Adaptive Approximate Nearest Neighbor Search for Fractal Image Compression

Chong Sze Tong, *Member, IEEE*, and Man Wong

***Abstract*—Fractal image encoding is a computationally intensive method of compression due to its need to find the best match between image subblocks by repeatedly searching a large virtual codebook constructed from the image under compression. One of the most innovative and promising approaches to speed up the encoding is to convert the range-domain block matching problem to a nearest neighbor search problem. This paper presents an improved formulation of approximate nearest neighbor search based on orthogonal projection and pre-quantization of the fractal transform parameters. Furthermore, an optimal adaptive scheme is derived for the approximate search parameter to further enhance the performance of the new algorithm. Experimental results showed that our new technique is able to improve both the fidelity and compression ratio, while significantly reduce memory requirement and encoding time.**

I. Introduction

A NUMBER of papers on fractal image encoding have been published since the pioneering idea of Barnsley and Sloan in 1988 (see [1] for the latest survey). It has since been used for special image archive applications such as MR, ECG, and space images [2]–[4], feature extractions [5], image signatures [6], image retrievals [7]–[10], texture segmentation [11] and many other image processing applications. Although it suffers from long encoding times, it has the advantage of very fast decompression as well as the promise of potentially very high compression ratios. These properties made it a very attractive method for applications in multimedia: for example, Microsoft adopted it for compressing thousands of images in its *Encarta* multimedia encyclopaedia.

The basic fractal compression scheme partitions an image into range blocks, the encoding of each range block consists of finding the best affine transformation by searching a global domain block pool (its own virtual codebook). Specifically, for each range block $R = (r_{ij})$, we search the domain pool to find the domain block $\hat{D} = (\hat{d}_{ij})$ and a transformation τ such that $\tau(\hat{D})$ provides the best matching for R. The mean square error (MSE) is the usual distortion criterion. The transformation τ consists of a spatial contraction map C followed by massic transformation T that is the composition of a contrast scaling s and a luminance shift g, viz. $\tau = T \circ C$, $T(x) = \mathrm{s}^* x + \mathrm{g}$. For convenience, the domain blocks can be pre-contracted to form a pool denoted by $\Omega = \{D = C(\hat{D})\}$. Thus, the fractal encoding problem is

$$\min_{D\in\Omega} E(R,D) = \min_{D\in\Omega}\min_{\mathrm{s,g}} \|\mathrm{s}D + \mathrm{g}I - R\|^2 = \min_{D\in\Omega}\min_{\mathrm{s,g}} \sum (\mathrm{s}^* d_{ij} + \mathrm{g} - r_{ij})^2 \quad (1)$$

where I is a matrix whose elements are all ones.

The optimal affine parameters can be obtained by the method of least squares and are given by

$$s = \frac{\langle R - \bar{r}I, D - \bar{d}I\rangle}{\|D - \bar{d}I\|^2}, \quad g = \bar{r} - s\bar{d} \quad (2)$$

where $\bar{r}$, $\bar{d}$ are the average of the pixel intensity of the range block R and the contracted domain block D, respectively.

However, it is well studied that such a least square solution is undesirable since the solution takes no account of the constraint on scaling as required by the condition of contractivity and that the post-quantization of the parameters often lead to poorer results as compared with pre-quantization [12]. Thus, because the scaling coefficient must be constrained and both transformation parameters must be quantized, the quantity that we actually want to minimize is the collage error $\tilde{E}(R,D)$

$$\min_{D\in\Omega} \tilde{E}(R,D) = \min_{D\in\Omega}\min_{s_i,g_k} \|s_i D + g_k I - R\|^2 \quad (3)$$

where s_i, g_k are pre-quantized levels of the scaling and offset parameters.

The minimization problem posed by (3) is the so-called full search problem and is computationally intensive. Let N_D be the size of the domain pool Ω. Many time complexity reduction methods have been proposed, but most of them can only reduce the factor of proportionality in the $O(N_D)$ complexity, while only the tree search approach is able to fundamentally reduce the *order* of encoding time from $O(N_D)$ to $O(\log N_D)$. The idea of tree-structured search to speed up encoding has long been used in the related technique of Vector Quantization (see e.g., [13]). And many formulations of tree-search for fractal encoding have been proposed, e.g., by Caso *et al.* [14] and Bani-Eqbal [15] (see [1] for recent survey). But perhaps the most elegant and general formulation is that proposed by Saupe [16], which also has the advantage of incorporating a small set of fixed basis blocks (a real codebook) to improve compression quality. In this paper, we shall discuss some of the drawbacks of Saupe's formulation and propose a new modified formulation that is faster and with better coding performance.

Manuscript received December 5, 2000; revised January 28, 2002. This work was supported in part by the Hong Kong Baptist University Faculty Research under Grants FRG/99-00/II-16. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Lina Karam.

The authors are with the Department of Mathematics, Hong Kong Baptist University, Kowloon Tong, Hong Kong.

Publisher Item Identifier S 1057-7149(02)03979-9.

This paper is organized as follows. In Section II, Saupe's nearest neighbor search approach is reviewed and its drawbacks are examined. A new formulation based on orthogonal projection and pre-quantization of the fractal transform parameters is given in Section III. Enhancements to the new algorithm are described in Section IV, followed by experimental results and discussion in Section V. Finally we summarized our findings in the concluding Section VI.

II. Saupe's Nearest Neighbor Search

A. *Review*

One of the most innovative and promising approaches in speeding up fractal image coding was proposed by Saupe [16]. Starting from (1) and (2), Saupe proved the theorem

$$E(R,D) = \|R - \bar{r}I\|^2 \, g(\Delta(R,D)) \tag{4}$$

where $\Phi(.)$ is a normalized projection operator, $g(\Delta) = \Delta\sqrt{1-\Delta^2/4}$ and $\Delta(R,D) = \min(\|\Phi(R)+\Phi(D)\|, \|\Phi(R)-\Phi(D)\|)$.

The essence of the new form for the distortion in (4) is that since the function $g(\Delta)$ is monotonically increasing in the domain of interest, the minimization of $E(R,D)$ over all domain blocks D in the domain pool Ω is equivalent to minimizing $\Delta(R,D)$. This is in turn equivalent to finding the nearest neighbor of $\Phi(R)$ in the set of $\{\pm\Phi(D) : D \in \Omega\}$. The domain-range matching problem is now converted to the nearest neighbor searching problem between a transformed range block $\Phi(R)$ and the transformed domain blocks $\Phi(D)$.

There are a number of nearest neighbor search data structure and algorithms which perform significantly better than the brute force linear search, which requires $O(N_D)$ operations. Saupe adopted a variant of the kd-tree algorithm by Arya *et al.* [17]. Before the search, a tree data structure is built to store the transformed domain blocks, which requires $O(N_D \log N_D)$ operations. The running time for each query then takes $O(\log N_D)$ operations on average. The time complexity of the encoding step for each range block is thus reduced from $O(N_D)$ to $O(\log N_D)$.

B. *Drawbacks*

The approach described in Section II-A suffers from two drawbacks. Firstly, there is a very large memory requirement for storing the tree. For the 64K-size domain pool of a 512×512 image (using 8×8 range block size), we need two bytes to store the position of the domain block and 256 bytes ($8 \times 8 \times 4$, assuming four bytes for one floating point number) to store the coordinates of in one leave node of the kd-tree. So, 16 MB of memory is required just to store the leave nodes of the tree. For larger images or when multiple domain pools are required (e.g., in quadtree encoding, see Section IV), the memory requirement may be prohibitive for a personal computer or even a workstation.

The second and more serious drawback is that Saupe's theorem is only valid when *continuous* and *unconstrained* values of the scaling and luminance coefficients could be used. In practice, however, the coefficients are quantized to a small number of bits in order to obtain a reasonable compression ratio. Furthermore, for the affine transformation to be contractive, the scaling coefficient s is usually restricted to the interval between -1 and $+1$. As a result, the domain block found in the nearest neighbor search may not be the one that yields the minimum $E(R,D)$ when the transform coefficients are quantized and clipped.

To deal with the problem, Saupe proposed to search for not only the nearest neighbor, but also the next five or ten nearest neighbors. However it is still possible that the optimal or near-optimal domain block is not obtained in the list of nearest neighbors [16]. As discussed in Section I, the real issue is that Saupe has solved the minimization problem in (1), when we actually want to solve the full search problem in (3). In the next section, we present a modified formulation that is based on full search.

III. New Approximate Nearest Neighbor Search Algorithm

A. *Solving the Full Search Problem*

We start with the full search problem in (3). As advocated by Øien & Lepsøy [18] and by Tong & Pi [19], we subtract the DC component of all the image blocks (i.e., we project the image block vectors onto the orthogonal complement of the subspace spanned by the vector I). The full search problem becomes

$$\min_{D\in\Omega} \tilde{E}(R,D) = \min_{D\in\Omega} \min_{s_i,\bar{r}_k} \left\|s_i(D-\bar{d}I) - (R-\bar{r}_k I)\right\|^2. \tag{5}$$

Note that the fractal parameters are now parameters s and $\bar{r}$, instead of the conventional scaling s and luminance offset g. Let $\{s_i\}$, $\{\bar{r}_k\}$ be the quantized levels of the fractal parameters s and $\bar{r}$ respectively. The collage error using these quantized parameters is

$$\begin{aligned}\tilde{E}(R,D;s_i,\bar{r}_k) &= \left\|s_i\left(D-\bar{d}I\right) + \bar{r}_k I - R\right\|^2 \\ &= s_i^2\left\|D-\bar{d}I\right\|^2 + \|R-\bar{r}_k I\|^2 \\ &\quad - 2s_i\left\langle D-\bar{d}I, R-\bar{r}_k I\right\rangle.\end{aligned} \tag{6}$$

Noting that $D' = D - \bar{I}$ is orthogonal to the subspace spanned by I, using (2), we have

$$\left\langle D-\bar{d}I, R-\bar{r}_k I\right\rangle = \left\langle D-\bar{d}I, R-\bar{r}I\right\rangle = \mathrm{s}\left\|D-\bar{d}I\right\|^2.$$

Substituting this back into (6) and subtracting the expression for $E(R,D)$ in (1), we have

$$\begin{aligned}\tilde{E}(R,D;s_i,\bar{r}_k) - E(R,D) = (s_i-\mathrm{s})^2\left\|D-\bar{d}I\right\|^2 \\ +N^2(\bar{r}-\bar{r}_k)^2.\end{aligned} \tag{7}$$

Equation (7) gives the additional error caused by the pre-quantization of the fractal parameters. In [19], Tong and Pi analyzed this quantization error and derived an optimal quantization and bit allocation scheme for the fractal parameters s and $\bar{r}$, which is henceforth adopted. Returning to (5) and noting that the quantization level $\bar{r}_k$ depends only on the range block R and is independent of the domain block D, the full search problem is now reduced to

$$\min_{D\subset\Omega} \tilde{E}(R,D) = \min_{D\subset\Omega} \min_{s_i} \left\|s_i\,(D-\bar{d}I) - (R-\bar{r}_k I)\right\|^2 = \min_{s_i} s_i^2 \left(\min_{D\subset\Omega} \left\|(D-\bar{d}I) - \frac{(R-\bar{r}_k I)}{s_i}\right\|^2 \right). \tag{8}$$

Equation (8) means that for each fixed quantized level for the scaling s_i, the best matching domain block can be found by searching for

$$err = \min_{D\subset\Omega} \left\|(D-\bar{d}I) - \frac{(R-\bar{r}_k I)}{s_i}\right\|^2 \tag{9}$$

which can also be converted to a nearest neighbor search similar to the steps outlined in Section II. The overall best matching block is then obtained by minimizing the weighted err over the quantized levels for s_i. Thus, (8) leads to the following new algorithm.

```
Build a kd-tree to store the normalized
 domains $\{D' : D' = D - \bar{d}I,\ D \in \Omega\}$;
FOR each range block $R' = R - \bar{r}I$,
 set min_err = infinity;
 FOR each $s_i$
  Search the tree for the $D$ such that
  $|D' - R'/s_i|^2$ is minimum (by nearest neighbor
  search). Return the corresponding domain
  $D$ and err $= |D' - R'/s_i|^2$;
  err = err * $s_i^2$;
   IF err < min_err
    min_err = err;   min_D = D;   min_si = $s_i$;
   ENDIF
 END
END
```

The new algorithm returns the optimal domain block D and its associated quantized scaling coefficient s_i, which minimizes the value of $\tilde{E}(R,D)$.

Although the running times for each range block is roughly proportional to the number of quantization levels for s, the search time is still comparable to that of Saupe's algorithm. This is because in order to compensate for the effects of quantization error, Saupe's algorithm searched for, say, the top five to ten closest neighbors. From [17], we know that the running time of the algorithm for each query is $O(k \log N_D)$, where k is the number of nearest neighbors to be returned. On the other hand, a small number (usually four or eight) of quantization levels of s are often sufficient to give good results [19], the actual running times of the two algorithms should be comparable.

B. Saving Memory Space

The problem of large memory requirement for the data structure in conventional nearest neighbors search approach is addressed in the implementation of our new algorithm, borrowing a useful programming trick from Fisher [12]. Before running the new algorithm, the image is first pre-decimated into four lower resolution subimages. Each pixel in the first subimage is the average value of four neighboring pixels in a nonoverlapping 2×2 block starting from position (0, 0) in the original image. The other three subimages are generated in a similar manner, using nonoverlapping 2×2 blocks starting from positions (0, 1), (1, 0) and (1, 1) of the original image respectively. An $N \times N$ subblock in any of these subimages is then just the pre-decimated version of a $2N \times 2N$ domain block in the original image. By doing so, we save a lot of time otherwise used for repeated spatial contraction of the domain blocks. Moreover, this trick allows us to save memory in the kd-tree construction.

For the kd-tree, data points are stored in the leave nodes. In our implementation, instead of storing the coordinates of each D' in a leave node, we store only the identity of the subimage that contains the corresponding domain block D and its position in the subimage, as well as the value of $\bar{d}$. We extract the pre-decimated domain block D from its subimage and calculate D' only when the leave node is encountered in a search. In this way the memory requirement is substantially reduced. In Section II-B, we have already noted that the space for a leave node is 258 bytes if coordinates of D' are stored. On the other hand, our scheme only takes six bytes (two bytes for position and four bytes for $\bar{d}$). Each internal node stores a pointer to each of the left and right children and several other fields for recording the space splitting parameters. Roughly it takes about 20 bytes for an internal node. As the kd-tree is a balanced binary tree, the number of internal nodes is almost the same as the number of leave nodes. Hence, the memory saving is

$$1 - \frac{(20+6)}{(20+258)} \approx 90\%.$$

The memory saving allows the processing of a larger image or allows a larger domain pool to be used in the search. The tradeoff for the memory saving is the extra processing time of data points in leave nodes. The overhead in the processing time will not be too significant because the computation of each data point involves only an additional $N \times N$ floating point *minus* operations, which should be relatively small in comparison with other operations in the search (e.g., calculating Euclidean distance, which accounts for most of the running time).

C. Approximate Nearest Neighbor Search

It is known that for a d-dimensional key, when 2^d is comparable to the number of keys in the kd-tree, the algorithm works poorly (for a 4×4 range block, d equals 16). Arya *et al.* [17] partially solved the problem by relaxing the requirement of finding the nearest neighbor to that of finding an *approximate* nearest neighbor. The method introduces a search parameter, ε, to the kd-tree query, which is the maximum allowable relative error of the approximate nearest distance that will be found to that of the true minimum distance. That is, the distance of the query point to the approximate nearest neighbor is

$$\tilde{d} \le (1+\varepsilon)d$$

where d is the distance to the true nearest neighbor.

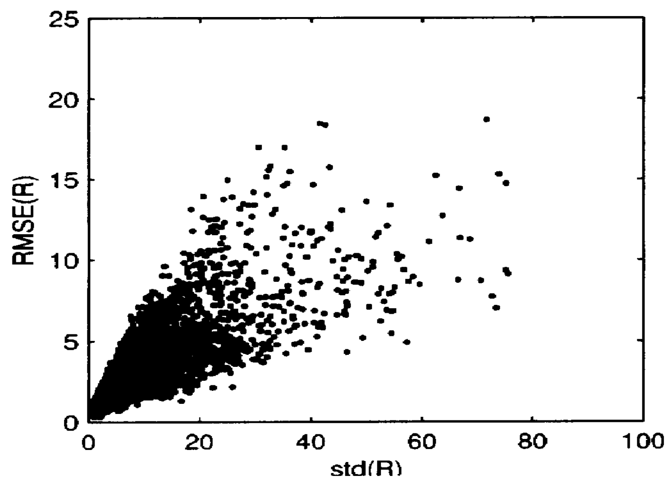

Fig. 1. Scatter plot of the RMSE versus standard deviation of range blocks for Lena512.

Arya *et al.* reported that empirically a seemingly large ε, say three, returns the true nearest neighbor about 50% of the times and the average relative error is within 10% of the true minimum distance, while the searching time is reduced by orders of magnitude [17]. As with Saupe, we have incorporated this enhancement to our new algorithm and gained significant improvement in speed at negligible degradation of search quality. As a further enhancement to the algorithm, we have derived an adaptive method to determine the best value of ε that should be used for each range block. The adaptive epsilon scheme and the incorporation of quadtree encoding into our new algorithm are discussed in the next section.

IV. Adaptive Scheme

A. Motivation

Let $\text{RMSE}(R) = \min_{D\in\Omega} \sqrt{\tilde{E}(R,D)/N^2}$, where N is the dimension of R and D. This quantity is the root-mean-square error between the range block and the best matching transformed domain block. There is theoretical and empirical evidence that RMSE(R) of a range block is positively correlated to the standard deviation of the range block [19]. Fig. 1 shows a typical scatter plot of the RMSE versus the standard deviation [denoted by std(R)] for the ranges blocks in the Lena image.

The plot shows that range blocks with smaller standard deviation usually have smaller RMSE. Since ε is the upper bound of the relative error in approximate nearest neighbor search, the results suggest that range blocks with smaller standard deviation can *tolerate* a larger value of ε. Thus, we propose to adapt the value of ε for each range block based on its standard deviation. It is anticipated that such an adaptive-ε scheme can speed up the search while maintaining the image quality.

B. Error Modeling

In the last section, we argue why the value of ε should be a function of the standard deviation of the range block. For each range block, there are two aspects of the encoding that are affected by the value of ε: i) time to find the best matching domain by approximate nearest neighbor search and ii) the matching

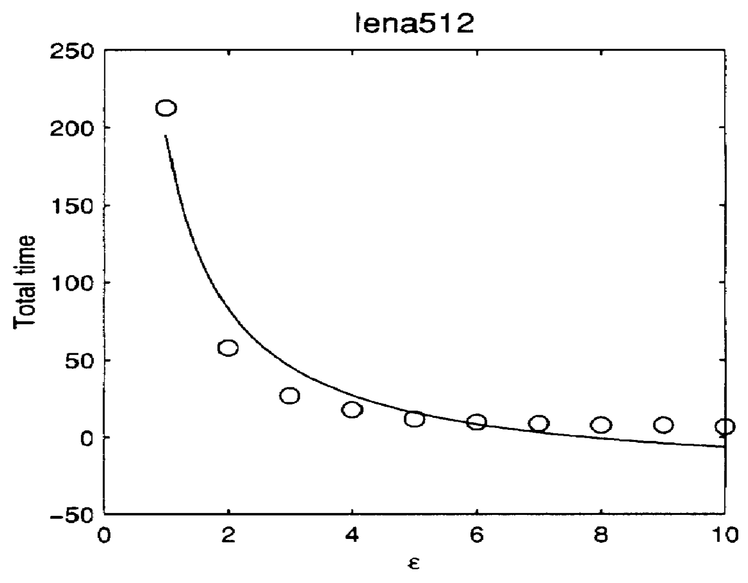

Fig. 2. Total running time versus epsilon.

error, ARMSE(R, ε), between the range block and the domain block that is found by the approximate nearest neighbor search

$$\text{ARMSE}(R,\varepsilon) = \sqrt{\frac{\tilde{E}(R,\tilde{D})}{N^2}}.$$

Since the search is approximate, this error depends on the value of ε and is in general larger than RMSE(R).

Our criteria to construct ε as a function of the standard deviation of range block is to minimize the average matching error

$$\operatorname*{Min}_{\varepsilon} T_e(\varepsilon) = \int_0^\infty \mathbf{E}\left[\text{ARMSE}(R,\varepsilon)|X(R) = x\right] p(x)dx \tag{10}$$

subject to the constraint on average encoding time

$$T_t(\varepsilon) = \int_0^\infty t(\varepsilon)p(x)dx = t_0 \tag{11}$$

where $p(x)$ is the probability density function for the standard deviation, $X(R) = x$, of the range blocks; $t(\varepsilon)$ is the searching time using the approximate nearest neighbor search with the corresponding ε parameter; and $\mathbf{E}[.]$ is the expectation operator. Note that RMSE(R), ARMSE(R, ε) and $X(R)$ are here treated as random variables with the sample space being the set of range blocks R over a certain class of images to be specified later.

In order to solve the above optimization problem, we have to find accurate models for $t(\varepsilon)$ and $\mathbf{E}[\text{ARMSE}(R,\varepsilon)|X(R) = x]$. These models should also be simple enough to facilitate analysis. To find $t(\varepsilon)$, we fixed the value of ε, ran our approximate nearest neighbor search algorithm for all the range blocks and measured the aggregate running time. The experiment was repeated for values of ε ranging from one to ten. The total running time is plotted against the value of ε for the Lena image in Fig. 2.

A curve of the form k/ε is fitted to the data points using regression analysis. It can be seen from Fig. 2 that this simple model fits the data point fairly well. Hence, the following equation is used to model $t(\varepsilon)$:

$$t(\varepsilon) = \frac{k_1}{\varepsilon}. \tag{12}$$

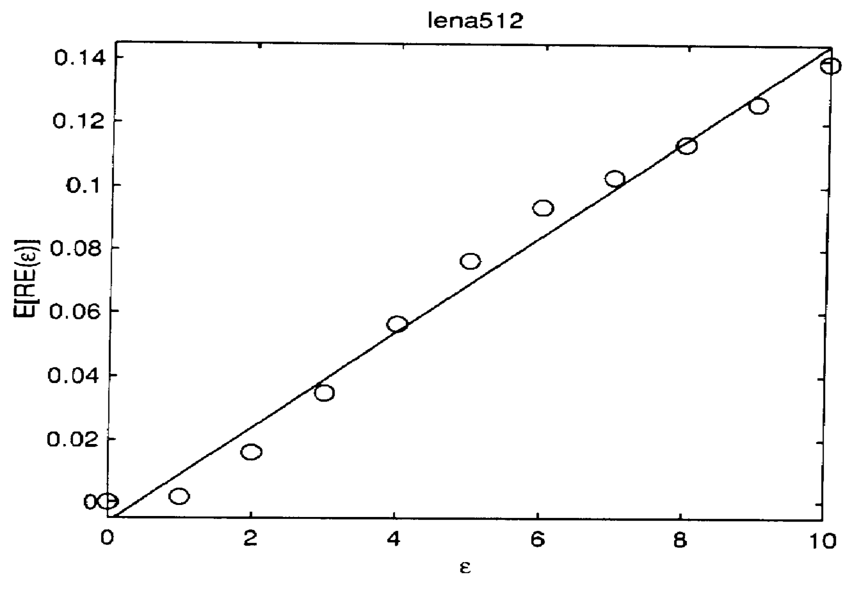

Fig. 3. Mean of RE(ε) versus ε.

The modeling of $\mathbf{E}[\text{ARMSE}(R,\varepsilon)|X(R)=x]$ requires more work as the quantity depends on the approximate searching algorithm as well as on the fractal transform. To isolate the effects of the searching algorithm, we first consider the quantity

$$\text{RE}(R,\varepsilon)=\frac{\text{ARMSE}(R,\varepsilon)}{\text{RMSE}(R)}-1$$

which measures the relative error of the approximate nearest neighbor search and hence should only depend on the search algorithm. For each fixed value of ε, we ran our approximate nearest neighbor search algorithm and calculated the mean value of the relative error for all the range blocks, which should be a good estimate for $\mathbf{E}[RE(\varepsilon)]$. The experiment was repeated for values of ε ranging from 1 to 10 for the Lena image. The plot of the mean value of $RE(R,\varepsilon)$ against ε is given in Fig. 3, which showed that a linear regression model fits the data quite well. This is consistent with the findings of Arya *et al.* [17]. We thus use the following equation to model $\mathbf{E}[RE(R,\varepsilon)]$

$$\mathbf{E}[RE(R,\varepsilon)]=k_2\varepsilon. \tag{13}$$

Next, we need to consider the collage error RMSE(R) and how it varies with the standard deviation of the range blocks R. The plot in Fig. 1 clearly showed that the RMSE is tightly bounded by linear functions of the standard deviation of the range blocks. This suggests (and is confirmed by empirical plots of many test images) that the mean value of RMSE can be modeled by the linear model

TABLE I
CORRELATIONS FOR THE LENA IMAGE

	ε = 5	ε = 10
ρ(RE(ε), X)	-0.1078	-0.0478
ρ(RE(ε), RMSE)	-0.0987	-0.0406

$$\mathbf{E}[\text{RMSE}(R)|X(R)=x]=k_3x. \tag{14}$$

Lastly, we further assume that

$$\text{RE}(R,\varepsilon) \text{ is independent of RMSE}(R) \text{ and } X(R). \tag{15}$$

Assumption (16) is intuitively plausible because the relative error made by approximate nearest neighbor search should only depend on the search method and its search parameter ε. There are no obvious reasons why this should have anything to do with the position of the query point in the Euclidean space or the distance to the true nearest neighbor. Nevertheless, some evidence of the validity of this assumption is given in the following.

Given two independent random variables X and Y, their correlation coefficient $\rho(X,Y)$ should be zero. Although the converse is in general not true, this serves as an evidence for the independence between the two random variables. The estimates of these correlation coefficients are calculated from the sample of range blocks taken from the Lena image and are tabulated in Table I. As the correlation coefficients are quite low, they provide statistical support for assumption (15).

We now derive an expression for $\mathbf{E}[\text{ARMSE}(R,\varepsilon)|X(R)=x]$ [see the equation at the bottom of the page].

C. Optimal Search Parameter

We first consider the constant ε case. Let ε_0 be the solution of the problem. From the constraint (11) and model (12)

$$T_t(\varepsilon_0)=t_0 \Rightarrow \frac{k_1}{\varepsilon_0}=t_0 \Rightarrow \varepsilon_0=\frac{k_1}{t_0}. \tag{16}$$

Now, let the search parameter $\varepsilon=\varepsilon(x)$ be a function of the standard deviation, x, of range blocks. The problem can be solved by the method of Lagrangian multiplier and calculus of variations. The constrained optimization problem (10)–(11) becomes

$$\min_{\varepsilon(x)}\int_0^{\infty}(\mathbf{E}[\text{ARMSE}(\varepsilon,x)|X(R)=x] \\ +\lambda t(\varepsilon))p(x)dx$$

$$\begin{aligned}\mathbf{E}[\text{ARMSE}(\varepsilon)|X(R)=x] &=\mathbf{E}\left[\frac{\text{ARMSE}(R,\varepsilon)}{\text{RMSE}(R)}\text{RMSE}(R)|X(R)=x\right]\\ &=\mathbf{E}[(\text{RE}(\varepsilon)+1)\text{RMSE}(R)|X(R)=x]\\ &=\mathbf{E}[(\text{RE}(\varepsilon)+1)]\mathbf{E}[\text{RMSE}(R)|X(R)=x] && \text{(by assumption (15))}\\ &=(1+k_2\varepsilon)k_3x && \text{(by models (13)–(14))}\end{aligned}$$

where λ is the Lagrangian multiplier. The solution to this optimization satisfies the Euler-Lagrange equation:

$$\frac{\partial g}{\partial \varepsilon} - \frac{d}{dx}\left(\frac{\partial g}{\partial \varepsilon'}\right) = 0$$

where ε' denotes $d\varepsilon/dx$ and

$$\begin{aligned} g(x,\varepsilon) =& (\mathbf{E}[\text{ARMSE}(\varepsilon,x)|X(R)=x] + \lambda t(\varepsilon))p(x) \\ =& \left((1+k_2\varepsilon)k_3x + \frac{\lambda k_1}{\varepsilon}\right)p(x) \\ & \text{(from models in Section IV-B).} \end{aligned}$$

Since g is independent of ε', the Euler-Lagrange equation reduces to

$$\begin{aligned} \frac{\partial g}{\partial \varepsilon} =& 0 \Rightarrow \left(k_2k_3x - \frac{\lambda k_1}{\varepsilon^2}\right)p(x) \\ =& 0 \Rightarrow \varepsilon(x) = \sqrt{\frac{\lambda k_1}{k_2k_3}}\sqrt{\frac{1}{x}}. \end{aligned} \tag{17}$$

Substituting the expression of $\varepsilon(x)$ into the constraint equation $\mathrm{T}_t(\varepsilon) = t_0$, we solve for λ

$$\begin{aligned} \int_0^\infty \frac{k_1}{\varepsilon}p(x)dx =& t_0 \Rightarrow \int_0^\infty k_1\sqrt{\frac{k_2k_3}{\lambda k_1}}\sqrt{x}p(x)dx \\ =& t_0 \Rightarrow \sqrt{\frac{k_1k_2k_3}{\lambda}}\mathbf{E}\left[\sqrt{X}\right] \\ =& t_0 \Rightarrow \lambda = k_1k_2k_3\left(\frac{\mathbf{E}\left[\sqrt{X}\right]}{t_0}\right)^2. \end{aligned}$$

Substituting the result back into the expression of $\varepsilon(x)$ in (17) and noting that $\varepsilon_0 = k_1/t_0$ for the case of constant ε, we finally obtain

$$\varepsilon(x) = \varepsilon_0\frac{\mathbf{E}\left[\sqrt{X}\right]}{\sqrt{x}}. \tag{18}$$

This adaptive scheme for the search parameter ε is optimal for the class of images for which our models and assumptions in Section IV-B hold.

D. Quadtree Partitioning

To improve compression ratio while maintaining the decompression quality, we incorporate the well-known technique of quadtree partitioning [12], allowing up to n_q number of quadtree levels. Hence there are n_q domain pools for the different range block sizes. Although there are several new split-decision functions (see [20] and references therein) that can speed up the conventional quadtree encoding, we shall not consider these so that we can concentrate on studying the effect of our adaptive approximate search. We thus adopted the usual split-decision function based on the collage error and split when ARMSE(R, ε) is larger than a given tolerance T_0, or when the highest quadtree level is reached (i.e., when the smallest allowable range-block size is reached).

It was mentioned in Section III-C that the kd-tree algorithm does not work well for points in high dimensional space. Thus in our quadtree implementation, we will run into problem at low quadtree levels (i.e., large range-block sizes). To cope with the problem, domain blocks which are larger than the smallest allowable size are down-filtered to the smallest allowable size before inserting into the kd-trees. The down-filtering is performed by the averaging of 2×2 blocks of neighboring pixels. Range blocks which are larger than the smallest allowable size are also down-filtered before querying for its nearest neighbor in the kd-trees. Recall that in order to save memory space we do not store the coordinates of transformed domain blocks in the kd-tree. Instead, we store only the identity of the subimage which contains the domain and its position in the subimage. The down-filtering of domain blocks is processed by a pre-decimation scheme similar to that described in Section III-B.

V. Experimental Results and Discussion

A. Implementation Details

All the results in this paper were obtained from experiments performed on a PC with an Intel Pentium II 450 MHz CPU and 128 MB memory. We tested our algorithms on six well-known 512×512 gray level images, namely, Lena, Baboon, Peppers, Bridge, Gold Hill, and Couple obtained from the City University IPL image database (http://www.image.cityu.edu.hk/imagedb/). However, only results for Lena are shown.

The range block mean is quantized uniformly between zero and 255. The scaling coefficient was restricted to values between zero and one and was quantized uniformly with quantization levels equal to $i/2^{b_s}$ for $i = 1$ to 2^{b_s}, where b_s is the number of bits for s. A total of nine quantization bits for the transformation parameters were used and were allocated by the optimal allocation scheme [19] referred to in Section III: the scaling parameter was allocated two bits and the parameter $\{\bar{r}_k\}$ was allocated seven bits. The smallest allowable range block size is 4×4. Additional results were generated using a total of ten quantization bits for Section III-B below; the optimal allocation scheme then assigns three bits for the scaling in our new algorithm.

B. Performance of the New Algorithm

First of all we examine the performance of our new algorithm with constant search parameter ε_0 and no quadtree (or equivalently, $n_q = 1$). Fig. 4 summarizes the PSNR at different running times between our new algorithm and Saupe's. Since Saupe uses the conventional scaling and luminance offset parameters, we cannot apply our optimal bit-allocation scheme as given in [19]. Thus we tried out various bit-allocation schemes as summarized in Fig. 4(a). Note that there is a wide variation of performance between the different bit allocations. The optimal allocation of five bits (i.e., 32 levels) for scaling and four bits for the luminance offset is used for comparison with our algorithm [see Fig. 4(b)]. Similarly, we repeated the calculations using a ten-bit quantizer: Fig. 4(c) shows the results of the different bit-allocation schemes for Saupe's method and the best allocation scheme is used for comparison with our algorithm, as shown in Fig. 4(d).

Each curve in Fig. 4(b) and (d) may be considered as a parametric curve with parameter ε_0. The point $\varepsilon_0 = 1$ starts at the far right of each curve and moves toward the left. As the value of parameter ε_0 increases, the encoding time drops substantially

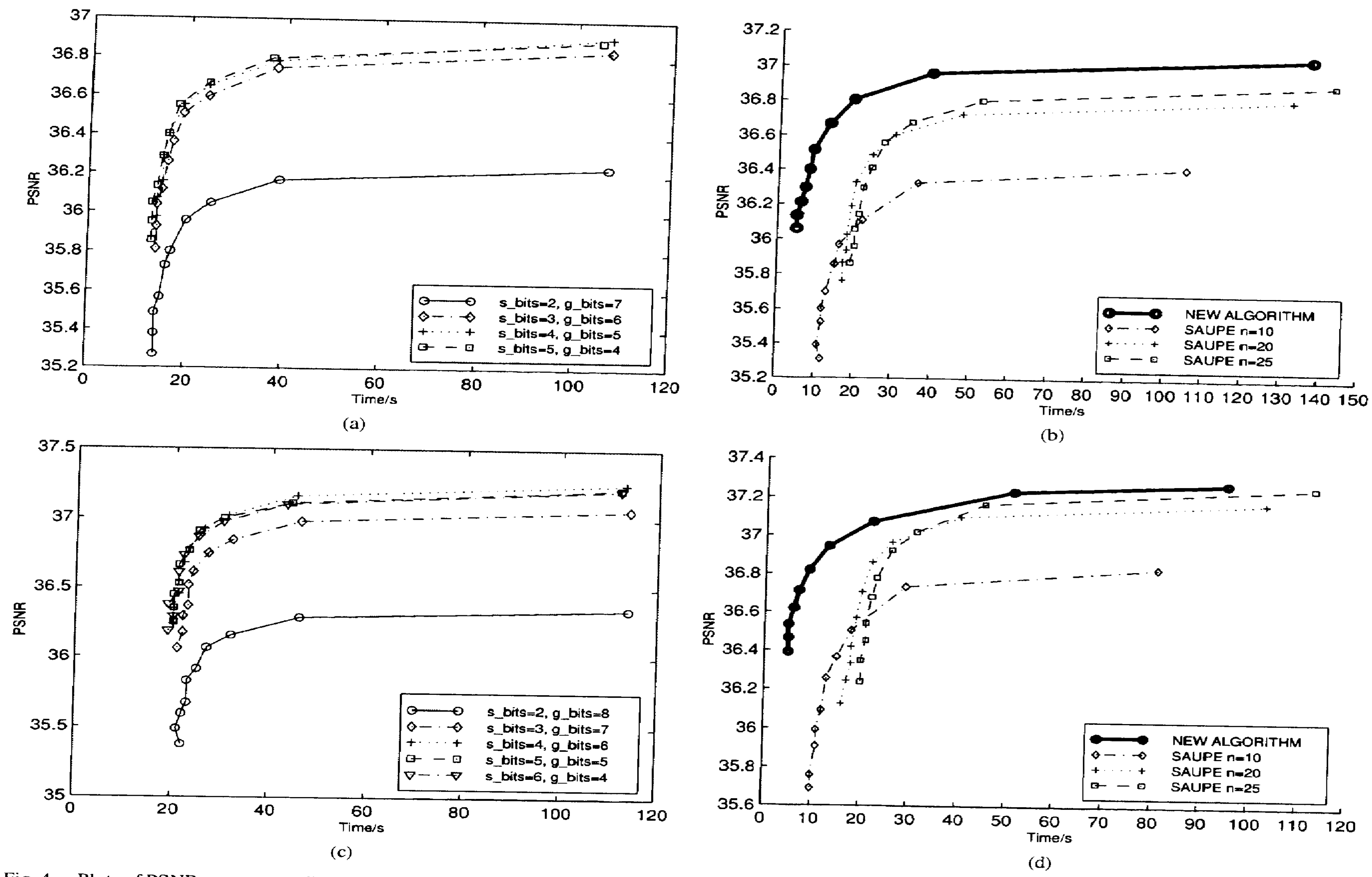

Fig. 4. Plots of PSNR versus encoding times nine-bit quantization: (a) bit-allocation schemes for Saupe's method, (b) comparison with our new algorithm; ten-bit quantization, (c) bit-allocation schemes for Saupe's method, and (d) comparison with our new algorithm.

while the PSNR decreases slowly at first but gradually the effect on PSNR becomes more marked at large ε_0, forming an upside down L-curve. Consider the top curve in Fig. 4(b) that represents the results of our new algorithm in the nine-bit quantization case. When $\varepsilon_0 = 3$, the encoding time is 21 s while the PSNR is 36.81. For comparison, the linear brute-force full search required 1717 s and the PSNR achieved is 37.08 dB. This represented a speed-up factor of over 80 at the expense of a slight drop of PSNR of 0.27 dB. When ε_0 is increased to nine, the speed up is more than 280 times while the PSNR decreased by less than 1 dB only. This shows that the performance of the our algorithm is excellent compared to brute-force linear search.

For Saupe's algorithm, there is one more parameter that is adjustable, namely the top n nearest neighbors returned by the approximate nearest neighbor search (see Section II-B). Each value of n traces a separate curve. From the curves in Fig. 4(b) and (d), it can be seen that a smaller n is favorable for small running time, while a larger n is favorable for higher PSNR.

Comparing the results with Saupe's, our new algorithm outperforms the conventional algorithm, regardless of the value of n. At small running times, our algorithm performs especially well and obtains a PSNR which is more than 1 dB better than the conventional algorithm. Moreover, our new algorithm can meet very high decompression requirement (e.g., 37 dB in the nine-bit quantization case), which the conventional algorithm cannot meet. Similar results were obtained for other test images.

C. *Effects of Adaptive Epsilon* ε

In this section, we investigate whether the use of adaptive epsilon could further enhance the new algorithm. Fig. 5 shows the plots of the PSNR against running time for our algorithm using both constant ε_0 and adaptive epsilon as given by (18).

From the figure, it can be seen that the adaptive epsilon scheme is able to give a better PSNR with the same running time, especially at small running times. The gain in PSNR roughly ranges from 0.2 to 0.4 dB. The improvement is much more impressive if we consider the gain in encoding time at a fixed PSNR level. For example, at PSNR of 36.6 dB, the adaptive epsilon scheme required 7 s versus 13 s for the constant epsilon scheme, a speed up factor of almost two. The speed up factor continues to increase as the PSNR increases.

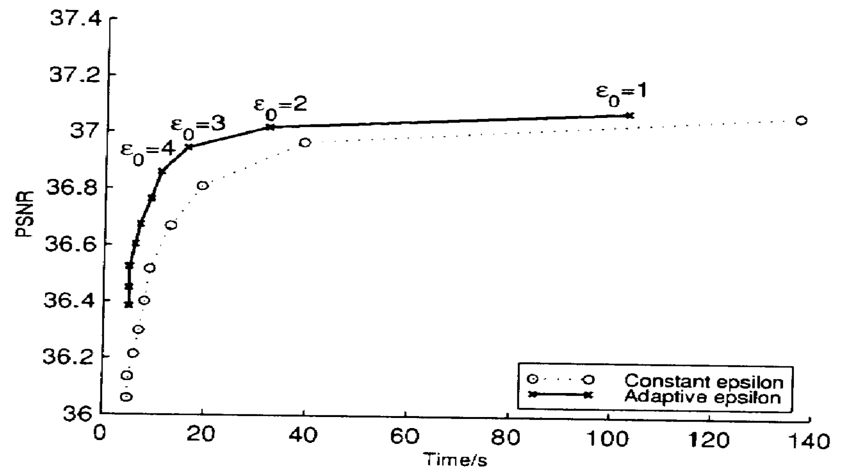

Fig. 5. Comparion of PSNR-time curves between using constant and adaptive epsilon.

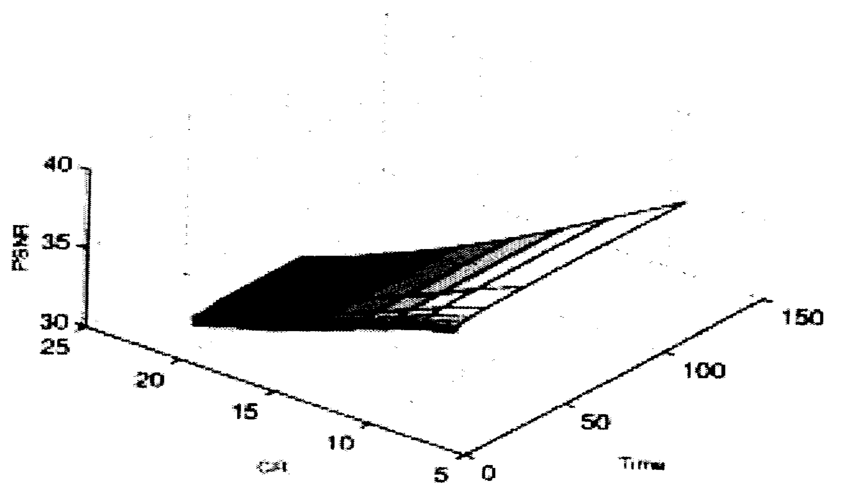

Fig. 6. Relationship between PSNR, CR, and encoding time for a fixed value of n_q.

At PSNR of 37 dB, the adaptive epsilon scheme required 29 s versus 73 s for the constant epsilon scheme, the speed up factor reaches 2.5.

Furthermore, in the adaptive scheme, the coding performance is more robust to the choice of ε_0, as reflected by the fact that the points for the adaptive epsilon case were more clustered on the curve than the respective points on the constant epsilon case. Specifically, for the constant epsilon case, the ten values of $\varepsilon_0 = 1$ to 10 lead to wide variation of performance: the PSNR ranged from 36 dB to 37 dB and the encoding time ranged from 7 s to 138 s. By comparison, the adaptive scheme was much more robust, with the PSNR ranging from 36.4 dB to just over 37 dB and the encoding time ranging from 7 s to 102 s. Other test images exhibited similar behavior. Nevertheless, we discuss how an appropriate default value of ε_0 can be determined in the next subsection.

D. Default Value of ε_0

In our approximate nearest neighbor search algorithm, different values of ε_0 give rise to different running times and PSNR. Thus ε_0 serves as an adjustable parameter to balance the tradeoff between encoding time and fidelity. We would like to find an optimal default value of ε_0 that users can start with. We borrow the idea of the method of *L-curve*, which is commonly used in solving ill-posed least square problems. The L shape of the PSNR versus encoding time curve suggests that the turning point gives the best tradeoff since at the other two extremes, the curve flattens and hence changing the parameter cannot give appreciable improvement in performance. This turning point is optimal in the sense that small deviation from this parameter lead to sharp deterioration of one aspect of performance while the other aspect of performance does not have appreciable improvement.

Referring back to Fig. 5 and focusing on the top curve (corresponding to the adaptive epsilon case), we see an (upside down) L-curve for the PSNR versus encoding time plot. The corner of the L-curve corresponds to $\varepsilon_0 = 3$ to $\varepsilon_0 = 4$. Slightly larger ε_0 leads to sharp drop of PSNR while encoding time only decreases slightly. Similarly, slightly smaller ε_0 increases the encoding time enormously while the PSNR only marginally increases. Hence, the default value of ε_0 should be set to a value around three to four. Results from the other test images also suggest a similar default value. This is consistent with the findings by Arya *et al.* for a wide range of problems [17].

TABLE II
RELATIONSHIP BETWEEN THE INPUT PARAMETERS AND DIFFERENT ASPECTS OF THE PERFORMANCE

Parameter/Performance	t	PSNR	CR
ε_0	large -ve	large -ve	very small -ve
n_q	large +ve	large -ve	large +ve
T_0	small -ve	large -ve	large +ve

E. Results of Quadtree Partitioning

In this section, we discuss the results of incorporating quadtree encoding into our new algorithm. In addition to the ε_0 parameter, we now have two more, namely i) the number of quadtree levels n_q and ii) the split-decision function tolerance T_0. The results are also three-dimensional (3-D), namely i) encoding time t, ii) compression ratio CR, and iii) PSNR.

The values of n_q are discrete and in the experiments four values of n_q were used: 1, 2, 3, and 4. If we fix the value of n_q and vary the values of ε_0 and T_0, the set of points (t, CR, PSNR) will form a surface in the 3-D space as shown in Fig. 6. For each value of n_q there is one such surface and the surfaces of different n_q may cut one another. We would like to investigate how the input parameters affect the performance of the encoder and hence find a heuristic for choosing the default values for these parameters.

From experimental results on Lena, we summarized the effect of each parameter on the three aspects of coding performance in Table II. Each cell describes the significance and the sign of correlation between a particular aspect of performance and an input parameter (keeping the other two parameters fixed). These results are intuitively plausible and were found to be true for the other test images.

Note that ε_0 has a very small effect on CR. Therefore, if we fix the value of n_q and T_0 and varies ε_0 only, we will be able

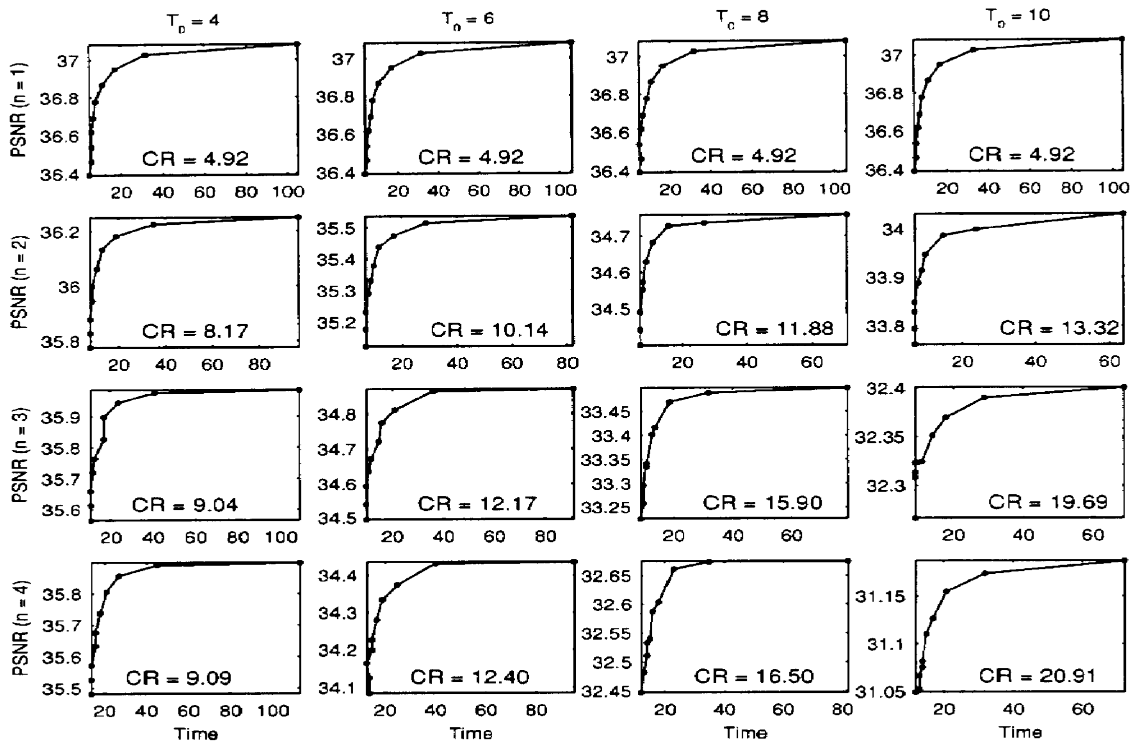

Fig. 7. PSNR versus compression ratio (CR) versus encoding time for different number of quadtree levels (n) and tolerance T_0. The average CR is given in each subplot.

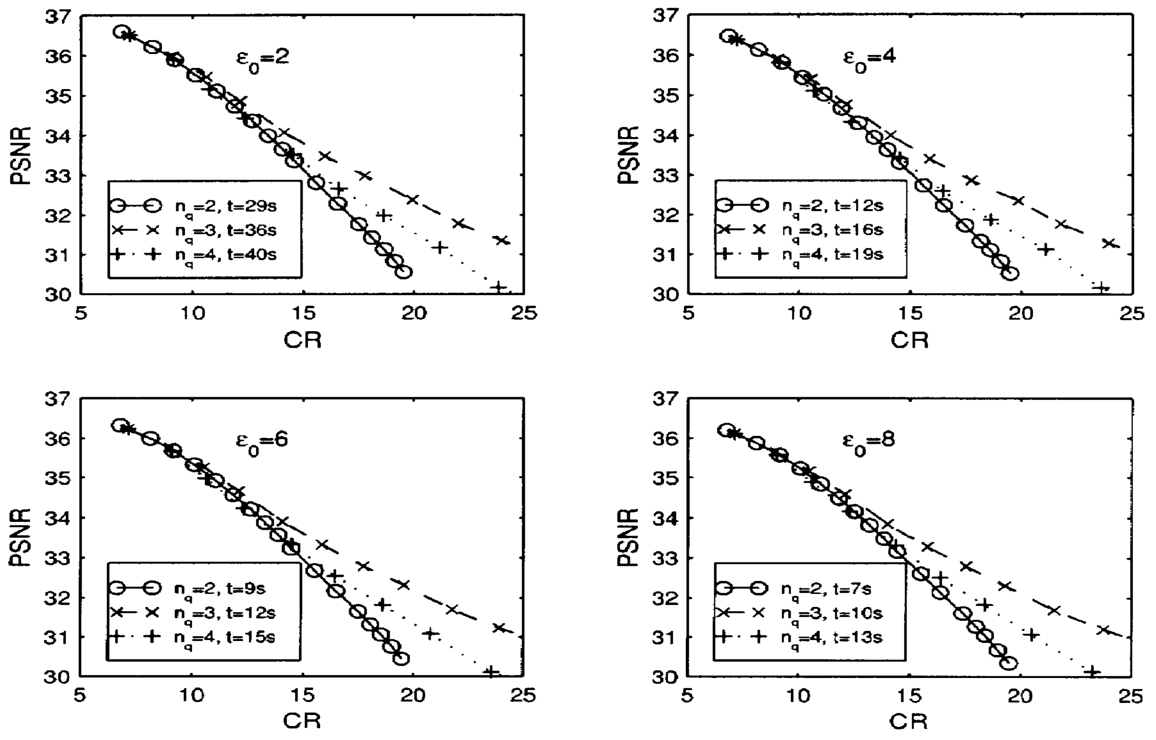

Fig. 8. Rate distortion curves for different number of quadtree levels n_q and ε_0. The average encoding time is shown in each subplot.

to trace the graph of PSNR against encoding time for a constant CR. Geometrically, it is a cross section of the surface shown in Fig. 6 when viewed along the CR-axis. The PSNR-encoding time graphs for different values of n_q and T_0 are shown in Fig. 7. These PSNR-encoding time graphs look similar to those when no quadtree partitioning is used. By the L-curve argument of Section V-D, we should choose the default value of ε_0 to be around three and four.

Similarly, because T_0 has little influence on the encoding time, if we fix the value of ε_0 and n_q and varies T_0 only, we will trace the graph of PSNR against CR for a constant encoding time. It is a cross section of the surface in Fig. 6 when viewed

TABLE III
COMPARISON WITH SAUPE'S RESULTS

Images	Saupe's Algorithm			Our Algorithm		
	PSNR	CR	Adjusted Time(s)	PSNR	CR	Time(s)
Lena 512	35.80	8.31	60	35.83	9.04	17
	35.40	8.39	60	35.40	10.55	21
	34.57	8.45	39	34.57	11.87	8
Baboon 512	26.69	4.16	168	26.73	5.21	25
	26.13	4.43	90	26.14	5.87	10
	25.19	4.75	60	25.82	6.03	8

along the t-axis. Such rate distortion curves for different values of ε_0 and n_q are shown in Fig. 8. Comparing the three curves of any subplot in Fig. 8, we found that the rate distortion curve is best when $n_q = 3$. Results for the other test images were similar, with the exception of the Bridge image, for which $n_q = 2$ is slightly better.

Finally, in Table III, we compared our results to those of the conventional nearest neighbor search approach that were reported in the original paper by Saupe [16]. It should be noted that the results on running times reported in Saupe's paper were measured on a SGI Indigo2 running an R4000 processor, a machine that is about three times as fast as our PC. Thus for fair comparison, we adjusted Saupe's times by a factor of three to compare with our times. It can be seen from the table that our new algorithm is able to achieve better compression ratios for the same or better PSNR and, moreover, were achieved at a much shorter running time.

VI. CONCLUSIONS

In this paper, we have examined Saupe's nearest neighbor search approach of fractal image compression and sought to improve it by reformulating the search approach in the discretized parameter space. We have adopted an alternative parameterization of the affine transformation and have shown how it could be combined with approximate nearest neighbor search to yield a superior algorithm. Experiments showed that our algorithm is able to obtain a better reconstruction PSNR with smaller encoding time. We have also described a simple modification to the data structure that can significantly reduce the memory requirement of storing the search tree.

Furthermore, we have proposed an adaptive approximate nearest neighbor search scheme and derived an optimal formula to determine the value of the search parameter epsilon ε that should be used in the approximate nearest neighbor query for each range block. Empirical results have confirmed that the adaptive scheme was able to yield further improvement to the algorithm.

We have also incorporated the quadtree partitioning to our new algorithm so that the compression ratio can be adjusted, with the reconstruction PSNR as a tradeoff. Results showed that our new algorithm leads to better rate distortion curves than the conventional nearest neighbor search. Moreover, our superior results were achieved at much better encoding times.

Lastly, many existing time complexity reduction techniques, such as the classification of range and domain blocks by Fisher [12] and domain pool reduction schemes (see survey in [1]), can be incorporated into our algorithm. Other interesting and useful future enhancements to our work include using different partitioning schemes and entropy encoding for the fractal codes to further improve the compression ratios.

REFERENCES

[1] B. Wohlberg and G. Jager, "A review of the fractal image coding literature," *IEEE Trans. Image Processing*, vol. 8, pp. 1716–1729, Dec. 1999.
[2] D. C. Popescu and H. Yan, "MR image compression using iterated function systems," *Magn. Res. Imag.*, vol. 11, pp. 727–732, 1993.
[3] G. E. Øien and G. Narstad, "Fractal compression of ECG signals," in *Proc. NATO ASI Fractal Image Encoding and Analysis*, Y. Fisher, Ed., Trondheim, Norway, July 1995.
[4] C. M. Wood, "General data compression algorithm for space images using fractal techniques," *Proc. SPIE*, vol. 2198, pp. 1336–1341, 1994.
[5] B. Schouten and P. de Zeeuw, "Feature extraction using fractal codes," in *Proc. Visual 99, Visual Information and Information Systems, 3rd Int. Conf.*, 1999, pp. 483–492.
[6] J. Puate and F. Jordan, "Using fractal compression scheme to embed a digital signature into an image," *Proc. SPIE*, vol. 2915, pp. 108–118, 1997.
[7] H. A. Cohen, "Thumbnail-based image coding utilizing the fractal transform," in *Proc. ICIP-96 IEEE Int. Conf. Image Processing*, Lausanne, Switzerland, Sept. 1996.
[8] J. Marie-Julie and H. Essafi, "Using the fractal transform for image and indexing and retrieval by content," in *Proc. Conf. Fractals Engineering*, June 1997.
[9] M. Ancis, W. Buchwald, P. Giusto, D. Daniele, and D. Schmidt, "Fractal zooming of thumbnails for progressive image coding," *Proc. SPIE*, vol. 3527, pp. 541–549, 1998.
[10] T. Tan and H. Yan, "Object recognition using fractal neighbor distance: Eventual convergence and recognition rates," in *Proc. ICPR2000*, vol. 2, 2000, pp. 785–788.
[11] L. M. Kaplan and C.-C. J. Kuo, "Texture segmentation via haar fractal feature estimation," *J. Vis. Commun. Image Represent.*, vol. 6, no. 4, pp. 387–400, 1995.
[12] Y. Fisher, *Fractal Image Compression: Theory and Applications*. New York: Springer-Verlag, 1995.
[13] L. M. Po and C. K. Chan, "Adaptive dimensionality reduction techniques for tree-structured vector quantization," *IEEE Trans. Commun.*, vol. 42, pp. 2246–2257, June 1994.
[14] G. Caso, P. Obrador, and C.-C. J. Kuo, "Fast methods for fractal image encoding," *Proc. SPIE*, vol. 2501, pp. 583–594, 1995.
[15] B. Bani-Eqbal, "Enhancing the speed of fractal image compression," *Opt. Eng.*, vol. 34, no. 6, pp. 1705–1710, June 1995.
[16] D. Saupe, "Fractal image compression via nearest neighbor search," in *Conf. Proc. NATO ASI Fractal Image Encoding and Analysis*, Trondheim, Norway, July 1995.
[17] S. Arya, D. Mount, N. Netanyah, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching in fixed dimensions," in *Proc. 5th Annu. ACM-SIAM Symp. Discrete Algorithms*, 1994, pp. 573–582.
[18] G. E. Øien and S. Lepsøy, "A class of fractal image coders with fast decoder convergence," in *Fractal Image Compression: Theory and Applications*. New York: Springer-Verlag, 1995, pp. 153–174.
[19] C. S. Tong and M. Pi, "Fast fractal image encoding based on adaptive search," *IEEE Trans. Image Processing*, vol. 10, pp. 1269–1277, Sept. 2001.
[20] R. Distasi, M. Polvere, and M. Nappi, "Split decision functions in fractal image coding," *Electron. Lett.*, vol. 34, no. 8, pp. 751–753, 1998.

Chong Sze Tong (M'99) received the B.A. degree in mathematics and the M.A. and a Ph.D. degrees all from Cambridge University, Cambridge, U.K.

After graduation, he joined the Signal and Image Processing Division of GEC-Marconi's Hirst Research Centre as a Research Scientist, working on image restoration and fractal image compression. He joined the Department of Mathematics at Hong Kong Baptist University in 1992.

Dr. Tong is a Fellow of the Institute of Mathematics and its Application and a Chartered Mathematician. His current research interests include image processing, fractal image compression, image segmentation, and neural networks.

Man Wong received the B.Eng. degree in computer science from the Hong Kong University of Science and Technology in 1996 and M.Sc. degree (with distinction) in scientific computing from the Hong Kong Baptist University in 2000.

He is now working for the Information Technology Services Department of the Hong Kong Government.

Mr. Wong was awarded the Huang Hong Ci Prize for best M.Sc. dissertation.