Pergamon

Pattern Recognition, Vol. 28, No. 3, pp. 421–430, 1995
Elsevier Science Ltd

0031 3203/95 $9.50 + .00

0031-3203(94)00113-8

PARTIAL EIGENVALUE DECOMPOSITION FOR LARGE IMAGE SETS USING RUN-LENGTH ENCODING

JAMES B. ROSEBOROUGH and HIROSHI MURASE
NTT Basic Research Laboratories, 3-1, Morinosato Wakamiya, Atsugi, Kanagawa 243-01, Japan

(*Received* 9 *November* 1993; *in revised form* 9 *August* 1994; *received for publication* 26 *August* 1994)

Abstract—Pattern recognition using eigenvectors is a recent active research area. Finding eigenvectors of a large image set, however, has been considered to require too much computation to be practical. We therefore propose a new method for reducing computation of the partial eigenvalue decomposition based on run-length encoding. In this method, called the constant regions method, spatial encoding is used to reduce storage and computation, then coeigenvectors are computed and later converted to eigenvectors. For simple images, and when the number of pixels in an image is much larger than the number of images, the resulting algorithm is shown to grow as the first power of the basic image dimension, rather than the fourth power as for conventional methods. For comparison, the power method, the conjugate gradient method, and a so-called direct method for computing the partial eigenvalue decomposition are also presented, and recommendations are given for when each method should be used. The advantage of the proposed method are verified by tests in which the first several eigenvectors are computed for sets of images having varying complexity. This algorithm is useful for a research area of pattern recognition using eigenvectors.

Eigenvalue decomposition Run-length encoding Image processing Character recognition Sub-space methods

1. INTRODUCTION

Eigenvector analysis is important for many areas including signal processing,[1,2] image analysis,[3,4] pattern analysis, or character recognition.[5] Especially, pattern recognition methods using image eigenvectors, such as face recognition using eigenfaces[6] or object recognition using parametric eigenspace,[7,8] have been a recent active research area. An important property for image processing applications is the ability to extract the most important dimensions of a large sample space, and provide a controlled way of reducing computation. In practice, however, computing eigenvectors of high resolution images of large sample sizes often requires prohibitive amount of computation.

The wide variety of computational methods found in the literature reflect the diversity of application areas to which eigenvalue decomposition may be applied. Among methods suitable for very large data sets, a large number may be considered iterative gradient search methods.[9–12] In addition, there are specialized methods for specific problems, such as decomposition of matrices having Toeplitz structure,[13] matrices arising in stochastic systems,[14] or adaptive methods for use with data that arrive sequentially.[15] For extremely large data sets such as those arising in image processing applications, however, the computation as a function of data size can still be unacceptable.

We propose a method called the constant regions method for computing the partial eigenvalue solution which is based on spatially encoding the images, and computing the coeigenvectors as an intermediate step rather than the eigenvectors directly. In particular we use run-length encoding as a spatial encoding method. By doing this the computationally expensive step in the partial eigenvalue solution is reduced from the fourth power of the basic image dimension to linear for many practical problems. To achieve these reductions, the image storage requirements must be significantly reduced by the encoding method. For binary images and many synthesized images, this condition is easily met. For natural images, however, such significant reductions will not be realized in general without introducing small approximation errors.

In the following, we first review previous methods applicable to the present problem. Then we present our improvements based on spatial encoding, and give algorithms for the case of run-length encoding. We then analyze the computational and storage requirements for each algorithm, and give criterion when each should be used. Finally, we include experimental results which demonstrate the effectiveness of the proposed method.

2. PRESENT ALGORITHMS FOR PARTIAL EIGENVALUE DECOMPOSITION

In this section we will present three existing methods of computing the partial eigenvalue decomposition: the power method, the conjugate gradient method, and a direct method. We begin by introducing some terminology.

We assume there is a set of m vectors, $\mathbf{x}_1, \ldots, \mathbf{x}_m$ representing m images of interest. We will call $\mathbf{x}_i$ the ith *template*. Each component $x_1, \ldots, x_n$ of a template $\mathbf{x}$ will correspond to the intensity of a pixel in the template's image array and n is the number of pixels in a template image. For convenience, the templates are combined into an $n \times m$ matrix, $\mathbf{X}$, with each template $\mathbf{x}_i$ forming a column of $\mathbf{X}$. That is,

$$\mathbf{X} \equiv [\mathbf{x}_1 \cdots \mathbf{x}_m]. \tag{1}$$

Note that by assuming the templates are vectors, we are not limiting ourselves to one-dimensional images, but rather assume that a convention has been chosen for converting data from a multidimensional array such as an image pixel array into a one-dimensional vector.

Since $\mathbf{X}$ is a linear operator, the $K \equiv \text{rank}(\mathbf{X})$ nonzero eigenvalues of $\mathbf{X}$, $\lambda_1, \ldots, \lambda_K$, and their associated eigenvectors $\mathbf{e}_1, \ldots, \mathbf{e}_K$ may be computed, which will satisfy the fundamental eigenvalue relationship

$$\lambda_i \mathbf{e}_i = \mathbf{R}\mathbf{e}_i, \quad i = 1, \ldots, K, \tag{2}$$

where $\mathbf{R}$ is the square, symmetric matrix computed from $\mathbf{X}$ as and its transpose, $\mathbf{X}^T$, as

$$\mathbf{R} = \mathbf{X}\mathbf{X}^T. \tag{3}$$

Often, only the $k \leq K$ largest eigenvalues $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_k| > 0$ and their associated eigenvectors $\mathbf{e}_1, \ldots, \mathbf{e}_k$ are retained to approximate image data. If we form a $n \times k$ matrix $\mathbf{E}^{(k)}$ from the first k eigenvectors,

$$\mathbf{E}^{(k)} \equiv [\mathbf{e}_1 \cdots \mathbf{e}_k], \tag{4}$$

then for each $\mathbf{x}_i$ a corresponding model $\mathbf{m}_i^{(k)}$ may be computed according to

$$\mathbf{M}^{(k)} \equiv [\mathbf{m}_1^{(k)} \cdots \mathbf{m}_m^{(k)}] = \mathbf{E}^{(k)T}\mathbf{X}. \tag{5}$$

We call $\mathbf{m}_i^{(k)}$ a kth *order model* of $\mathbf{x}_i$. It is a projection of the vector $\mathbf{x}_i$ into the subspace defined by the basis vectors of $\mathbf{E}^{(k)}$. Approximations $\hat{\mathbf{x}}_i^{(k)}$ to the original $\mathbf{x}_i$ may be recovered according to

$$\hat{\mathbf{X}}^{(k)} \equiv [\hat{\mathbf{x}}_1^{(k)} \cdots \hat{\mathbf{x}}_m^{(k)}] = \mathbf{E}^{(k)}\mathbf{M}^{(k)}. \tag{6}$$

The power of the eigenvalue method as it may be applied to image processing is that the $\mathbf{M}^{(k)}$ maximally represent the data $\mathbf{X}$ under the l^2 norm. That is, $\mathbf{E}^{(k)}$ is an $n \times k$ matrix that minimizes the error in the approximations $|\hat{\mathbf{X}}^{(k)} - \mathbf{X}|$ for any $1 \leq k \leq K$.

2.1. *The power algorithm*

A simple algorithm for computing eigenvalues and their corresponding eigenvectors is the power algorithm. In this method, an iteration is introduced involving the matrix $\mathbf{R}$ that computes the largest remaining eigenvalue and it's eigenvector, then this dimension is removed from $\mathbf{R}$ and the process is repeated until k pairs are found.

Suppose $\mathbf{R}$ is a square symmetric matrix computed according to (3). Then in the iteration defined by,

$$\left.\begin{aligned} &\hat{\mathbf{e}}_0\colon\ |\hat{\mathbf{e}}_0| = 1, \quad \hat{\mathbf{e}}_0^T\mathbf{e}_{\max} \neq 0 \\ &\left.\begin{aligned} \hat{\mathbf{e}}_t' &= \mathbf{R}\hat{\mathbf{e}}_{t-1} \\ \hat{\lambda}_t &= |\hat{\mathbf{e}}'| \end{aligned}\right] \\ &\hat{\mathbf{e}}_t = \hat{\lambda}_t^{-1}\hat{\mathbf{e}}_t' \end{aligned}\right]_t\colon \left|\frac{\hat{\lambda}_t}{\hat{\lambda}_{t-1}} - 1\right| > \varepsilon \tag{7}$$

the estimates $\hat{\lambda}_t$ and $\hat{\mathbf{e}}_t$ at iteration t will converge to the largest eigenvalue $\lambda_{\max}$ and eigenvector $\mathbf{e}_{\max}$. Because $\hat{\mathbf{e}}_0$ must not perpendicular to $\mathbf{e}_{\max}$, a randomly oriented unit vector is generally used for the initial guess, $\hat{\mathbf{e}}_0$. Throughout the paper, we use the notation

$$\left.\begin{aligned} a_c \\ b_c \end{aligned}\right] \quad \{c\} \tag{8}$$

to indicate that the steps a_c and b_c are to be computed under the set of c given in $\{c\}$. Thus in (7), $\{c\}$ specifies a stopping condition, with ε specifying a numerical tolerance usually chosen to reflect the precision of the computer.

Given an algorithm for finding the single largest eigenmode, the k largest eigenmodes may be found according to,

$$\left.\begin{aligned} &\mathbf{R}_1 = \mathbf{R} \\ &\hat{\lambda}_s = \cdots \\ &\hat{\mathbf{e}}_s = \cdots \\ &\mathbf{R}_{s+1} = \mathbf{R}_s - \hat{\lambda}_s\hat{\mathbf{e}}_s\hat{\mathbf{e}}_s^T \end{aligned}\right] s = 1, 2, \ldots, k \tag{9}$$

While the power algorithm is easy to implement, the main drawback is slow convergence under the condition that $|\lambda_s/\lambda_{s+1}|$ is close to 1. For practical problems this occurs frequently so a more sophisticated algorithm is required.

2.2. *Conjugate gradient method*

The conjugate gradient method for finding maximal values of a function is applied to the present problem by defining a suitable scalar function. The function to be maximized is the Raleigh quotient $F(\hat{\mathbf{e}})$ defined as

$$F(\hat{\mathbf{e}}) \equiv \frac{(\hat{\mathbf{e}}^T\mathbf{R}\hat{\mathbf{e}})}{(\hat{\mathbf{e}}^T\hat{\mathbf{e}})}. \tag{10}$$

It can easily be shown using (2) that when $F(\hat{\mathbf{e}})$ takes on a maximum value, $F(\hat{\mathbf{e}})$ will be equal to $\lambda_{\max}$ and $\hat{\mathbf{e}}$ will be colinear with $\mathbf{e}_{\max}$.

We now state the conjugate gradient algorithm for finding the largest eigenvalue of $\mathbf{R}$. More detailed explanations can be found in references (9) and (10). In the following, $\mathbf{g}_t$ is the gradient of $F(\hat{\mathbf{e}})$ evaluated at $\hat{\mathbf{e}} = \hat{\mathbf{e}}_t$, $\mathbf{h}_t$ is a direction of search, d_t is the distance along the search direction that minimizes the Raleigh coefficient, and the algorithm is reset every T steps,

$$\hat{\mathbf{e}}_0\colon\ \hat{\mathbf{e}}_0^{\mathrm{T}}\mathbf{e}_{\max}\neq 0$$

$$\left.\begin{aligned}
&\hat{\mathbf{e}}_t' = \mathbf{R}\hat{\mathbf{e}}_t\\
&\hat{\lambda}_t = F(\hat{\mathbf{e}}_t) = \frac{\hat{\mathbf{e}}_t^{\mathrm{T}}\mathbf{R}\hat{\mathbf{e}}_t}{\hat{\mathbf{e}}_t^{\mathrm{T}}\hat{\mathbf{e}}_t} = \frac{\hat{\mathbf{e}}_t^{\mathrm{T}}\hat{\mathbf{e}}_t'}{|\hat{\mathbf{e}}_t|^2}\\
&\mathbf{g}_t = \frac{2}{|\hat{\mathbf{e}}_t|^2}(\hat{\mathbf{e}}_t' - \hat{\lambda}_t\hat{\mathbf{e}}_t)\\
&\mathbf{h}_t = \begin{cases} \mathbf{g}_t & t = jT, j = 0,1,2,\ldots\\ \mathbf{g}_t + \dfrac{|\mathbf{g}_t|^2}{|\mathbf{g}_{t-1}|^2}\mathbf{h}_{t-1} & t \neq jT \end{cases}\\
&a_t = (\mathbf{h}_t^{\mathrm{T}}\mathbf{R}\mathbf{h}_t)(\hat{\mathbf{e}}_t^{\mathrm{T}}\mathbf{h}_t) - (\hat{\mathbf{e}}_t^{\mathrm{T}}\mathbf{R}\mathbf{h}_t)(\mathbf{h}_t^{\mathrm{T}}\mathbf{h}_t)\\
&b_t = (\mathbf{h}_t^{\mathrm{T}}\mathbf{R}\mathbf{h}_t)(\hat{\mathbf{e}}_t^{\mathrm{T}}\hat{\mathbf{e}}_t) - (\hat{\mathbf{e}}_t^{\mathrm{T}}\mathbf{R}\hat{\mathbf{e}}_t)(\mathbf{h}_t^{\mathrm{T}}\mathbf{h}_t)\\
&c_t = (\hat{\mathbf{e}}_t^{\mathrm{T}}\mathbf{R}\mathbf{h}_t)(\hat{\mathbf{e}}_t^{\mathrm{T}}\hat{\mathbf{e}}_t) - (\hat{\mathbf{e}}_t^{\mathrm{T}}\mathbf{R}\hat{\mathbf{e}}_t)(\hat{\mathbf{e}}_t^{\mathrm{T}}\mathbf{h}_t)\\
&d_t = \frac{-b_t + \sqrt{b_t^2 - 4a_tc_t}}{2a_t}\\
&\hat{\mathbf{e}}_{t+1} = \hat{\mathbf{e}}_t + d_t\mathbf{h}_t
\end{aligned}\right\}\ t\colon\ \frac{|\mathbf{g}_t|^2|\hat{\mathbf{e}}_t|^2}{|\hat{\lambda}_t|^2} > \varepsilon$$

$$\hat{\mathbf{e}} = |\hat{\mathbf{e}}_{t_{\mathrm{final}}}|^{-1}\hat{\mathbf{e}}_{t_{\mathrm{final}}}. \tag{11}$$

As before $\hat{\mathbf{e}}_0$ must be non-zero, and not perpendicular to $\mathbf{e}_{\max}$. The value ε is a stopping tolerance. The integer j is a dummy variable used to reset the algorithm every T steps. In this paper, we used $T = 5$. Note that in this case, the final estimate $\hat{\mathbf{e}}$ must be rescaled to be a unit vector after the convergence criterion has been met. Successive eigenvalues are found in the manner of (9).

This algorithm performs well and grows approximately linearly with the number of elements in $\mathbf{R}$ for large n. However, the initial computation of $\mathbf{R}$ varies with the square of n, so computing $\mathbf{R}$ becomes the limiting step for many practical problems.

2.3. *Eigenvalue determination without computing R*

The previous algorithms can be modified to compute the partial eigenvalue decomposition without explicit computation of the matrix $\mathbf{R}$. In this case the step $\hat{\mathbf{e}}_t' = \mathbf{R}\hat{\mathbf{e}}_t$ which occurs in both the power algorithm and conjugate gradient algorithm is replaced by the two steps,[10]

$$\begin{aligned}
\hat{\mathbf{e}}_t^- &= \mathbf{X}\hat{\mathbf{e}}_t\\
\hat{\mathbf{e}}_t' &= \mathbf{X}^{\mathrm{T}}\hat{\mathbf{e}}_t^-.
\end{aligned} \tag{12}$$

In this case, successive eigenvectors must be removed directly from a copy of the data matrix $\mathbf{X}$, rather than from $\mathbf{R}$, since the latter is not computed. This may be done according to

$$(\mathbf{x}_i)_{s+1} = \hat{\mathbf{e}}_s\hat{\mathbf{e}}_s^{\mathrm{T}}(\mathbf{x}_i)_s \quad i = 1,\ldots,m. \tag{13}$$

We call the algorithm resulting from the substitution of (12) and (13) into the conjugate gradient algorithm the *implicit* $\mathbf{R}$, or *IR* algorithm.

While the conjugate gradient algorithm almost always improves the computational speed over the power method, the substitution given by (12) will not always increase speed. If many eigenvalues are required, or m is very much larger than n, for example, the original conjugate gradient algorithm will be faster.

2.4. *The singular value decomposition algorithm*

Although $\mathbf{R}$ is an $n \times n$ matrix, the matrix

$$\tilde{\mathbf{R}} \equiv \mathbf{X}^{\mathrm{T}}\mathbf{X} \tag{14}$$

is $m \times m$ and is much smaller in practical problems. The matrix $\tilde{R}$ has eigenvalues $\tilde{\lambda}_1,\ldots,\tilde{\lambda}_K$ and associated eigenvectors $\tilde{\mathbf{e}}_i,\ldots,\tilde{\mathbf{e}}_K$ related to those of $\mathbf{R}$ by

$$\left.\begin{aligned}
\lambda_i &= \tilde{\lambda}_i\\
\mathbf{e}_i &= \tilde{\lambda}_i^{-1/2}\mathbf{X}\tilde{\mathbf{e}}_i
\end{aligned}\right\}\quad i = 1,\ldots,K. \tag{15}$$

We call $\tilde{\mathbf{e}}$ a coeigenvector and note that it is of length m, whereas the eigenvector $\mathbf{e}$ is of length n. When m is much smaller than n, using the above relations to compute the eigenstructure can greatly reduce the computational load. The coeigenvectors' elements are later used as weights for computing the true eigenvectors as linear combinations of the input images. The relation (15) was developed within the framework of the *singular value decomposition* theory,[11] so this approach of first computing the coeigenvectors and then using them to compute the eigenvectors is referred to as the *singular value decomposition* (SVD) algorithm. This method performs well for some problems, but the limiting step with many large data sets is still the computation of $\tilde{\mathbf{R}}$.

Tradeoffs among the various algorithms are taken up in more detail in Section 4.

3. EIGENVALUE DECOMPOSITION BASED ON SPATIAL ENCODING

The natural redundancy in images can be exploited to speed up the computation, often dramatically, through a spatial encoding scheme, such as run-length encoding, a quad-tree representation, or representation by polygon vertices. The run-length encoding results in simple algorithms and is suitable for natural as well as synthetic images, so we focus on that method in this paper. A general reference on image encoding methods is Rosenfeld and Kak.[16]

We basically enhanced the SVD method. To see how encoding reduces computation, we note that while $\mathbf{R}$ is an $n \times n$ matrix, the matrix

$$\tilde{\mathbf{R}} \equiv \mathbf{X}^{\mathrm{T}}\mathbf{X} \tag{16}$$

is $m \times m$, and has eigenvalues $\tilde{\lambda}_1,\ldots,\tilde{\lambda}_K$ and associated eigenvectors $\tilde{\mathbf{e}}_i,\ldots,\tilde{\mathbf{e}}_K$ related to those of $\mathbf{R}$ by,

$$\left.\begin{aligned}
\lambda_i &= \tilde{\lambda}_i\\
\mathbf{e}_i &= \tilde{\lambda}_i^{-1/2}\mathbf{X}\tilde{\mathbf{e}}_i
\end{aligned}\right\}\quad i = 1,\ldots,K. \tag{17}$$

The relation (17) comes from singular value decomposition.[12] It can yield good performance for some problem types, but the limiting step becomes the computation of $\tilde{\mathbf{R}}$ for many large problems.

The importance of (16) lies in the fact that the elements $\tilde{r}_{ij}$ of the matrix $\tilde{\mathbf{R}}$ are computed according to

$$\tilde{r}_{ij} = \mathbf{x}_i^T \mathbf{x}_j. \tag{18}$$

That is, $\tilde{r}_{ij}$ is the dot product between two images, $\mathbf{x}_i$ and $\mathbf{x}_j$. This means that by using suitable algorithms for the computation of the dot products in (18) which act directly on spatially encoded data, computations can be substantially reduced. Subsequently, we present the data structures and algorithms to achieve these reductions for the case of run-length encoding.

3.1. *Run-length encoding of images*

We now introduce notation for run-length encoding of images. By run-length encoding, we mean representing an image $\mathbf{x}$ by two sequences, namely, a sequence of run lengths, $\{u_k\}$, and values, $\{\chi_k\}$, defined recursively as

$$\left.\begin{aligned} u_k &= \max\{i : x_{\sigma(u_k)+1} = x_{\sigma(u_k)+2} = \cdots x_{\sigma(u_k)+i}\} \\ \chi_k &= x_{\sigma(u_k)+1} \end{aligned}\right] \quad k:\ k \geq 1,\ \sigma(u_k) < n. \tag{19}$$

We call the ordered pair (u_i, χ_i) the ith *run-length encoded cell of* x, or when the context is clear the ith cell. The notation $\sigma(u_k)$ is used to refer to the sum of the first $k-1$ run lengths, or 0 if k is 1, that is

$$\sigma(u_k) \equiv \sum_{i=1}^{k-1} u_i \tag{20}$$

and so refers to the number of components of $\mathbf{x}$ already encoded before the kth cell. In addition, we will use the notation c_x to refer to the cell count in the run-length encoded image of $\mathbf{x}$.

3.2. *Dot product of two run-length encoded images*

The dot product of two run-length encoded images may be computed directly from the run-length encoded values to reduce the computation of $\tilde{\mathbf{R}}$ in (16). To do so, another run-length encoded sequence is generated representing the partial products of the two images.

Assume image $\mathbf{x}$ is encoded by $\{(u, \chi)\}$ image $\mathbf{y}$ is encoded by $\{(v, \psi)\}$, and the sequence of run-length encoded partial products is given by $\{(w, \zeta)\}$. The sequences have numbers of cells equal to c_x, c_y and c_z, respectively. To perform the computation, we introduce i_s and j_s which are indices into the sequences $\{\chi\}$ and $\{\psi\}$, respectively, and l_s and m_s which give the number of components yet to be used in the partial product computations for the i_sth and j_sth cell respectively. Then the dot product $\mathbf{x}^T\mathbf{y}$ may be computed as follows,

$$\left.\begin{aligned} w_0 &= \min(u_0, v_0) \\ i_0 &= j_0 = 1 \\ l_0 &= u_1 \\ m_0 &= v_1 \end{aligned}\right]$$

$$\left.\begin{aligned} w_s &= \min(l_s, m_s) \\ (i_{s+1}, l_{s+1}) &= \begin{cases} (i_s, l_s - w_s) & w_s < l_s \\ (i_s + 1, u_{i_s+1}) & w_s = l_s \end{cases} \\ (j_{s+1}, m_{s+1}) &= \begin{cases} (j_s, m_s - w_s) & w_s < m_s \\ (j_s + 1, u_{j_s+1}) & w_s = m_s \end{cases} \\ \zeta_s &= \chi_{i_s}\psi_{j_s} \end{aligned}\right] \quad s:\ s \geq 1,\ \sigma(w_s) < n$$

$$\mathbf{x}^T\mathbf{y} = \sum_{s=1}^{c_z} w_s\zeta_s. \tag{21}$$

This is illustrated in Fig. 1. There, it can be seen that the number of cells in the partial product sequence is at most the sum of the number of cells in the component sequences, that is, $c_z \leq c_x + c_y$. The computation saved over non-run-length encoded methods is therefore directly related to the reduction in storage achieved by run-length encoding.

3.3. *Computation of eigenvectors from co-eigenvectors*

Once the elements of the matrix $\tilde{\mathbf{R}}$ are computed, the previous algorithms can be used to find the coeigenvectors, $\tilde{\mathbf{E}}^{(k)}$. For example, the conjugate gradient method was used for this stage in our experiments. Then, these are converted to eigenvectors according to

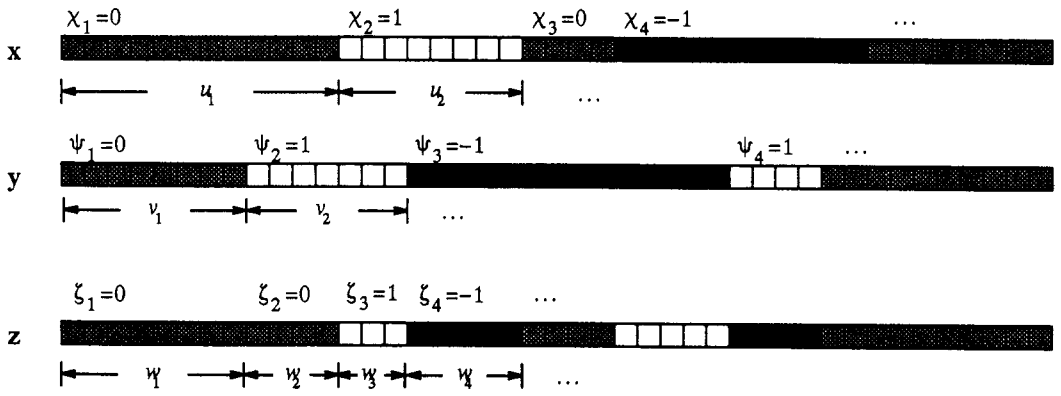


Fig. 1. Schematic diagram of the computation of a dot product from two run-length encoded images. Here, grey represents the level 0, black -1, and white 1.

(17). In this step as well, the run-length encoding can be used to save computation by multiplying a value only once, then summing repeatedly.

If the coeigenvector $\tilde{\mathbf{e}}$ is to be converted to the eigenvector $\mathbf{e}$ by (17), the computation may be expressed as

$$e_1 = \lambda^{-1/2} \sum_{j=1}^{m} x_{1j}\tilde{e}_j$$

$$e_i = e_{i-1} + \lambda^{-1/2} \sum_{j=1}^{m} (x_{ij} - x_{(i-1)j})\tilde{e}_j \quad i = 2, \ldots, n. \tag{22}$$

If the templates are substantially reduced by run-length encoding, than many of the differences $x_{ij} - x_{(i-1)j}$ in will be equal to zero, so the computations can be simplified.

We have presented the constant regions method for the specific case of run-length encoding. We call the algorithm consisting of the run-length encoded algorithms presented in this section, and the conjugate gradient algorithm to compute $\tilde{\mathbf{E}}$ from $\tilde{\mathbf{R}}$ the *run-length encoded* or *RLE* algorithm.

4. ANALYSIS

In this section, we compare the theoretical performance of the algorithms that have been presented and indicate the conditions under which each will be preferred over the others. Figure 2 shows the block structure of each algorithm. Here blocks represent operations on data, and arrows represent data of a specific type being passed from one process to another. From these diagrams, the storage and computation requirements may be readily determined.

4.1. *Problem characteristic parameters*

To review, the problem consists of m templates each of size n, and the k largest eigenvectors are to be computed. The value m is called the sample size, n is called the data size, and together they describe the basic problem size. The value k is also specified by the user based on the problem requirements, and will influence the decision as to which algorithm is preferred.

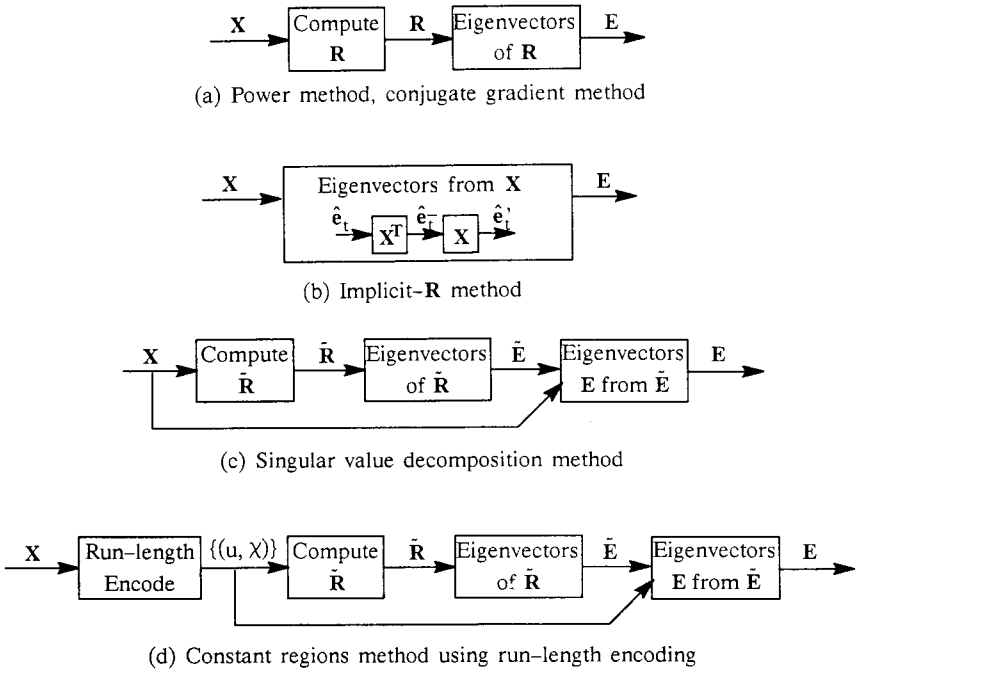
In Section 3.2, we said that the reduction in computation is related to the amount of data reduction so it will also depend on the image material. We introduce a data reduction factor f_x for template x, and average data reduction factor $\bar{f}$ for the set of templates $\mathbf{X}$ to describe these dependencies. These are defined as

$$f_x \equiv \frac{c_x}{n^{1/2}} \tag{23}$$

$$\bar{f} \equiv \frac{1}{m} \sum_{j=1}^{m} f_{x_j}, \tag{24}$$

where c_x is the cell count of Section 3.2.

The average data reduction factor cannot be reliably known before the data are encoded, so for our purposes we estimate this number based on knowledge about the application. For example, consider the template shown in Fig. 3. Under run-length encoding, the first several rows of pixels, which are all white, will be condensed into a single run-length encoded cell. Subsequent rows will be split into between two and six (for



(a) Power method, conjugate gradient method

(b) Implicit-$\mathbf{R}$ method

(c) Singular value decomposition method

(d) Constant regions method using run-length encoding

Fig. 2. Data flow diagrams by algorithm. Arrows represent data and blocks represent operations on data.

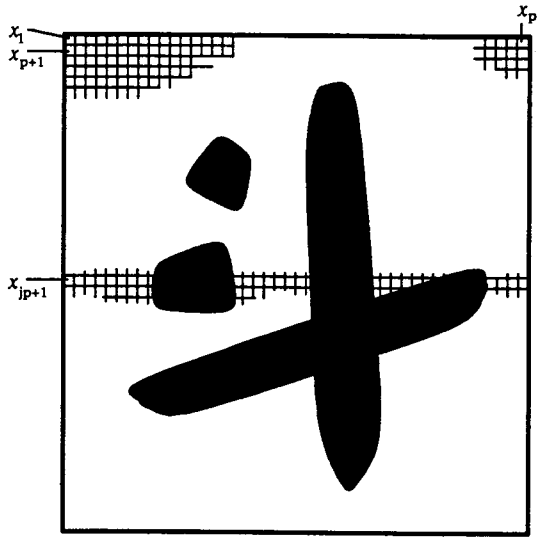


Fig. 3. Example template showing data encoding method. Rows near the jth row require 6 run-length encoded cells to be represented.

the example shown) cells per row, even if the number of rows is doubled or halved. Hence the cell count c_x will be approximately linear with the resolution of a single image dimension, that is $c_x \approx f_x n^{1/2}$. Defined in this way, f_x will is largely independent of the image resolution, and serves to characterize the complexity of the image.

Finally, we use the value $\bar{\imath}$ to refer to the average number of iterations per eigenvector to meet the convergence criterion. As with $\bar{f}$, this can only be estimated. Without attempting to analyze this further, we note that for a variety of problem types and sizes we have used, the conjugate gradient algorithm required between 10 and 15 iterations to converge independent of the eigenvalues and specific problems.

4.2. *Computation and storage requirements*

The computation and storage requirements for the algorithms as a function of the problem parameters of Part A of this section are summarized in Table 1 as functions of the problem parameters n, m, k, $\bar{f}$ and $\bar{\imath}$. Computation requirements refer to approximate number of floating point multiplications that must be performed to execute that algorithm, and storage requirements refers to the sum of the storage requirements for each type of data produced by the algorithmic elements.

From Table 1, it can be seen that for high-resolution problems, that is $n \gg m \gg k$, the run-length encoded algorithm will be superior to the others. By way of specific example, suppose 200 templates of size 128×128 and similar in complexity to Fig. 3 are used, and 10 eigenvectors are desired. For this case $n = 16384$, $m = 200$ and $k = 10$, and is typical of the applications we are using. Estimating $\bar{f} \approx 6$ and $\bar{\imath} \approx 12$, the number of multiplications for the classical and run-length encoded methods are 3.01×10^{10} and 8.49×10^7, respectively. The numbers of storage locations required are 1.38×10^8 and 4.30×10^5, respectively. Thus computation and storage are reduced two orders of magnitude by run-length encoding over the power method.

Table 1. Expected computation and storage requirements by (a) algorithmic sub-process, and (b) complete algorithm

(a) By algorithmic subcomponent

Step	Algorithm CG	IR	SVD	RLE	Computational requirements Storage	Multiplications
Retain $\mathbf{X}$	•	•	•		nm	—
Encode $\mathbf{X}$				•	$\bar{f}n^{1/2}m$	—
$\mathbf{R} = \mathbf{X}\mathbf{X}^T$	•				n^2	$1/2n^2m$
$\tilde{\mathbf{R}} = \mathbf{X}^T\mathbf{X}$			•		m^2	$1/2nm^2$
$\mathbf{R}(\{(u_j, \chi_j)_m\})$				•	m^2	$1/2\bar{f}n^{1/2}m^2$
$eig(\mathbf{R})$	•				n	$k\bar{\imath}n^2$
$eig(\tilde{\mathbf{R}})$			•	•	m	$k\bar{\imath}m^2$
$eig(\mathbf{X}, \mathbf{X}^T)$		•			m	$2k\bar{\imath}nm$
$\tilde{\mathbf{e}}(\mathbf{e}, \mathbf{X})$			•		—	knm
$\tilde{\mathbf{e}}(\mathbf{e}, \{(u_j, \chi_j)\})$				•	—	$k\bar{f}n^{1/2}m$
Retain $\mathbf{E}$	•	•	•	•	kn	—

(b) By complete algorithm

Algorithm	Storage	Multiplications
Conjugate gradient	$n(n+m+1)+kn$	$n^2(\frac{1}{2}m + k\bar{\imath})$
Implicit-$\mathbf{R}$	$n(m+1)+kn$	$2k\bar{\imath}nm$
Singular value decomposition	$m(n+m+1)+kn$	$nm(\frac{1}{2}m + k) + k\bar{\imath}m^2$
Run-length encoded	$m(\bar{f}n^{1/2}+m+1)+kn$	$n^{1/2}m(\frac{1}{2}\bar{f}m + k) + k\bar{\imath}m^2$

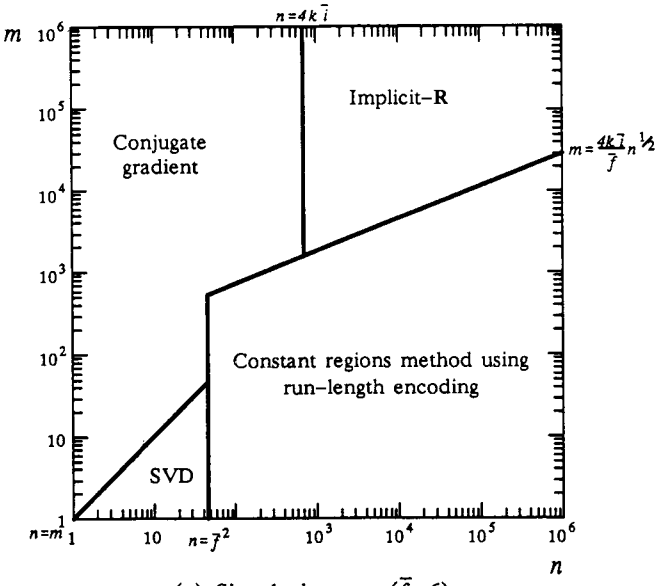
4.3. *Regions of dominance for each algorithm*

The results of the previous subsection can be combined into a plot showing regions of dominance for each algorithm as a function of the problem size parameters n and m. This has been done and is presented in Fig. 4. This figure has been drawn for specific values of $\bar{f}$, $\bar{\imath}$, and k, but asymptotes and intercepts have been indicated to illustrate the dependencies on these parameters. Note that in Fig. 4(b), although the SVD algorithm appears to be of more use than the RLE algorithm, for square images the assumption that $\bar{f} = 100$ implies that n greater than 1000, so in practice, RLE will always dominate SVD.

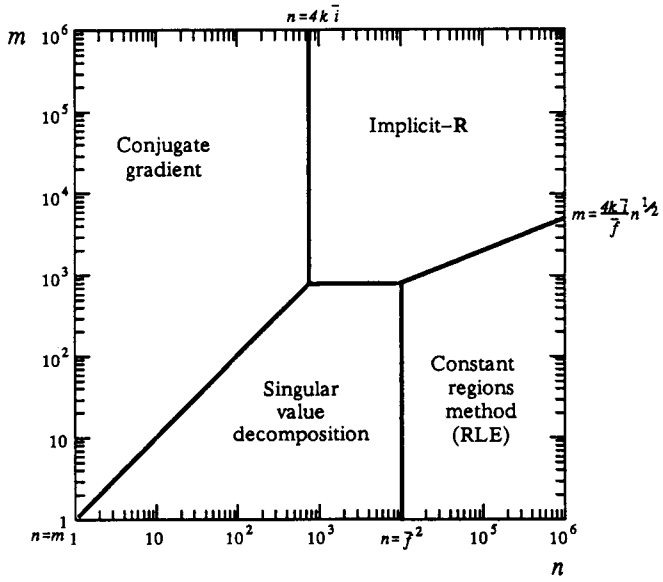
4.4. *Summary of comparison among methods*

We briefly summarize the results of this section regarding the applicability of the various algorithms under various conditions.

- For large sample size n compared with the data size m, run-length encoding should be used.



(a) Simple images ($\bar{f}$=6)



(b) Complex images ($\bar{f}$=100)

Fig. 4. Tradeoffs among algorithms and regions of applicability for the case of (a) simple images ($\bar{f} = 6$, $\bar{\imath} = 12$, $k = 20$), and (b) complex images ($\bar{f} = 100$, $\bar{\imath} = 12$, $k = 20$).

- For large data size m compared with the sample size n, the normal conjugate gradient method is preferred.
- For m and n approximately equal, and as image complexity increases,
  — very small number k of eigenvectors is desired, the implicit-**R** is preferred.
  — moderate or large number k of eigenvectors is desired, run-length encoding is preferred.

More precise tradeoffs, if necessary, can be computed according to Fig. 4. If desired, software can be programmed to switch automatically among the algorithms based on the problem specifications at run time.

5. EXPERIMENTAL RESULTS

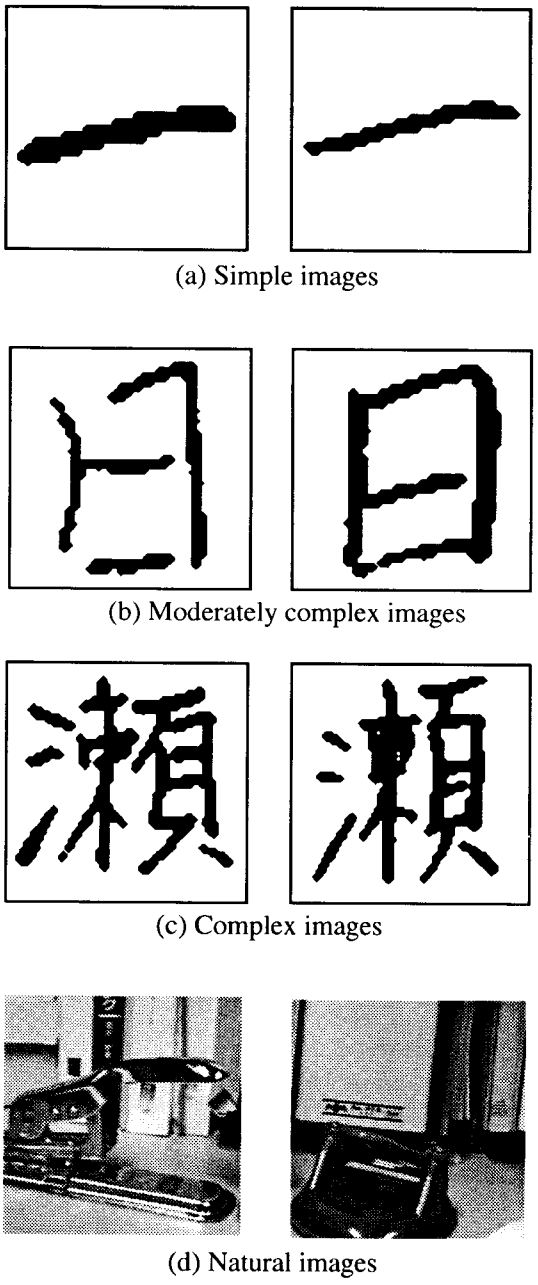
To test the methods in a realistic setting, eigenvectors were computed for four sets of data of varying complexity. We present the results of these tests in this section.

We used three subsets of the Japanese daily use Kanji having small, medium, or large numbers of strokes to test the algorithms for a variety of image complexities. In addition, we used a set of natural images. Figure 5 shows a single sample image from each of these four image sets.

In all cases, a sample size $m = 100$ was used, and the image resolution was varied for each data set. We assumed that $k = 20$ eigenvectors would be sufficient to characterize the data. We then computed the first $k = 20$ eigenvectors for each image set using the methods presented in this paper. Figure 6 shows a sample of the computed eigenvectors for the second set of images, characters having a moderate number of strokes. As none of the methods rely on approximation other than the fact that they are all iterative methods, the computed eigenvectors for all methods did not differ significantly.

We recorded the storage requirements and computing time for the various problem types and image resolutions, and the results are summarized in Table 2. The power method, because it is almost always inferior to the conjugate gradient method, was not analyzed.

In Table 2 we see that the results of the analytical section are confirmed. Namely, for the run-length en-



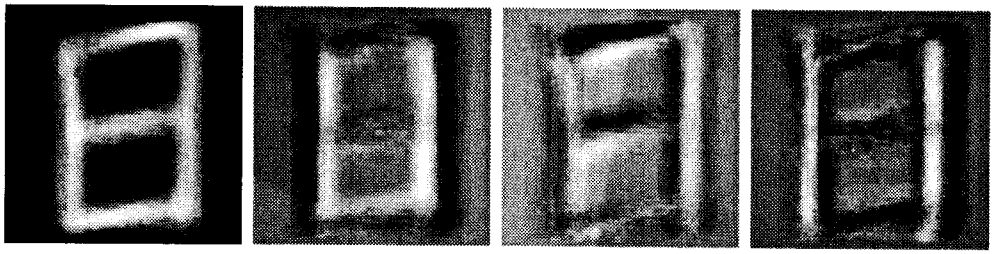
(a) Simple images

(b) Moderately complex images

(c) Complex images

(d) Natural images

Fig. 5. Sample of images having varying complexity used to test algorithms.



$\lambda_1 = 33.1$ $\lambda_2 = 9.4$ $\lambda_3 = 4.4$ $\lambda_4 = 3.6$

Fig. 6. Sample of computed eigenvectors for the test set of moderately complex character images.

Table 2. Measured storage locations (processor time) by algorithm, resolution, and image complexity

Size	Type	CG (Kb)	CG (s)	SVD (Kb)	SVD (s)	IR (Kb)	IR (s)	RLE (Kb)	RLE (s)
	Simple	8813	(1978)	1736	(148)	426	(553)	135	(19.3)
$n = 1024$	Moderate	8813	(2299)	1736	(152)	426	(553)	247	(33)
(32×32)	Complex	8813	(2552)	1736	(153)	426	(553)	321	(42)
	Natural	8813	(2147)	1736	(151)	426	(464)	1283	(95)
	Simple	×	×	6702	(583)	1703	(4034)	255	(45)
$n = 4096$	Moderate	×	×	6702	(583)	1703	(4034)	486	(78)
(64×64)	Complex	×	×	6702	(583)	1703	(4034)	649	(90)
	Natural	×	×	6702	(576)	1703	(2156)	4791	(343)
	Simple	×	×	×	×	×	×	2379	(466)
$n = 65536$	Moderate	×	×	×	×	×	×	3051	(549)
(256×256)	Complex	×	×	×	×	×	×	3735	(625)
	Natural	×	×	×	×	×	×	×	×

coded method, the storage requirements and computations are reduced, especially for large problems such as the complex Kanji problem. In addition, it can be seen that computation and storage requirements are dependent on image complexity, and are approximately proportional to the basic image resolution $n^{1/2}$ for the run-length encoded algorithm, as expected.

In some cases, we exceeded the memory capacity of our workstation (approximately 8 megabytes of available virtual memory) and were unable to compute the partial decomposition. These cases have been indicated by an "X" in Table 2. Thus the run-length encoding method demonstrates clear advantages in that otherwise infeasible problems have been made feasible. Note that no algorithm was able to handle the set of natural images at the highest resolution.

6. DISCUSSION

In Section 3, we presented an algorithm based on run-length encoding, but other spatial encoding methods could be used with similar results. For example, reduction by quad-tree representation[17] could also be used. In the case of images composed of discrete patches with smooth boundaries, both methods will have storage sizes and computation requirements which are linear with the basic image resolution in the limit as this dimension is increased. For synthesized images of consisting only of polygons, a resolution independent eigenvectors also consisting of polygons may be computed.

Perhaps the biggest drawback of the method is its failure to significantly improve computation for grey scale images of natural scenes. However, in our tests run-length encoding did improve performance for these cases as well. The exact run-length encoding according to Section 3 will seldom reduce the storage for natural images significantly, and it is possible that it may increase it. This is because adjacent pixels have approximate, rather than exact correspondence, so the encoded run-lengths are usually one or a few pixels. By allowing small approximation errors in the encoding process to increase run lengths, the run-length encoding method as given above can again be used to realize significant gains in speed for small reductions in accuracy.

Specifically, when encoding, a tolerance value can be set representing the largest grey scale error allowable. Data can be collected in a run-length encoded cell until the range of values in the cell exceeds the tolerance. The value of the cell is determined from the average of components within the cell. This will bound the overall error to a predetermined amount, and reduce storage and computation. Note that because approximation errors are limited in the l^2 norm, error in the computed eigenvectors will also be limited to within predetermined bounds.

Alternatively, the histogram of the image can be computed, and divided into h grey level groups of equal numbers. The image is then discretized according to these values, using a closest-level scheme. This will assure that the run-length set is good with respect to the l^2 norm. We did not examine the approximate methods in our approach, but rather restricted our study to cases where exact eigenvectors are computed.

Wallace[18] has developed an automatic clustering algorithm which automatically produces an encoded image with the shortest description length. Because computation is directly related to the storage size, such a method may be useful in combination with our method for handling natural images.

7. CONCLUSIONS

We have shown that by introducing spatial encoding and computing coeigenvalues rather than eigenvalues, the computational and storage requirements can be severely reduced for an important class of problems in image processing, namely large data sizes and moderately complex images. In such cases, the computation can be effectively reduced from the fourth power of the basic image resolution to the first power for images containing large, separable regions of uniform intensity. The method uses the natural redundancy occurring in images to reduce the number of repeated computations that must be performed.

We analyzed five algorithms for computation and storage requirements, and gave tradeoffs based on problem size for choosing among them. The run-length encoding methods appear most useful for very large problems, where high image resolution is desired, and fairly large sets of templates will be used.

We also demonstrated the effectiveness of the method on a variety of data. We computed the partial eigenvalue reduction of the subsets of the daily use Japanese Kanji characters of varying complexity and a set of natural images. Computation and storage requirements were reduced by the run-length encoded method in all cases examined, and the characteristic of linearly increasing computation based on the basic image resolution was confirmed. Some very large problems were only computable using our method. Though the method loses some of its power for natural images, in our experiments it still improved speed and storage over the other methods.

REFERENCES

1. D. W. Tufts and C. D. Melissinos, Simple, effective computation of principle eigenvectors and their eigenvalues and application to high resolution of frequencies, *IEEE Transactions of Acoustic, Speech, and Signal Processing* **ASSP-34**(5), 1046–1053 (October 1986).
2. H. Messer and Y. Rockah, On the eigenstructure of the signal-only tempo-spatial covariance matrix of broadband sources using a circular array, *IEEE Transactions of Acoustic, Speech, and Signal Processing* **38**(3), 557–559 (March 1990).
3. S. Umeyama, An eigendecomposition approach to weighted graph matching problems, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **10**(5), 695–703 (September 1988).
4. J. B. Burl, Estimating the basis functions of the Karhunen–Loeve transform, *IEEE Transactions of Acoustic, Speech, and Signal Processing* **37**(1), 99–105 (January 1989).
5. H. Murase, F. Kimura, M. Yoshimura and Y. Miyaka, An improvement of the autocorrelation matrix in the pattern matching method and its application to hand-printed 'HIRIGANA' recognition, *Trans. IECE* **J64-D**(3), 267–283 (1981).
6. M. Turk and A. Pentland, Face recognition using eigenfaces, *Proc. of IEEE Conference on Computer Vision and Pattern Recognition* 586–591 (June 1991).
7. H. Murase and S. Nayar, Learning object models from apparance, AAAI-93, Washington D.C., pp. 836–843 (July 1993).
8. H. Murase and S. Nayar, Illumination planning for object recognition in structured environments, *IEEE Conference on Computer Vision and Pattern Recognition* 31–38 (June 1994).
9. X. Yang, T. K. Sarkar and E. Arvas, A survey of conjugate gradient algorithms for solution of extreme eigen-problems of a symmetric matrix, *IEEE Transactions of Acoustic, Speech, and Signal Processing* **37**(10), 1550–1555 (October 1989).
10. R. Haimi-Cohen and A. Cohen, Gradient-type algorithms for partial singular value decomposition, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-9**(1) (January 1987).
11. H. Murakami and V. Kumar, Efficient calculation of primary images from a set of images, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-4**(5), 511–515 (September 1982).
12. S. Shlien, A method for computing the partial singular value decomposition, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-4**(6), 671–676 (November 1982).
13. Y. H. Hu and S. Y. Kung, Toeplitz eigensystem solver, *IEEE Transactions of Acoustic, Speech, and Signal Processing* **ASSP-33**(4), 1264–1271 (October 1985).
14. A. K. Jain, A fast Karhunen–Loeve transform for a class of random processes, *IEEE Transactions of Communications* 1023–1029 (September 1976).
15. J. Karhunen, Adaptive algorithms for estimating eigenvectors of correlation type matrices, in *Proc. 1984 IEEE Int. Conf., Acoust., Speech, Signal Processing* San Diego, California, pp. 14.7.1–14.7.4 (March 1984).
16. A Rosenfeld and A. C. Kak, *Digital Picture Processing.* Academic Press, New York (1976).
17. H. Samet, Region representation: quadtrees from boundary codes, *Comm. ACM 23*, **3**, 163–170 (March 1980).
18. R. S. Wallace and Y. Suenaga, Color face image segmentation using MDL clustering, *1990 Spring National Convention Record, the Institute of Electronics, Information and Communication Engineers* Tokyo (March 1990).

About the Author—HIROSHI MURASE received his B.E., M.E. and Ph.D. degrees in electrical engineering from the University of Nagoya, Japan, in 1978, 1980 and 1987, respectively. From 1980 to the present, he has been engaged in pattern recognition, and character recognition at Nippon Telegraph and Telephone Corporation (NTT). From 1992 to 1993, he was a visiting research scientist at Columbia University in New York. He is presently a senior research scientist in NTT Basic Research Laboratories, and a distinguished technical member in NTT. His current research interests lie in the areas of computer vision, pattern recognition and human visual perception. He was awarded the best paper at the 1994 IEEE conference on Computer Vision and Pattern Recognition. He is a member of the IEEE, the IEICE Japan, and the Audio-Visual Information Research Group of Japan.

About the Author—JAMES B. ROSEBOROUGH was born in Wheaton, Illinois on 11 January 1960. He received the B.S.M.E. degree in 1982 from Duke University, and the S.M. degree in mechanical engineering and Ph.D. degree in man–machine systems from M.I.T. in 1984 and 1988, respectively. He was a researcher in NTT Basic Research Laboratories from 1989 to 1990, where he was working on computer vision. He is now working for GO corporation.