the output. In the fourth column we report the number of steps.

The learning error is indicated in the fifth column. It is the average error between the network output and the expected output, calculated on all training sets. When the difference between the network output and the expected output is less then a predefined value (i.e., 0.01), we consider that the network succeeded. The number of right examples is expressed in column sixth of Table 1 as the percentage of right answers. Finally the last two rows of Table 1 show the importance of a.f.m. and r.f.m. on the learning error.

To evaluate the effectiveness of our approach we created a set of 100 images composed of lines with different thickness; 50 images were used to train the network and 50 to validate it. Note that the pictures were composed of a different number of lines. For every picture we built the quadtree with seven different

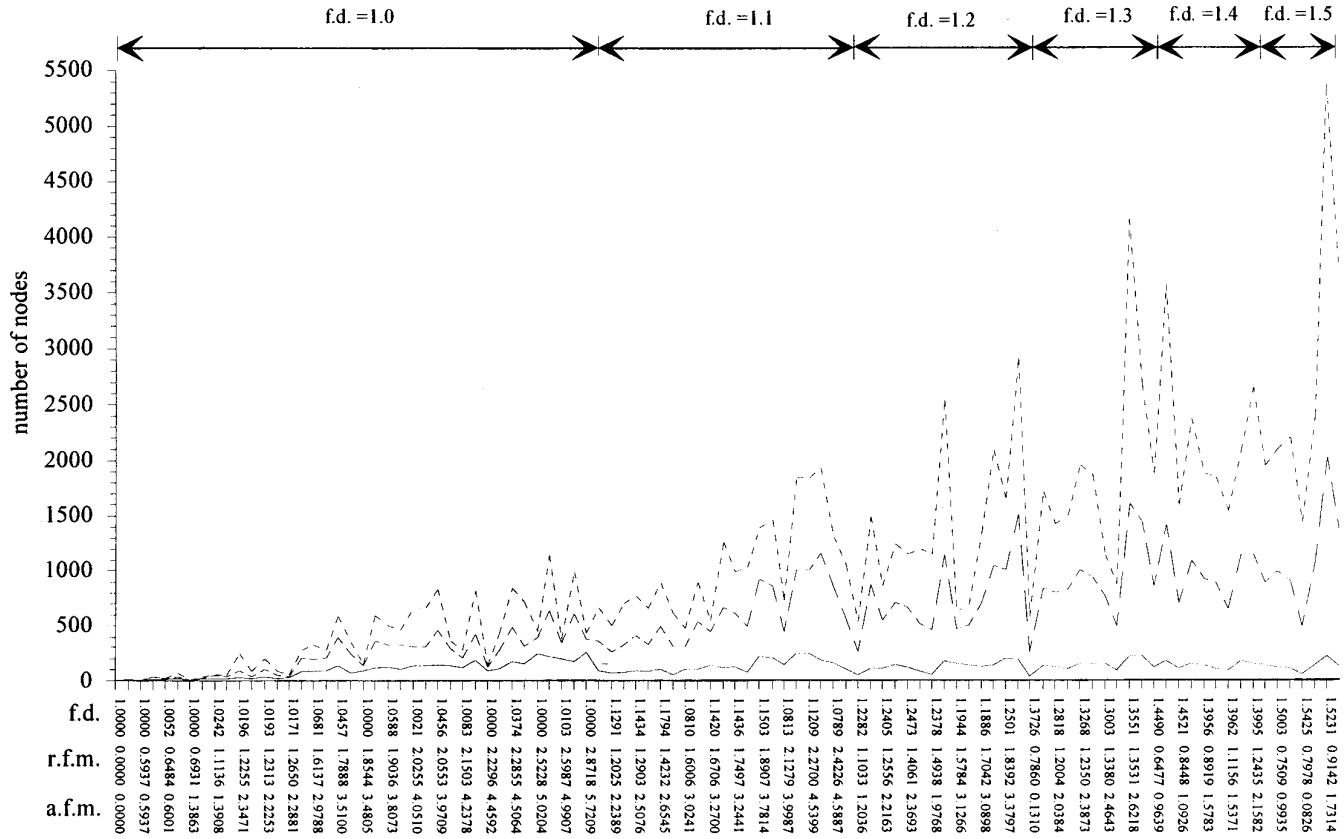


FIG. 23. Number of nodes vs fractal dimension and fractal measure.

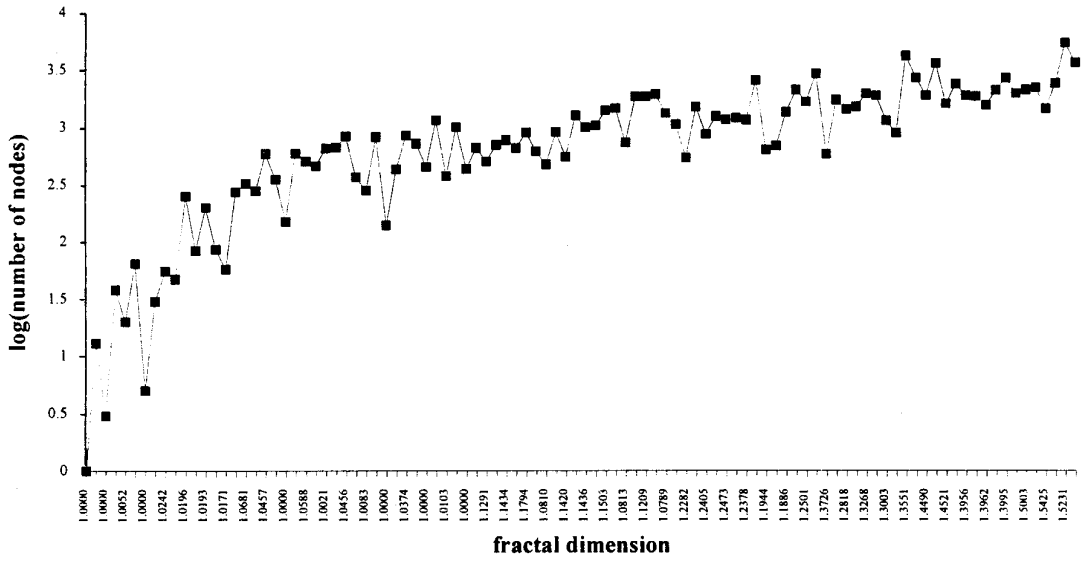


FIG. 24. The log number of nodes as a function of the fractal dimension.

precisions, ranging from 100 to 2 pixels for the box side. We also calculated the fractal dimension and the two fractal measures.[1]

In Figs. 9–14 three images used in the training set are given, together with their results. In each graph we report the real function and the estimated function. Figures 15–22 represent four images of the validation set and their relative graphs. Under every picture we report the value of the fractal dimension (f.d.), the absolute fractal measure (a.f.m.) and the relative fractal measure (r.f.m.). Let us notice that we obtain a good result; in fact the neural network gives an effective estimation of the space complexity. When this is not true we obtain that the estimation error is less than 20% in the average (but for Figs. 15, 19) and, in any case, the order of magnitude is maintained.

Another observation is that the neural network gives a better answer when the input image is complex with respect to images made of one line only. This behavior is caused by the type of normalization used and by the fact that the training set is not homogeneous. In fact the normalization with the log operator compresses the values, so the neural network does not perceive the difference between small values (e.g., 1 and 2) once they are normalized. When the image is more complex, the difference between the number of nodes increases as the precision increases, so that the neural network is capable to recognize the difference. The training set is not homogeneous because complex images are a little more numerous in it than simple ones, but we think this is enough to influence the results. When the fractal dimension is calculated without noise, we obtain a better result.

[1] The time needed for these calculations is much lower than that needed to build the actual edge quadtree.

6. CONCLUSIONS

6.1. *Discussion of the Results*

The idea that a relation exists between the number of nodes and the three characteristics of the image (shape, resolution, and precision) has been proved and the preceding results show the effectiveness of the solution used. The approach used is experimental, so in this work an analytical function that represents the relation is not reported.

In any case in Fig. 23 the plot of the number of nodes vs. the fractal dimension and the ordered values of the two fractal measure is shown for the 100 figures at the levels of precision of 20, 5, and 2 pixels. At the bottom of the figure as x coordinates we can read the fractal dimension, the relative fractal measure and the absolute fractal measure, whereas at the top of the figure we show the interval of points with the same fractal dimension.

We plot in a two-dimensional space a function of four variables. The points are initially sorted by the fractal dimension and then, once identified the group of figures with equal fractal dimension, we sort these pictures by the relative fractal measure. It is not possible to sort the figures by the absolute fractal measure too, but the absolute fractal measure and the relative fractal measure are correlated. The absolute fractal measure is useful when two pictures with the same fractal dimension and relative fractal measure have a different number of nodes and different values of absolute fractal measure: this proves that a picture is univocally defined by the three inputs.

Let us comment on Fig. 23. The first observation is that the distance (number of nodes) between the three lines (low, medium, and high precision) increases when the fractal dimension increases. The second relates to figures with

valleys and peaks in the number of nodes. As said before, some of this irregular behaviour can be explained on the basis of absolute fractal measure. Moreover let us note that the first figure with a different fractal dimension corresponds to a valley in the number of nodes, although the fractal dimension increases. This results from the fact that the pictures with the same fractal dimension are sorted by their fractal measure, so the first figure of the series has a very low value of the fractal measure.

We first conjectured of an exponential function of the kind $C \cdot 4^{p \cdot [f.d. - 1]}$, with the constant C depending on the fractal measure, p being the precision and $f.d.$ the fractal dimension. As we plotted the log of the number of nodes at the best precision level as a function of the fractal dimension, Fig. 24, we realized that the complexity could be a logarithmic function or a polynomial. Further research is needed to better qualify the type of curve.

6.2. *Future Work*

The goal of this work was to find a methodology to obtain the number of nodes of an edge quadtree as a function of three input parameters: shape, precision, and resolution. We found four parameters (fractal dimension—related to the shape—relative and absolute fractal measure—concerning the position of the figure in the image space—and precision—with which we implicitly accounted also for resolution) which permit a neural network to give, as its output, the number of nodes. This approach seems effective, but before considering it a success it would be necessary to define the range of δ with other methodologies. It also would be interesting to compare the box counting technique with other methods and to optimize the neural network's configuration.

An interesting extension of this method could be its application to colored region images and to 3D objects.

ACKNOWLEDGMENTS

The authors would like to thank G. Storti Gajani, L. Piroddi, L. Mussone, and F. Sansò for their helpful comments and discussions on the subject of this work. Thanks also to G. Cantalupo and A. Banfi for their contribution to the programming of the experiments with the neural network.

Last, but not least, we acknowledge the work of the anonymous referees for the precious contribution to the improvement of the quality of the paper.

REFERENCES

1. D. J. Abel and J. L. Smith, A data structure and algorithm based on a linear key for a rectangle retrieval problem, *Comput. Vision Graphics Image Process.* **24,** 1983, 1–13.
2. M. Barnsley, ''Fractals Everywhere,'' Academic Press, San Diego, 1988.
3. P. J. Davis, ''Interpolation and Approximation,'' New York, Blaisdell, 1963.
4. P. Flajolet, G. Gonnet, C. Puech, and J. M. Robson, The analysis of multidimensional searching in quad-tree, in *Proceedings of 2nd SODA, San Francisco, 1991,* pp. 100–108.
5. I. Gargantini, An effective way to represent quadtrees, *Comm. ACM* **25,** No. 12, 1982, 905–910.
6. F. Girosi and T. Poggio, Networks for learning, in *Neural Networks: Concepts, Applications and Implementation,* (P. Antognetti and V. Milutinovic, Eds.), pp. 110–154, Prentice–Hall, Englewood Cliffs, NJ, 1991.
7. R. Hecht-Nielsen, *Neurocomputing,* Addison–Wesley, Reading, MA, 1991.
8. R. C. Hilborn, *Chaos and Nonlinear Dynamics,* Oxford University Press, London.
9. P. G. Hoel, *Introduction to Mathematical Statistics,* Wiley, New York.
10. B. B. Mandelbrot, *The Fractal Geometry of Nature,* Freeman, New York, 1977.
11. B. G. Mobasseri, A new quadtree complexity theorem, in *Proceedings of 11th ICPR, 1992,* Vol. 4, pp. 389–392.
12. L. Mussone, Le reti neurali artificiali nei trasporti, *Trasporti & Trazione,*. **2,** 1994, 56–72.
13. H. Peitgen, H. Jürgens, and D. Saupe, *Fractals for the Classroom. Part One. Introduction to Fractals and Chaos,* Springer-Verlag, New York, 1992.
14. N. Pérez de la Blanca, J. Fdez-Valdivia and J. A. Garcia, Characterizing planar outlines, *Pattern Recognition Lett.* **14,** 1993, 489–497.
15. L. Piroddi, *Reti neurali per il controllo predittivo non lineare,* Ph.D. thesis, Politecnico di Milano, 1995.
16. H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS,* Addison–Wesley, New York, 1990.
17. H. Samet, *The Design and Analysis of Spatial Data Structures,* Addison–Wesley, New York, 1989.
18. F. Schreiber and E. Proserpio, Algoritmi per la memorizzazione e la gestione di curve mediante Quadtree, *Riv. Informat.* **21,** No. 3, 1991, 253–273.
19. M. Shneier, Two hierarchical linear features representations: Edge pyramids and Edge quadtrees, *Comput. Graphics Image Process.* **17,** No. 3, 1981, 211–224.
20. J. Theiler, *Phys. Lett. A* **155,** 1991, 480.
21. T. R. Walsh, On the size of Quadtrees Generalized to d-dimensional Binary Pictures, *Comput. Math. Appls.* **11,** No. 11, 1985, 1089–1097.
22. X. Wu, Image coding by adaptive tree-structured segmentation, *IEEE Trans. Information Theory,* **38,** No. 6, 1992, 1755–1767.