Pergamon

Neural Networks, Vol. 8, No. 1, pp. 67–86, 1995

0893-6080/95 $9.50 + .00

0893-6080(94)00061-1

*CONTRIBUTED ARTICLE*

# A Multilayer Self-Organizing Feature Map for Range Image Segmentation

JEAN KOH,[1] MINSOO SUK, AND SUCHENDRA M. BHANDARKAR[2]

[1]Syracuse University and [2]University of Georgia

(*Received* 12 *November* 1992; *revised and accepted* 17 *May* 1994)

**Abstract**—*This paper proposes and describes a hierarchical self-organizing neural network for range image segmentation. The multilayer self-organizing feature map (MLSOFM), which is an extension of the traditional (single-layer) self-organizing feature map (SOFM) is seen to alleviate the shortcomings of the latter in the context of range image segmentation. The problem of range image segmentation is formulated as one of vector quantization and is mapped onto the MLSOFM. The MLSOFM combines the ideas of* self-organization *and* topographic mapping *with those of* multiscale *image segmentation. Experimental results using real range images are presented.*

**Keywords**—Range image segmentation, Self-organizing feature map, Neural networks, Computer vision.

## 1. INTRODUCTION

The availability of fast, accurate, reliable, and economical range sensors has prompted a rapid increase in the use of range images as input data for computer vision systems in recent years. A range image is usually formatted as an array of pixels such that the pixel values encode the depths or the distances of points on a visible scene surface from the range sensor. The depth value at each pixel reflects (1) the surface geometry and viewing geometry in terms of the distance of the corresponding point on a visible scene surface from the range sensor, and (2) the characteristics of the range sensor such as spatial resolution, range resolution, dynamic range, and the sensor noise parameters. The most attractive feature of using range images is that the surface information is made explicit. Although surface information can also be inferred from intensity images, it is a more difficult problem. Since a large number of factors, such as surface geometry, surface reflectance, surface texture, scene illumination, etc., are encoded in the pixel brightness value during the intensity image formation process, techniques for deriving 3-D structure from 2-D images such as shape from shading, shape from texture, and shape from motion tend to be ill-posed and need to make constraining assumptions about the scene and imaging parameters.

Computer vision can be looked upon as an information processing activity that involves construction of representations at successive levels of abstraction (Marr & Nishihara, 1978). A *segmented image,* produced by grouping the elements of an input image into semantically meaningful entities, is generally considered to be the highest domain-independent abstraction of the input data. Typically, a segmented image is the input to high-level vision which then utilizes domain-specific knowledge to interpret and analyze the image contents. Although depth information is explicitly available in a range image, the problems of 3-D segmentation and 3-D feature extraction still need to be addressed as they do for intensity images. In the context of range images, the problem of segmentation could be looked upon as one of grouping range image pixels into clusters that represent smooth surface regions bounded by surface discontinuity contours.

The purpose of this paper is to describe a neural network structure and the associated learning procedure suitable for the task of range image segmentation. The segmentation technique described in this paper is based on feature vector clustering and is mapped onto the proposed network that consists of multiple layers. Each layer is a conventional (single-layer) self-organizing feature map (SOFM) consisting of Kohonen units. The overall structure is in the form of a pyramid, thus

Acknowledgements: The authors wish to thank the anonymous referees for their insightful and detailed comments on previous versions of our paper. Their comments and suggestions have made the final paper much more readable than the earlier versions. The authors also wish to thank the Pattern Recognition and Image Processing Laboratory at Michigan State University, East Lansing, MI for range images from their range image data base.

Requests for reprints should be sent to Jean Koh, Department of Electrical and Computer Engineering, 121 Link Hall, Syracuse University, Syracuse, NY 13244-1240.

allowing us to compute the multiscale abstraction of the input which is termed as an *abstraction tree.* The multilayer self-organizing feature map (MLSOFM) that we have proposed (Suk & Koh, 1993) combines the ideas of *self-organization* and *topographic mapping* with those of *multiscale* image segmentation. It is distinctly unique from previous approaches to image segmentation using neural networks (Geman & Geman, 1984; Zerubia & Geiger, 1991; Daily, 1989; Tenorio & Hughes, 1987; Lin, Tsao, & Chen, 1991; Bilbro, White, & Snyder, 1987; Pentland, 1989; Hurlbert & Poggio, 1989; Grossberg & Mingolla, 1987; Scherf & Roberts, 1990). The performance of the network is demonstrated with a few selected experimental results using real range images.

## 2. RANGE IMAGE SEGMENTATION

Range image segmentation techniques can be classified as either edge-based or region-based depending on whether they emphasize the detection of surface discontinuities or the detection of smooth surface regions respectively.

The basic idea behind edge-based range image segmentation techniques (Langridge, 1984; Zucker & Hummel, 1973; Inokuchi et al., 1982; Mitiche & Aggarwal, 1983; Brady et al., 1985; Fan, Medioni, & Nevatia, 1987; Lee & Pavlidis, 1988) is to detect and classify range image pixels that signify surface discontinuities and classify them as one of jump edges, crease edges, or curvature edges. The surface discontinuities detected are linked together to form surface discontinuity contours or boundaries. One of the primary drawbacks of edge-based segmentation techniques is the inevitable fragmentation of the edges which then need to be linked using a heuristic technique. Smoothing operations for reducing noise tend to smear the edge over several pixels, making edge localization difficult. For these reasons, edge-based segmentation techniques have proved to be less popular than region-based segmentation techniques.

The central idea behind region-based range segmentation techniques is to estimate the surface curvature at each range pixel and cluster range pixels with homogeneous surface curvature properties to form smooth surface regions. There are two broad categories of region-based segmentation techniques: (a) region-growing techniques and (b) feature vector clustering techniques.

In region-growing techniques (Besl & Jain, 1988; Haralick, Watson, & Laffey, 1983; Vemuri & Aggarwal, 1986; Flynn & Jain, 1991), each pixel is classified as one of predefined qualitative surface types based on the signs of the mean, Gaussian, and principal curvatures. The initial classification is used to form seed regions for the subsequent region-growing stage that attempts to segment the range image using flexible bivariate surface fitting. Shortcomings of this approach are: (i) a good criterion based on sensor noise and surface curvature estimates is critical for merging and splitting adjacent regions, and (ii) it is possible for adjacent object surfaces separated by a crease or curvature edge to be accidentally merged (i.e., undersegmentation) and also for a given object surface to be incorrectly segmented into more than one surface region (i.e., oversegmentation).

In feature vector clustering techniques (Ittner & Jain, 1985; Hoffman & Jain, 1987), each pixel is associated with an appropriate feature vector, and the segmentation problem can be posed as one of (feature) vector quantization. Vector quantization is a process of partitioning an $n$-dimensional feature vector space into $M$ subspaces to minimize a criterion function when all the points in each subspace are approximated (or represented) by the representative vector $\mathbf{X}_i$ associated with that subspace.

The segmentation method proposed in this paper is based on feature vector clustering. This choice was prompted by the close relationship between vector quantization and the SOFM. To design a segmentation algorithm based on vector quantization, we need to address the following issues: (1) choice of an appropriate set of features, (2) choice of an appropriate partitioning algorithm that would minimize an objective function based on overall quantization distortion, and (3) choice of a computational structure on which to map the partitioning algorithm. These issues are discussed in detail in subsequent subsections. We show that the MLSOFM proposed in this paper overcomes two major problems inherent in the vector quantization approach to image segmentation:

1. In the vector quantization procedure, the number of subregions or subspaces $M$ is assumed to be known a priori.
2. The vector quantization procedure does not guarantee spatial connectivity. That is to say, pixels that correspond to input vectors that belong to a single subspace or cluster in the feature space are not guaranteed to be spatially connected in terms of the image coordinates.

### 2.1. Choice of Feature Vectors

In segmentation via vector quantization, the homogeneity of a segmented region is enforced by the appropriate choice of feature vectors. Because range images are an explicit representation of the scene surface geometry in sampled form, it is reasonable to expect that segmentation of range images should be based on a homogeneity criterion that incorporates geometrical properties of 3-D surfaces. The geometrical properties need to satisfy the following criteria:

1. The properties should be *generic* so that they have applicability over a wide spectrum of application domains.

2. The properties should fully characterize the surface.
3. The properties should provide a rich enough description to be of use to higher-level vision processes.
4. The properties should have local support, that is, they should be computable in a local window centered around a range image pixel of interest.
5. The properties should be stable and invariant over large ranges of viewpoint and scale.
6. The properties should be reliable, robust, and readily detectable by procedures that are computationally stable.

The mean, Gaussian and principal surface curvatures satisfy the above criteria.

A homogeneous region in range images can be bounded by three types of edges: jump edges, crease edges, or curvature edges. Jump edges occur where depth values are discontinuous in a range image. Such edges occur when an object occludes another object or when a part of an object occludes itself. We therefore need to incorporate the depth information (i.e., the range value) as one of features in the feature vector because we intend to use the position of jump edges as one of the decision boundaries during the process of clustering or vector quantization. Crease edges correspond to surface pixels where the surface normals are discontinuous, whereas curvature edges are characterized by continuity of the surface normals but discontinuity of the surface curvature. Thus, we also need to incorporate the unit surface normal and the invariant surface curvature values as features in the feature vector to use crease edges and curvature edges, respectively, as decision boundaries between clusters or surface regions. Among the invariant surface curvatures, the mean and Gaussian curvatures have been the most widely used in range image segmentation algorithms (Besl & Jain, 1988). The position coordinates $x$ and $y$ are also selected as features to preserve the spatial connectivity of subregions in the final segmented image. The input feature vector for the clustering algorithm is therefore an eight-dimensional vector: $\mathbf{x} = (x, y, z; n_x, n_y, n_z; H; K)$, where the $(n_x, n_y, n_z)$ is the unit normal vector at the point $(x, y, z)$ on the three-dimensional surface. $H$ and $K$ are the mean and Gaussian curvatures, respectively, at the point $(x, y, z)$.

We assume that the input image is given in the form of a Monge patch (O'Neill, 1966), that is, $z = f(x, y)$. In parametric form, the Monge patch can be represented as $x = u$, $y = v$, and $z = f(u, v)$. One can show that

$$\mathbf{x}(u, v) = (u, v, f(u, v)) \tag{1}$$

$$\mathbf{x}_u(u, v) = (1, 0, f_u(u, v)) \tag{2}$$

$$\mathbf{x}_v(u, v) = (0, 1, f_v(u, v)) \tag{3}$$

$$\mathbf{x}_{uu}(u, v) = (0, 0, f_{uu}(u, v)) \tag{4}$$

$$\mathbf{x}_{vv}(u, v) = (0, 0, f_{vv}(u, v)) \tag{5}$$

$$\mathbf{x}_{uv}(u, v) = \mathbf{x}_{vu}(u, v) = (0, 0, f_{uv}(u, v)) \tag{6}$$

$$\mathbf{n}(u, v) = \frac{\mathbf{x}_u(u, v) \times \mathbf{x}_v(u, v)}{\|\mathbf{x}_u(u, v) \times \mathbf{x}_v(u, v)\|} = \frac{(-f_u, -f_v, 1)}{\sqrt{1 + f_u^2 + f_v^2}} \tag{7}$$

where $\mathbf{x}_u(u, v) = \{[\partial \mathbf{x}(u, v)]/\partial u\}$, $\mathbf{x}_v(u, v) = \{[\partial \mathbf{x}(u, v)]/\partial v\}$, $\mathbf{x}_{uu}(u, v) = \{[\partial^2 \mathbf{x}(u, v)]/\partial u^2\}$, $\mathbf{x}_{uv}(u, v) = \{[\partial^2 \mathbf{x}(u, v)]/\partial u \partial v\}$, $\mathbf{x}_{vv}(u, v) = \{[\partial^2 \mathbf{x}(u, v)]/\partial v^2\}$ and $\mathbf{n}(u, v)$ is the surface normal vector at $(u, v)$.

The elements $g_{ij}$ of the *first fundamental form* matrix $\mathbf{G}$ of the surface are given by

$$g_{11} = E = \mathbf{x}_u \cdot \mathbf{x}_u = 1 + f_u^2 \tag{8}$$

$$g_{12} = g_{21} = F = \mathbf{x}_u \cdot \mathbf{x}_v = f_u f_v \tag{9}$$

$$g_{22} = G = \mathbf{x}_v \cdot \mathbf{x}_v = 1 + f_v^2. \tag{10}$$

The elements $b_{ij}$ of the *second fundamental form* matrix $\mathbf{B}$ of the surface are given by

$$b_{11} = L = \mathbf{x}_{uu} \cdot \mathbf{n} = \frac{f_{uu}}{\sqrt{1 + f_u^2 + f_v^2}} \tag{11}$$

$$b_{12} = b_{21} = M = \mathbf{x}_{uv} \cdot \mathbf{n} = \frac{f_{uv}}{\sqrt{1 + f_u^2 + f_v^2}} \tag{12}$$

$$b_{22} = N = \mathbf{x}_{vv} \cdot \mathbf{n} = \frac{f_{vv}}{\sqrt{1 + f_u^2 + f_v^2}}. \tag{13}$$

The principal curvatures $\kappa_1$ and $\kappa_2$ are the roots of the equation

$$\|\mathbf{G}\|\kappa_n^2 - (g_{11}b_{22} + d_{11}g_{22} - 2g_{12}d_{12})\kappa_n + \|\mathbf{B}\| = 0 \tag{14}$$

where $\|\mathbf{G}\| = \det \mathbf{G}$ and $\|\mathbf{B}\| = \det \mathbf{B}$. The mean curvature $H$ and the Gaussian curvature $K$ at a point are defined to be

$$H = \frac{\kappa_1 + \kappa_2}{2} = \frac{1}{2}[\operatorname{trace}(\mathbf{G}^{-1}\mathbf{B})] \tag{15}$$

$$K = \kappa_1 \kappa_2 = \frac{\|\mathbf{B}\|}{\|\mathbf{G}\|}. \tag{16}$$

The feature vector $\mathbf{x} = (x, y, z; n_x, n_y, n_z; H; K)$ was selected to impose *homogeneity* constraints on the vector quantization, but generally speaking, segmentation via feature vector clustering suffers from one significant deficiency viz. pixels for which the feature vectors belong to a single cluster in the feature space may not be spatially connected in the image. The following three aspects of our approach directly address this deficiency:

1. The position coordinates $x$ and $y$ are selected as features in the feature vector in an effort to preserve spatial connectivity.
2. The use of the SOFM as the means of achieving partitioning alleviates the spatial connectivity problem to a great extent due to the topology preserving property of the SOFM. The SOFM, however, does not guarantee the preservation of the exact connectivity relationship between clusters or regions.
3. Traversal of the abstraction tree (as described in

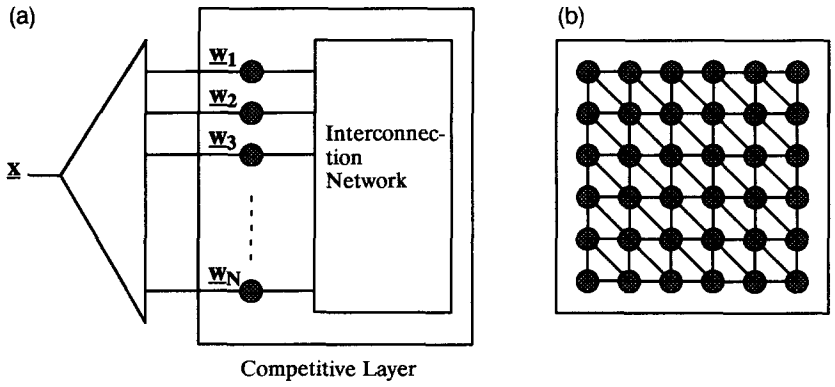



**FIGURE 1. The network structure of SOFM. (a) Overall structure. (b) Interconnection (an example).**

Section 4.4) ensures that one can determine the *connected* components in the segmented image.

## 2.2. Vector Quantization on the SOFM

The SOFM is a popular choice for implementing vector quantization using neural networks (Ritter & Schulten, 1986a,b, 1987; McDermott & Katagiri, 1988; Martinelli, Ricotti, & Ragazzini, 1990; Kohonen et al., 1987; Visa, 1990). Many examples of the use of the SOFM for vector quantization can be found in the literature. Naylor and Li's (1988) work concerns the use of the SOFM to generate a vector quantization codebook for speech recognition. Nasrabadi and King (1988) have considered the use of the SOFM for vector quantization for image compression. They compare the use of the SOFM to the Linde–Buzo–Gray (LBG) algorithm (Linde, Buzo, & Gray, 1980), which is a deterministic iterative vector quantization algorithm wherein the final codebook vectors are dependent on the initial codebook. Matsuyama has considered the use of the SOFM for vector quantization in speech and image processing (Matsuyama, 1988).

A typical SOFM structure is shown in Figure 1. It consists of two layers,[1] a layer of input nodes and a competitive layer consisting of neural units called Kohonen's units (Kohonen, 1982). A weight vector is associated with each connection from the input layer to a neural unit. The neural units in the competitive (and cooperative) layer are organized in a regular geometric structure such as a two-dimensional mesh. The units are interconnected with their local neighbors and these connections could be excitatory.

Let $\mathbf{x}$ be an input vector to the SOFM. The competitive phase of the learning algorithm employed in the SOFM determines a *winning* neural unit whereas the cooperative phase of the learning algorithm updates the weights of the winner and the neural units in its neighborhood. The learning algorithm for the SOFM could be described as follows:

1. The distances between the input vector $\mathbf{x}$ and all the reference vectors (i.e., the weight vectors) are computed using a prespecified distance measure.
2. A *winner* (i.e., a neural unit for which the corresponding weight vector is at a minimum distance from the input vector) is determined.
3. The weight vectors corresponding to the winner and the neural units in its topological neighborhood are updated to align them towards the input vector.

Steps 1 and 2 comprise the competitive phase of the learning algorithm whereas step 3 comprises the cooperative phase. Steps 1 through 3 of the learning algorithm are carried out for each input vector that is presented to the SOFM. The learning rate is defined to be the constant of proportionality, indicating the extent to which the weight vectors are adjusted or updated towards a given input vector. The learning rate (updating rate) and the size of neighborhood are reduced as the learning progresses.

One of the important properties of the SOFM is that it represents a *topology preserving mapping* or a *topographic* mapping. The location of the winning unit in the 2-D mesh of neural units comprising the competitive layer conveys some information about the input vector. Winning units that are proximate in the competitive layer correspond to input vectors that are proximate in the input vector space. If $\mathbf{x}_1$ and $\mathbf{x}_2$ are two input vectors with corresponding winning units at locations $\mathbf{r}_1$ and $\mathbf{r}_2$ in the competitive layer, then $\mathbf{r}_1$ and $\mathbf{r}_2$ get closer and eventually coincide as $\mathbf{x}_1$ and $\mathbf{x}_2$ are made more and more similar. Conversely, if $\mathbf{r}_1 = \mathbf{r}_2$ one could conclude that the corresponding input vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ are similar. A topographic mapping thus preserves topological relations in the input space while at the same time performing a dimensionality reduction (i.e., projection) of the input space onto the 2-D mesh of neural units in the competitive layer. A topographic map entails a suitably defined *topological neighborhood* around a winning neural unit in the competitive layer and also a distance metric associated with the topolog-

[1] Usually the input layer is not counted when we specify the number of layers (i.e., this is called a single-layer SOFM).

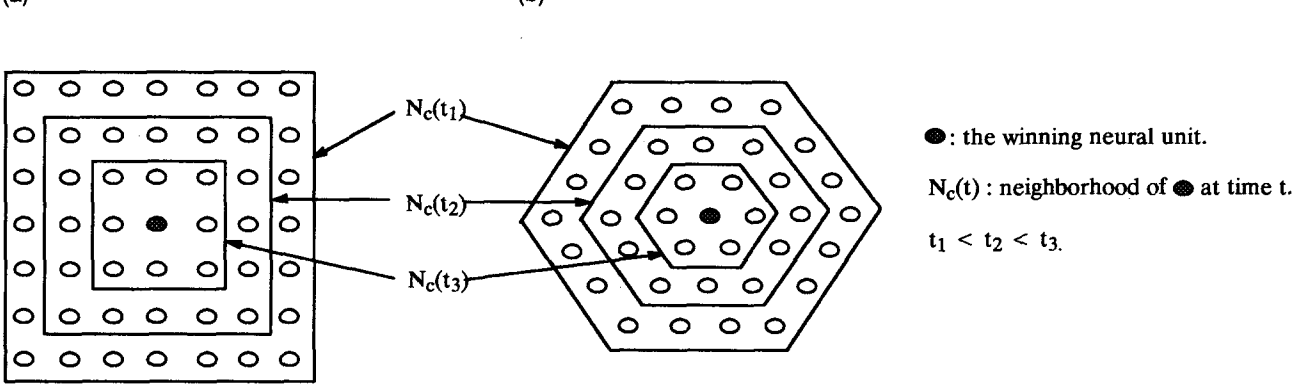



**FIGURE 2. Examples of topological neighborhood. (a) Rectangular. (b) Hexagonal.**

ical neighborhood. Common choices for a topological neighborhood are (a) rectangular and (b) hexagonal as shown in Figures 2a and b, respectively.

The popularity of the SOFM for vector quantization is primarily due to the similarity between the competitive learning process employed in the SOFM and the vector quantization procedure. The competitive learning process in the SOFM produces weight vectors that correspond to distinct clusters of the input vectors. In fact, the weight vectors can be considered to be the cluster centers of the probability density function of the input data. Representing or approximating an input vector by the weight vector associated with the winning unit in the SOFM is equivalent to deriving the minimum-residual-error-approximation to the input vector, which is exactly the same result sought by the vector quantization process. Table 1 shows the correspondence between the two schemes.

### 2.3. SOFM for Range Image Segmentation

Because the SOFM can be used for vector quantization, we examine how it may be used for range image segmentation. Let $R_1, R_2, \ldots, R_M$ be the $M$ regions after segmentation. The SOFM must identify or learn the mapping

$$\mathbf{x}(x, y) \mapsto R_i \tag{17}$$

**TABLE 1**
**Similarities Between SOFM and Vector Quantization**

| SOFM | Vector Quantization |
|---|---|
| Weight vectors | Codebook (reference) vectors |
| Distance measure | Distortion measure |
| Adaptation process {Phase 1 and 2 in learning} | Training process |
| Clustering process {Phase 1 after adaptation} | Encoding process |

for all pixels in the image of size $N_x$ by $N_y$ which minimizes a prespecified criterion. As described previously, the input is presented to the input layer, pixel by pixel, as an eight-dimensional vector $\mathbf{x}_i = (x, y, z; n_x, n_y, n_z; H; K)$, where $(n_x, n_y, n_z)$ is the unit normal vector and $H$ and $K$ are the mean and Gaussian curvatures, respectively, at the point $(x, y, z)$ on the three-dimensional surface. Ideally, on the completion of learning, each neural unit in the competitive layer should be made sensitive to a single region, where the weight vector $\mathbf{w}_j = (w_x, w_y, w_z, w_{nx}, w_{ny}, w_{nz}, w_H, w_K)$ of each unit is the representative vector of a region where $(w_x, w_y, w_z)$ are the coordinates of the center of the region in the three-dimensional space and $(w_{nx}, w_{ny}, w_{nz})$ is the representative unit normal vector of the region. The components $w_H$ and $w_K$ are the representative mean and Gaussian curvature values of the region. The values of $w_H$ and $w_K$ are close to the average of the mean and Gaussian curvature values, respectively, where the average is taken over all the pixels in the region.

The use of the SOFM requires the specification of $M$ beforehand. Ideally, $M$ (and hence the number of neural units in the competitive layer) needs to be chosen very close to the number of regions desired in the final segmentation. However, the number of regions in the segmented image is very much dependent on the scene content and cannot be accurately determined a priori. This makes the use of the original single-layer SOFM for image segmentation rather difficult because once $M$ is set, one does not have direct control over the number of regions resulting from the segmentation. This shortcoming is also characteristic of most clustering algorithms wherein the final number of clusters needs to be specified beforehand. Figure 3b shows what might happen if the SOFM has a fewer number of neural units than the number of visually distinguishable regions in the image whereas Figure 3c shows the case wherein the SOFM has a greater number of neural units than the number of visually distinguishable regions in the image. Figure 3b represents an *undersegmented*

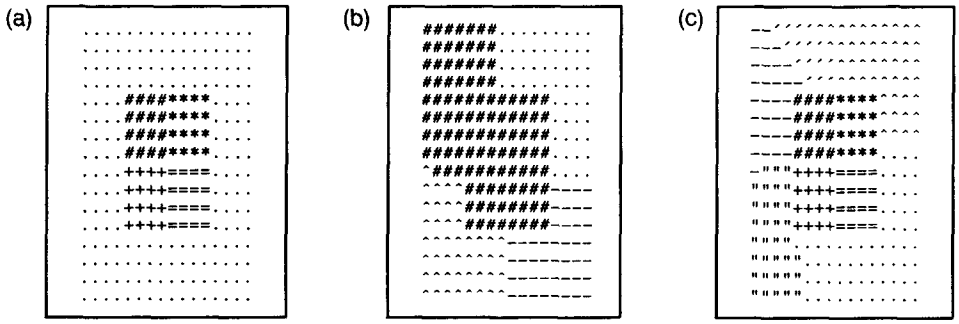



**FIGURE 3. Image segmentation using a single layer SOFM. (a) Input image. (b) Segmented image using four neural units. (c) Segmented image using 16 neural units.**

image whereas Figure 3c represents an *oversegmented* image, both of which are undesirable. We therefore conclude that the original single-layer SOFM is not adequate for range image segmentation.

We propose an extension of the SOFM called the MLSOFM, which consists of multiple SOFM layers. The overall structure is in the form of a pyramid where the sizes of the individual SOFMs become smaller as one moves up to higher levels. Learning in each competitive layer produces a segmented image at a different level of abstraction or feature resolution. Therefore, the output of the MLSOFM, called an *abstraction tree,*[2] is a multiscale or multiresolution segmentation of the input image. As we show in the subsequent sections, the abstraction tree, when properly traversed, can yield a final segmented image at the desired levels of feature resolution locally adapted to the local scene content. The final segmented image produced by the traversal of an abstraction tree consists of different levels of feature resolution in different portions of the image depending on the local scene content. The use of the MLSOFM in image segmentation, thus, does not require the a priori specification of the final number of clusters in the segmented image. This alleviates the problem of oversegmentation or undersegmentation that primarily arises due to a single choice of feature resolution for the entire image irrespective of the local scene content.

### 2.4. Abstraction Tree and Multiscale Image Representation

The process of image segmentation can be interpreted as one of data abstraction. We propose a multilevel hierarchical approach to segmentation that produces a multilevel abstraction of the input image. Initially the input image is segmented into $M_1$ regions using the feature vector clustering algorithm. Here the value of $M_1$ is chosen sufficiently large so that the busiest portion of input image can be adequately represented. The resulting set of segmented regions is the first-level abstraction of the input image. Each segmented region is now represented by its representation vector, and the next level of segmentation is performed using these representation vectors as input, resulting in $M_2$ regions. The segmentation process is repeated at successively higher levels, resulting in $M_1 > M_2 > M_3 > \ldots$ regions until the highest level corresponding to the trivial case consisting of a single region covering the entire input image is reached. The partitioning of the input image at each level can be considered as an abstraction of the input image at that level. The abstraction is performed in the feature space, that is, different levels of abstraction correspond to the clustering of feature vectors at different scales. Once two regions (or two pixels at the input image scale) are merged into a single region at the $l$th level, they will continue to remain in a single region at any level $l' > l$. We can, therefore, construct a tree called an *abstraction tree* containing all levels of abstraction, as shown in Figure 4a. Each node in the tree represents a subregion at a particular level of abstraction.

The abstraction tree is related to, but is more general than, the conventional multiscale structures such as *pyramids,* multiresolution images, *multiscale* images, or *scale-space* images (Witkin, 1983; Mokhtarian & Mackworth, 1986; Lu & Jain, 1992; Dyer, 1987). A multiscale image representation can be achieved by the MLSOFM as a special case by selecting the image intensity and the two-dimensional spatial location as components of the input vector. In conventional multiscale structures the different scales of abstraction represent different scales of the image space. The multilevel abstraction characterized by the abstraction tree is more general in the sense that different levels of abstraction represent different scales of the *feature vector space,* not merely the image space.

The abstraction tree representation of an input image can be used in many different ways by high-level vision processes. In situations where it is necessary to generate a segmented image, a segmented image can be generated by traversing the abstraction tree in a

[2] The abstraction tree will be defined in the next section.

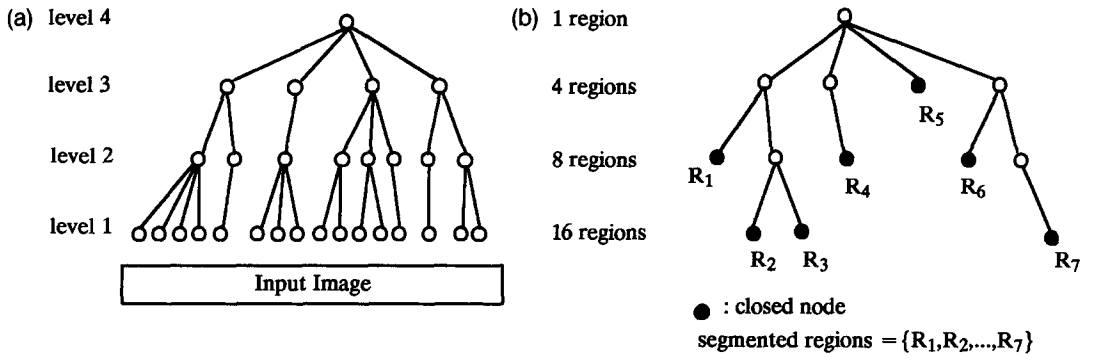



**FIGURE 4. Abstraction tree. (a) A three-level abstraction tree. (b) A tree corresponding to the segmented image.**

breadth-first manner from the root node; that is, a node is expanded if the variances of range values, mean curvature magnitude values, and unit normal vectors of the pixels in the regions is larger than a threshold value. Otherwise the node is labelled as a closed node and not expanded any further. Note that the resulting segmented image usually contains regions from different abstraction levels. Regions corresponding to the *closed* nodes in the abstraction tree represent the final segmented image, as shown in Figure 4b. Also note that one can define a more sophisticated criterion for expanding the nodes in the abstraction tree depending on the nature of the image. In other situations, we may regard the abstraction tree itself as the input to high-level vision. In such cases, the abstraction tree represents the highest domain-independent abstraction of the input data. For example, we are currently experimenting with the idea of using the abstraction tree as the basis for hypotheses generating sets for three-dimensional object recognition (Koh, 1994).

We will describe in the next section how such an abstraction tree can be generated by using the MLSOFM. We note that the image segmentation using a single-layer SOFM would produce an abstraction of the input image at a *prespecified* (and somewhat arbitrarily chosen) abstraction level. On the other hand, the segmented image produced by the MLSOFM would consist of regions belonging to different levels of abstraction adapted to local scene content. This local adaptation is highly desirable because different levels of abstraction are needed for different portions of the image depending on the local scene content. In the final segmented image, portions of the image where the scene attributes are largely homogeneous (e.g., constant depth or similar curvatures) are mapped to the neural units at higher levels of abstraction in the MLSOFM, whereas those portions of the image where the scene attributes vary greatly and rapidly (i.e., busy regions in the image[3]) are mapped to the neural units at lower levels of abstraction in the MLSOFM. The MLSOFM thus represents a significant improvement over the single-layer SOFM in the context of image segmentation because the MLSOFM is capable of adapting the level of abstraction in accordance with the local scene content.

[3] The busy regions represent those portions of the image where the feature values are highly nonuniform, so they must be decomposed into several segments.

## 3. MULTILAYER SELF-ORGANIZING FEATURE MAP (MLSOFM)

The MLSOFM consists of multiple layers, each layer comprised of an SOFM. The number of units in each layer decreases at successive levels, resulting in a pyramidal structure. The number of representative vectors to be generated in each layer is proportional to the number of neural units in the layer. Thus, in a layer at a higher level that has a fewer number of neural units, each weight vector represents a larger cluster. Hence, the representation produced at a higher level in the MLSOFM corresponds to a higher level of abstraction of the input data, thus making the MLSOFM well suited for hierarchical range image segmentation.

### 3.1. Computation of the Input Vectors

The input feature vector for every pixel in an image needs to be computed before segmentation of the image by the MLSOFM. As described in Section 2.1, the input feature vector is given by the 8-tuple $\mathbf{x}_i = (x, y, z, n_x, n_y, n_z, H, K)$, where $(x, y)$ are the image coordinate values, $z$ the range value, $(n_x, n_y, n_z)$ the unit normal vector, and $H$ and $K$ the mean and Gaussian curvature values, respectively. The surface normal and curvature values can be directly computed from a range image using window operators, which are implemented as fixed-weight feedforward neural networks as described below.

3.1.1. *Computation of the Normal Vector.* The unit normal vector, $\mathbf{n_p}$, at a surface point $\mathbf{p}$, is perpendicular to the tangent plane at that point. It can be computed as the normalized cross-product of the two first-order derivatives of a range vector with respect to the two orthogonal $\mathbf{u}$ and $\mathbf{v}$ axes as

$$\mathbf{n}_p = \frac{\frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v}}{\left\| \frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v} \right\|}. \quad (18)$$

The first-order derivatives are estimated by two $5 \times 5$ equally weighted least-squares derivative estimation operators, $D_u$ and $D_v$,

$$D_u = \mathbf{d}_0 \cdot \mathbf{d}_1^T, \quad (19)$$

$$D_v = \mathbf{d}_1 \cdot \mathbf{d}_0^T, \quad (20)$$

where

$$\mathbf{d}_0 = \tfrac{1}{5}[1, 1, 1, 1, 1]^T, \quad (21)$$

$$\mathbf{d}_0 = \tfrac{1}{5}[-2, -1, 0, 1, 2]^T. \quad (22)$$

3.1.2. *Computation of Surface Curvatures.* The mean and Gaussian curvatures can be computed using the first- and the second-order derivative estimation operators. Besl and Jain (1988) obtained good results using a single $7 \times 7$ Gaussian smoothing filter and five $7 \times 7$ equally weighted least-squares derivative estimation operators in their experiments. However, our experience has shown that a size of $7 \times 7$ for the window operators is too large, resulting in an *edge effect* (i.e., the distortion of the true normal and curvature values in the vicinity of jump and crease edges). Moreover, large window sizes are computationally expensive even with array processors. Therefore, we use an alternative technique for estimating the surface curvatures (Ittner & Jain, 1985; Hoffman & Jain, 1987), where the surface curvature is estimated by approximating the derivative of the surface normal. The simple estimate of the surface curvature at pixel $\mathbf{p}$ in the direction of pixel $\mathbf{q}$ is given by

$$k(\mathbf{p}, \mathbf{q}) = \frac{\|\mathbf{n}_p - \mathbf{n}_q\|}{\|\mathbf{p} - \mathbf{q}\|} \times s(\mathbf{p}, \mathbf{q}), \quad (23)$$

where

$$s(\mathbf{p}, \mathbf{q}) = 1, \quad \text{if} \quad \|\mathbf{p} - \mathbf{q}\| \leq \|\mathbf{n}_p - \mathbf{n}_q\|, \quad (24)$$

$$s(\mathbf{p}, \mathbf{q}) = -1, \quad \text{otherwise.} \quad (25)$$

$\mathbf{n}_p$ and $\mathbf{n}_q$ are the unit normal vectors at points $\mathbf{p}$ and $\mathbf{q}$, respectively. This definition makes two assumptions: (1) the underlying surface is smooth and (2) $\mathbf{q}$ is close enough to $\mathbf{p}$ on the surface so that the arc length can be adequately approximated by the Euclidean distance $\|\mathbf{p} - \mathbf{q}\|$. $s(\mathbf{p}, \mathbf{q})$ is a sign factor that states that if the two surface normals, $\mathbf{n}_p$ and $\mathbf{n}_q$ at pixels $\mathbf{p}$ and $\mathbf{q}$, respectively, approach each other as $\mathbf{q}$ approaches $\mathbf{p}$ then the surface curvature has a negative value, indicating a concave surface, else, the surface curvature has a positive value, indicating a convex surface. Using the value of $k(\mathbf{p}, \mathbf{q})$ one can compute the mean and Gaussian curvatures. Let $\Omega(\mathbf{p})$ be the set of pixels in the neighborhood of pixel $\mathbf{p}$. The mean curvature, $H$, and Gaussian curvature, $K$, are given by

$$H = \frac{(k^{\min}(\mathbf{p}) + k^{\max}(\mathbf{p}))}{2}, \quad (26)$$

$$K = (k^{\min}(\mathbf{p}) \times k^{\max}(\mathbf{p})), \quad (27)$$

where

$$k^{\min}(\mathbf{p}) = k(\mathbf{p}, \mathbf{q}_0) = \min_{\mathbf{q}=\Omega(\mathbf{p})} k(\mathbf{p}, \mathbf{q}), \quad (28)$$

$$k^{\max}(\mathbf{p}) = k(\mathbf{p}, \mathbf{q}_1) = \max_{\mathbf{q}=\Omega(\mathbf{p})} k(\mathbf{p}, \mathbf{q}). \quad (29)$$

We use a $3 \times 3$ neighborhood (i.e., only eight nearest-neighbor pixels are considered) when computing the mean and Gaussian curvatures.

The mean and Gaussian curvatures can be computed by a combination of MAXNETs, MINNETs, and fixed-weight networks, as shown in Figure 5a. Each pixel provides as input its position vector and the unit normal vector computed as shown previously. Each curvature computing unit receives these inputs from a center pixel and one of its eight nearest neighbors, and computes a curvature value. The MAXNET and MINNET collect the curvature values to find a maximum and a minimum curvature value, respectively. The last units—the multiplying unit and the averaging unit—compute the $H$ and $K$ values using the maximum and minimum curvature values from the MAXNET and MINNET, respectively. We refer to this structure for computing $H$ and $K$ values as an *HK* column, as shown in Figure 5a. An *HK* column receives input from a square $3 \times 3$ region from the input image. The number of *HK* columns equals the number of pixels in an input image. The overlapping square regions constitute the predefined receptive fields of the *HK* columns (Figure 5b).

It is to be noted that the preprocessing involves only estimating the values of $H$ and $K$ at a range image pixel. The range image pixel is not classified as belonging to a particular qualitative surface type (i.e., spherical, cylindrical, etc.) based on the values of $H$ and $K$, as is the case with Besl and Jain (1988) or Hoffman and Jain (1987).

### 3.2. Structure of the MLSOFM

The MLSOFM consists of multiple competitive layers (Figure 6). The structure of a typical competitive layer is shown in Figure 7. The input layer receives input from the external world and propagates the input to all neural units in the first competitive layer. Competitive learning takes place and the resulting weights are converted and propagated to the next layer as input to that layer. The same process is repeated until the top layer is reached. A higher layer contains a smaller number of neural units than a lower layer. Without loss of generality, we can assume that the input is a square image of size $N_f$ by $N_f$, and neural units in each competitive

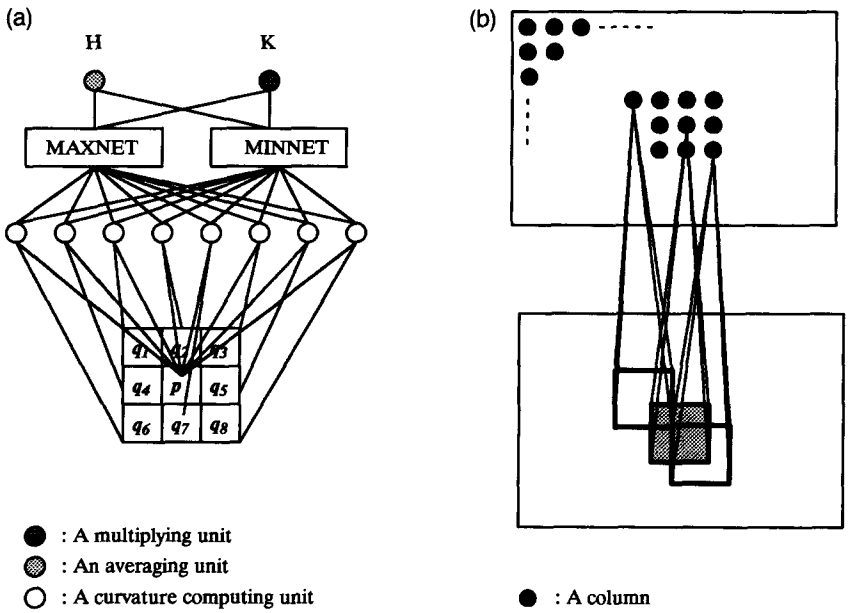



**FIGURE 5. Neural network implementation for computing *H* and *K*. (a) An *HK* column for computing *H* and *K*. (b) Receptive fields of the *HK* columns.**

layer are organized into square arrays. Typically, the size of competitive layers are $1 \times 1, 2 \times 2, 4 \times 4, \ldots,$ $N \times N$ where $N = N_f/2 = 2^n$ is the size of the first layer.

This structure, on the surface, appears to be similar to that of the quadtree. However, there are some significant differences between the two. The MLSOFM structure is much more flexible than the quadtree with regard to the parent–child node relationship and the shape of the resulting regions. In the quadtree, the mapping between the parent and the child nodes is fixed; a parent node has four child nodes, each of which covers a square area of the input image at a specified location. A homogeneous region of the input image that spans two such squares would be split unnaturally in a quadtree. Thus, the image regions in a segmented image obtained by a segmentation algorithm that uses the quadtree data structure (such as a split-and-merge algorithm) tend to have a *boxy* appearance. On the other hand, the mapping between the nodes (i.e., neural units) in two adjacent layers of the MLSOFM is not fixed, but it is supposed to be *learned,* in the fashion of preserving the topology of the input. Therefore, a parent node can have an arbitrary number of child nodes, thereby adapting to the topology of the input (i.e., the scene content). The shape of the resulting region represented by a node in the MLSOFM can thus

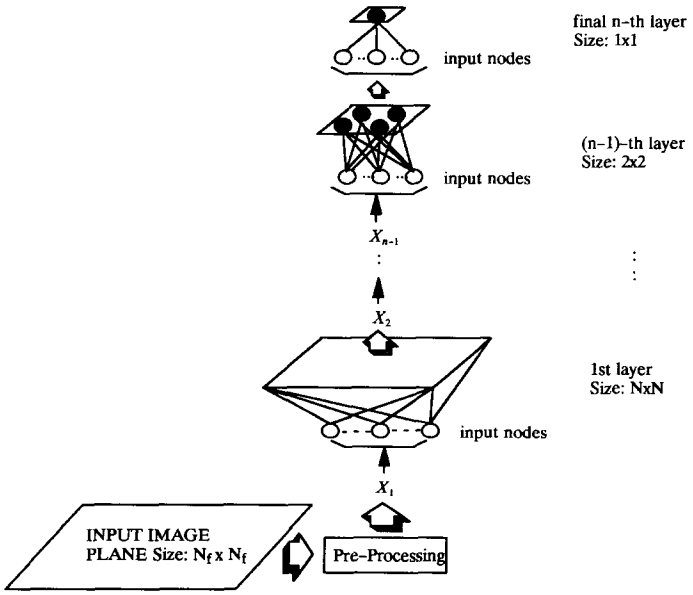



**FIGURE 6. The structure of the multilayer SOFM.**

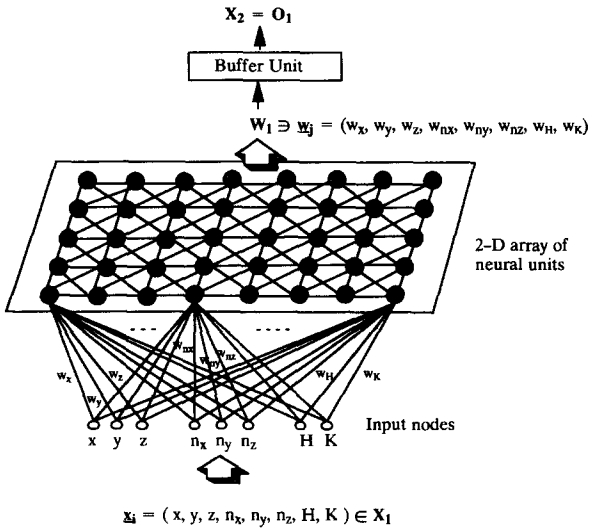



**FIGURE 7. A typical competitive layer (the first layer).**

follow more closely the natural shape of the region boundary.

### 3.3. Information Flow in the MLSOFM

Input data arrives at the lowest layer and information flows to the higher layers in a strictly feedforward manner. Output from any given layer is converted into the input for the next layer by a buffer unit. Let $\mathbf{x}_i = (x, y, z, n_x, n_y, n_z, H, K)$ be an input vector and $\mathbf{X}_1$ be the set of input vectors $\mathbf{X}_1 = \{\mathbf{x}_i, i = 1, 2, \ldots, N_f^2\}$. The set $\mathbf{X}_1$ is the input space for the first layer. Let $\mathbf{W}_1$ be the set of weight vectors for the first layer, $\mathbf{W}_1 = \{\mathbf{w}_j = (w_x, w_y, w_z, w_{nx}, w_{ny}, w_{nz}, w_H, w_K), j = 1, 2, \ldots, N^2\}$. On the completion of training, $\mathbf{W}_1$ contains the representative vectors corresponding to the first level of abstraction of the input vectors. Let $\mathbf{O}_1 \subseteq \mathbf{W}_1$ be the set of weight vectors corresponding only to the winning neural units. $\mathbf{O}_1$ is used as the input vector set for the second layer (i.e., $\mathbf{X}_2 = \mathbf{O}_1$). Only the weight vectors of the winning units are used as the input to the next layer because these are the only correct representative vectors. Other neural units are adapted to interpolate the values of the winning units. This is called the ordering property of the SOFM, as shown by Kohonen (1988) for the one-dimensional case. The interpretation of $\mathbf{X}_i$, $\mathbf{W}_i$, $\mathbf{O}_i$, $i = 2, 3, \ldots, n$ is straightforward. Note that the sizes of $\mathbf{W}_i$ are fixed, but the sizes of $\mathbf{X}_i$ and $\mathbf{O}_i$ cannot and need not be predetermined because the number of winning units is determined via the learning process and is hence dependent on the input scene content.

For an $n$-layer MLSOFM, the input $\mathbf{X}_1$ is mapped to $\mathbf{O}_n$ by the mapping $\Pi_{i=1}^{n} \mathbf{TM}_i$, where $\mathbf{TM}_i$ is the topological mapping performed by the $i$th layer SOFM: $\mathbf{TM}_i : \mathbf{X}_i \mapsto \mathbf{O}_i$.

### 3.4. The Learning Procedure in the MLSOFM

Learning in each layer repeats the following two phases for each input vector: the first phase determines the winning neural unit for weight adjustment and the second phase adjusts the weight vectors of units in the neighborhood of the winner. The learning scheme employed in our range image segmentation algorithm incorporates three modifications to the original learning algorithm proposed by Kohonen (Kohonen, 1988): (1) separate weights are assigned to the position vector, the unit normal vector, and the two curvature values in feature space, (2) the *conscience* method as proposed by DeSieno (DeSieno, 1988; Hecht-Nielsen, 1987) is incorporated in the first phase of the learning procedure, and (3) the scheme proposed by Ritter and Schulten (1988) for adapting the parameters for weight adjustment is incorporated in the second phase. The *conscience* method aims at achieving faster convergence and improved representation of the input data. For further details on the conscience method, the interested reader is referred to the paper by DeSieno (DeSieno, 1988).

3.4.1. *Weighted Euclidean Distance Measure.* There are four heterogeneous pieces of information in a feature vector: the position vector, $(x, y, z)$, the unit normal vector, $(n_x, n_y, n_z)$, and the curvature values, $H$ and $K$. To control the extent of the contributions from each of these four components of information, the Euclidean distance measure used in the SOFM is modified. A *weighted Euclidean distance measure,* which is a weighted sum of four separate Euclidean distance measures (one between the position vectors, one between the unit normal vectors, and one each from the two curvature values) is defined:

$$d(\mathbf{x}_i, \mathbf{w}_j) = a \cdot \|\mathbf{x}_p - \mathbf{w}_p\| + b \cdot \|\mathbf{x}_n - \mathbf{w}_n\| + c \cdot |H - w_H| + d \cdot |K - w_K|, \quad (30)$$

where $0 \le a, b, c, d \le 1$,

$$\mathbf{x}_i = (x, y, z, n_x, n_y, n_z, H, K) = (\mathbf{x}_p; \mathbf{x}_n; H; K),$$

$$\mathbf{x}_p = (x, y, z),$$

$$\mathbf{x}_n = (n_x, n_y, n_z),$$

$$\mathbf{w}_i = (w_x, w_y, w_z, w_{nx}, w_{ny}, w_{nz}, w_H, w_K) = (\mathbf{w}_p; \mathbf{w}_n; w_H; w_K),$$

$$\mathbf{w}_p = (w_x, w_y, w_z),$$

$$\mathbf{w}_n = (w_{nx}, w_{ny}, w_{nz}).$$

With this distance measure, we can selectively control the relative dominance of the contribution of each component. The actual weights used in our experiments are given in Section 4.3. The same set of values for $\{a,$

$b, c, d\}$ was used for all the images that we experimented with.

3.4.2. *Conscience Mechanism.* The *conscience* mechanism attempts to overcome certain limitations of simple competitive learning.

1. The presence of a bias that favors regions with lower densities of input vectors during the process of dividing an input vector space into equiprobable regions.
2. The underutilization of neural units. Attempts to bring nonwinning processing elements into the solution tend to increase the number of iterations required for convergence.

The idea behind the conscience mechanism is to make a frequently winning neural unit feel *guilty* and make it restrain itself from winning excessively. The conscience mechanism keeps track of how frequently each neural unit wins the competition. By using this information, weight vectors of neural units can be modified approximately an equal number of times. The goal of the conscience mechanism is to bring all the neural units into the solution quickly and to bias the competition so that each neural unit has an equal chance of winning the competition for optimal vector quantization. A speed-up in convergence of almost 10-fold is reported to have been achieved by adding the conscience mechanism to the SOFM (DeSieno, 1988). The conscience mechanism has also proved to be effective in practice for developing a set of equiprobable features or prototypes for representation of the input data. The conscience mechanism is a major improvement upon the simple competitive learning proposed by Kohonen.

Other alternatives to simple competitive learning have been described in the literature. Grossberg (1976a,b, 1987) has suggested a scheme that would force a losing neural unit to become more sensitive, and a winner to become less sensitive. Another alternative approach is the FSCL (frequency-sensitive competitive learning), suggested by Ahalt et al. (1990). We have used the FSCL in our multilayer segmentation algorithm for image coding (Kim et al., 1993). Although the FSCL training method yields lower distortion than the *conscience* mechanism (Ahalt et al., 1990), the FSCL training method does not guarantee the spatial neighborhood relationship between segmented regions. This is not crucial in the case of a segmentation algorithm for which the goal is image coding but is crucial in the case of a segmentation algorithm for which the output would be used for high-level cognitive tasks, such as object recognition.

In the *conscience* mechanism, the Euclidean distance measure, which is used in most SOFM networks, is modified to include the effects of the frequency of winning. The modified distance measure is obtained by multiplying the squared Euclidean distance by the frequency of winning. The winning flag $z_i$ of a neural unit $i$ whose weight vector is at a minimum distance from a given input vector is set for adjusting the weight vectors. All the flags are cleared after adjusting the weight vectors and before the next input. The winning flag $z_i(t)$ is modified as

$$z_i(t) = 1, \quad \text{if} \quad d_i(t) = d(\mathbf{x}, \mathbf{w}_i(t)) - b_i(t) \leq d_j(t) = d(\mathbf{x}, \mathbf{w}_j(t)) - b_i(t) \quad \forall j \neq i, \tag{31}$$

$$z_i(t) = 0, \quad \text{otherwise}, \tag{32}$$

where $t$ is the time step, $\mathbf{x}$ an input vector, $\mathbf{w}_i(t)$ a weight vector for the $i$th neural unit, $b_i(t)$ the bias term used to modify the competition, and $d_i(t)$ the distance measure between the weight vector of the $i$th neural unit and the input vector. The bias term $b_i(t)$ is defined as

$$b_i(t) = C\left(\frac{1}{N} - p_i(t)\right), \tag{33}$$

where the constant $C$ is the conscience bias factor, $N$ the number of neural units in the competitive layer, and $p_i(t)$ the fraction of time that the $i$th neural unit has been the winner until time $t$. $p_i(t)$ is defined as

$$p_i(t) = p_i(t-1) + B(y_i(t) - p_i(t-1)), \tag{34}$$

where $0 < B \ll 1$ and $y_i(t)$ is the output of the $i$th neural unit, which is 1 iff the $i$th neural unit is the winning unit and is 0 otherwise. The constant $B$ should be chosen small enough so that the value of $p_i(t)$ does not reflect the random fluctuations in the data. We used the values $B = 10^{-4}$ and $C = 10.0$ in our experiments.

We note that for a frequently winning neural unit, the modified distance measure $d(t)$ between the weight vector of the unit and the input vector is increased by the presence of the bias $b(t)$. This reduces the likelihood of that neural unit becoming a winner again, and consequently increases the chance of other neural units winning. We also note that by setting $C = 0$, the conscience mechanism degenerates to Kohonen's learning, which can therefore be considered to be special case of the conscience mechanism. The learning rate in the case of the conscience mechanism is the same as that in the case of Kohonen's learning.

3.4.3. *Weight Vector Adjustment.* After a winning neural unit is determined, the weight vectors of the neural units in its neighborhood are adjusted as follows:

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + lr(t) \cdot h_{rs}(t) \cdot (\mathbf{x}(t) - \mathbf{w}_j(t)), \tag{35}$$

$$lr(t) = lr_{\text{init}} \cdot \left(\frac{lr_{\text{final}}}{lr_{\text{init}}}\right)^{t/t_{\max}}, \tag{36}$$

$$h_{rs}(t) = \exp\left(-\frac{\|\mathbf{i} - \mathbf{j}\|^2}{2 \cdot \sigma^2(t)}\right), \tag{37}$$

$$\sigma(t) = \sigma_{\text{init}} \cdot \left( \frac{\sigma_{\text{final}}}{\sigma_{\text{init}}} \right)^{t/t_{\max}}, \tag{38}$$

where **i** is the location vector of a winning neural unit, **j** is the location vector of a neighboring neural unit, and $lr(t)$ is the learning rate. The parameters in the above equations were set empirically. The learning rate $lr(t)$ is decreased from $lr_{\text{init}}$ set to 0.8 to $lr_{\text{final}}$ set to $10^{-3}$ so that Kohonen's learning rule ensures final convergence to the asymptotic values. In an MLSOFM layer of size $N \times N$, the parameters, $\sigma_{\text{init}}$, $\sigma_{\text{final}}$, and $t_{\max}$ are set as follows: the value of $\sigma_{\text{init}}$ is set to $N_{\max}/2$ and the $\sigma_{\text{final}}$ to 0.1 where $N_{\max}$ is the maximum neighborhood radius. Most researchers have used and recommended a maximum initial neighborhood size of $(N \times N)/4$. Accordingly, we have set the value of $N_{\max} = N/2$. As the learning progresses, the lateral width $h_{rs}$, which redefines the neighborhood range, is slowly decreased to a single winning neural unit to make the sensitivities of the neural units more localized as the learning progresses.

The same learning procedure is used in every layer, with the initial weight vectors set to random values with a small variance around the average of the input vectors. In each competitive layer, the two phases of learning are applied repeatedly a sufficient number of times to ensure that the computed error for all input vectors lies within an acceptable range. In this case, the real distance measure (i.e., the Euclidean norm between a weight vector and an input vector) is taken to reflect the error. The value of $t_{\max}$, which was experimentally determined, was selected to be large enough to ensure that the computed error fell within the acceptable error range for all the range images. The value of $t_{\max}$ was set to $100 \times (N_{\max} + 1)$.

### 3.5. Neighborhood Topology and Topology Preserving Mapping

A major difference between the SOFM and other competitive learning neural networks is that a neural unit in the SOFM has a topological neighborhood associated with it and all the neural units in the neighborhood of a winning unit are updated along with the winning unit. The idea of associating a topological neighborhood with each neural unit in the SOFM has its origins in the study of the lateral interaction between cells of the two-dimensional neural layers in the mammalian brain. The fact that all the neural units in the topological neighborhood are updated along with the winning unit is the major reason behind the topological ordering property of the SOFM, as illustrated by a simple proof in Kohonen (1988). An appropriate definition of a topological neighborhood is therefore crucial for the SOFM to function as a topology preserving (i.e., topographic) mapping.

The topological structure of the neighborhood of the neural units determines the dimension of the topological ordering of input vectors. In general, for a single-layer SOFM, either four, six, or eight neighbors are used. Defining the neighborhood topology is very important in image segmentation using the single-layer SOFM, because the neighborhood topology determines the final topology of mapping and thereby the shape of the resulting regions. The topological neighborhood relationship in the original single-layer SOFM can be found only in the interconnection topology of the neural units. If two neural units share a connection, the corresponding regions mapped on to those two neural units are adjacent. For example, let us consider six regions $\mathbf{r}_i$, $i = 1, 2, \ldots, 6$ neighboring a region **r** as shown in Figure 8. The topological mapping cannot be preserved by a single mapping if we choose the *four-nearest-neighbors* neighborhood topology. As shown in Figure 8, $\mathbf{r}_4$ and $\mathbf{r}_6$ are not mapped as neighbors of **r**. But the use of the hexagonal neighborhood topology, for this particular example, would map all six regions onto neural units that are direct neighbors of the neural unit corresponding to **r**, thus resulting in a topologically correct mapping.

The MLSOFM shares the property of being a topology preserving mapping with the original SOFM. In a single layer of the MLSOFM, the topological neighborhood relationship depends on the topology of the interconnection of the neural units just as in the case of the original SOFM. However, the choice of the

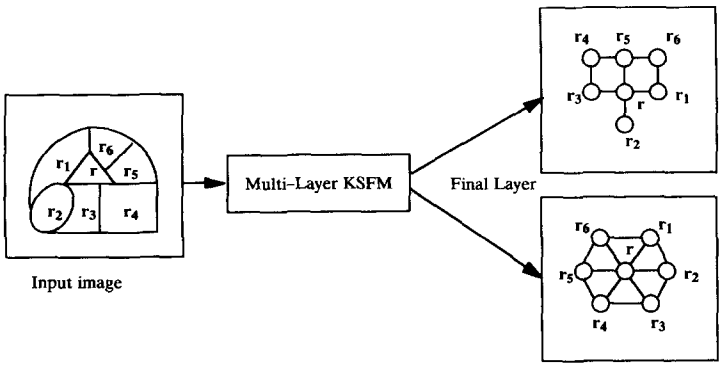



**FIGURE 8. The effect of neighborhood topology on the mapping.**

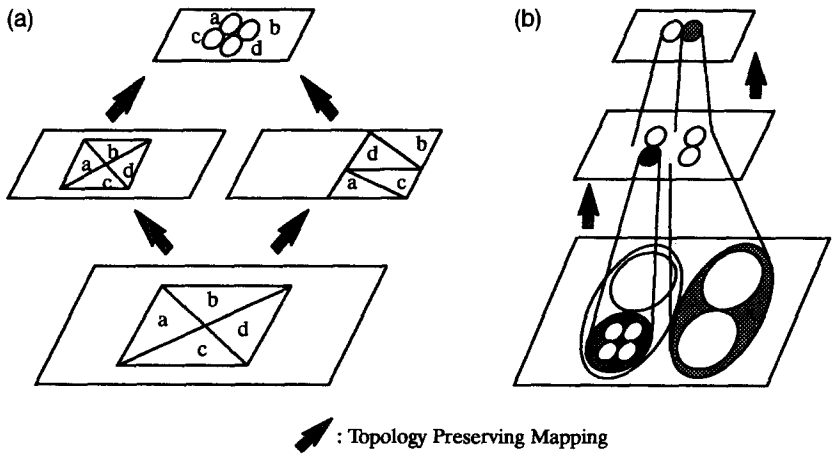



**FIGURE 9. Topological ordering over variable sizes of elements in each layer. (a) Maintaining topological neighborhood. (b) Resolution of topological ordering.**

neighborhood topology is not as critical for the MLSOFM as it is for the SOFM. This is so because some of neighborhood information is encoded by the abstraction tree in the sense that the child nodes of the same parent node have a higher probability of being topological neighbors. In the final layer, which contains a single neural unit, the topological neighborhood relationship cannot be defined either in the interlayer structure or in the intralayer structure. For range image segmentation, we used a circular neighborhood, where the size of the neighborhood was made a function of $((i-k)^2+(j-l)^2)^{1/2}$, where $(i, j)$ and $(k, l)$ are the positions of the neural units in the competitive layer.

One of the significant features of the MLSOFM is that we can also consider different resolutions of topological ordering. The size of input vectors on which the topological ordering is defined is different at each layer. A neural unit at a higher layer represents a coarser region (i.e., the size of the region represented by that neural unit is larger). Thus, the size of the regions represented by the vectors mapped to a neural unit belonging to the closed set defined by the topological ordering relation varies with each layer. This is illustrated in Figure 9. Figure 9a shows examples of a topologically correct mapping. In the first layer, both region $a$ and region $d$ are each adjacent to both region $b$ and region $c$, and conversely. Two regions are considered adjacent if they share a common boundary. The regions $a$, $b$, $c$, and $d$ can be mapped in a topologically correct manner in two ways, as depicted in the second layer. These regions are mapped to four groups of neural units that are direct neighbors, thus preserving the topological relationship in the first layer. In the third layer, these four regions are finally mapped to four neural units that have the same neighborhood relationship as the four regions in the first layer. Figure 9b shows the different resolutions of topological ordering. The four neural units represented by the four smallest circles in the first layer are mapped to a single neural unit in the second layer. Since the four neural units share the same parent node in the abstraction tree, they are considered adjacent. The four neural units in the second layer, each of which represents four neural units in the first layer, are topological neighbors. The two groups of two neural units each in the second layer are mapped onto two neural units in the third layer, which are topological neighbors. Thus, different resolutions of topological ordering can be explored in each layer. The four neural units in the second layer preserve the neighborhood relationship of the four medium-size circles whereas the two neural units in the third layer preserve the neighborhood relationship of the two largest circles.

## 4. EXPERIMENTAL RESULTS AND DISCUSSIONS

### 4.1. Simulation

The MLSOFM was simulated on the Connection Machine (CM-2).[4] The speed-up achievable by the massively parallel computer is limited because the SOFM learning procedure is inherently sequential (i.e., based on examining one input vector at a time). On completion of learning in a given layer, the input for the next layer is extracted from the winning neural units. The geometry of the CM processors is changed to that of the next layer and the learning process is repeated. We do not wish to imply that the CM is the best choice for simulating a SOFM or MLSOFM. In fact, we have not tried to optimize the implementation in any way whatsoever. Our sole purpose in this paper was to demonstrate the utility of the MLSOFM, both in concept and practice, in the context of range image segmentation. It is clear that any implementation scheme pro-

[4] For the details on the implementation, see Koh (1994).

posed for the original SOFM is also, to a fair extent, applicable to the MLSOFM, and we refer the interested reader to previous works for implementation details (Ahalt et al., 1990; Ritter & Schulten, 1987; Ritter & Schulten, 1988).

### 4.2. Input Images

Real range images[5] containing the following objects were used in our experiments.

1. *Grnblk:* This object consists of planar surfaces (i.e., this is a polygonal object).
2. *Column:* This object consists of a cylinder and a cube joined together.
3. *Taperoll:* This object is a simple tapered roll containing both convex and concave cylindrical surfaces.
4. *Cone:* This object is a cone.
5. *Half-sphere:* This object is a hemisphere.
6. *Agpart:* This object consists of three different size tapered rolls and contains both convex and concave cylindrical surfaces with different radii. This is a synthetic image.
7. *Curvblock:* This object consists of planar surfaces and a concave curved surface.
8. *Agpart + block:* In this scene the *Agpart* occludes a toy block.
9. *Box + cap:* In this scene a plastic cap occludes a cube.
10. *Cup + block:* In this scene a cup occludes a toy block.

The size of input images vary from 160 × 80 pixels to 240 × 240 pixels. The real range images were obtained from the Pattern Recognition and Image Processing Laboratory at Michigan State University, East Lansing, MI and they were produced by the Technical Arts 100× White scanner. The White scanner is a triangulation-based laser range sensor, which is the reason why some of the real range images used in our experiments suffer from shadows.

### 4.3. Configuration

The first competitive layer in the MLSOFM has neural units arranged as a 16 × 16 array, the second layer 8 × 8, the third layer 6 × 6, the fourth layer 4 × 4, the fifth layer 3 × 3, and the topmost layer has 2 × 2 neural units. The attentive reader will notice the discrepancy between the size of the layers in the MLSOFM as mentioned in Section 3.2 and in the CM-2 implementation mentioned here. In Section 3.2 we had mentioned that the size of the competitive layer at the $i$th level in the MLSOFM was $2^{(n-i-1)} \times 2^{(n-i-1)}$. The reason for this discrepancy is that because the range images used in this experiments did not contain many distinct regions, the first layer was limited to a small number of neural units (i.e., 16 × 16). The third and the fifth layers were added to ensure accurate merging at higher abstraction levels.

The weights used in Euclidean distance were determined experimentally. During the course of our experiments, the objects in the input images were observed to have relatively simple surfaces so that the value of the Gaussian curvature was not critical. Thus, the contribution of the Gaussian curvature to the distance measure was deemphasized; that is, the weight $d$ in the weighted Euclidean distance measure was chosen to be very small. The magnitude of the mean curvature was found to be much more effective than the mean curvature values themselves. So, the magnitude of the mean curvature was used instead of the mean curvature value. The weight ratio $a : b : c : d = 1 : 10^{-2} : 10^{-1} : 10^{-5}$ was used for the weighted Euclidean distance measure all throughout in our experiments.

### 4.4. Results and Discussions

We carried out a series of experiments using synthetic and real range images to see the effects of various image characteristics on the segmentation produced by the MLSOFM. A set of synthetic range images for which the scene contents were similar to those of the real range images were created. The use of synthetic images was convenient during the development phase of the MLSOFM because we knew precisely the ideal outcome. Also, because the real images contained the same objects that we dealt with in the synthetic images, the segmentation results on the real images could be readily evaluated by comparing them to those obtained on the corresponding synthetic images. Some of the real images contained multiple objects that enabled us to evaluate the performance of the MLSOFM on scenes containing partially occluded objects whereas others contained a very complex single object that enabled us to evaluate the performance of the MLSOFM on an image containing several surface regions. The real images that we used contained various types of surfaces, such as planar, cylindrical, conical, and spherical surfaces. This paper shows only the experimental results using real range images.

To illustrate the segmentation process as it proceeds in each successive layer of the MLSOFM, the intermediate results of the synthetic image *Taperoll* are shown in Figure 10. The first image shows the input range image and the rest of the images illustrate the intermediate segmentation results for each successive layer in the MLSOFM starting with the first layer. For convenience of display, the range values are converted to intensity values between 0 and 255. Also, the segmentation results from each of the layers are mapped back to the pixel locations in the input image in a man-

[5] The only exception is *Agpart*.

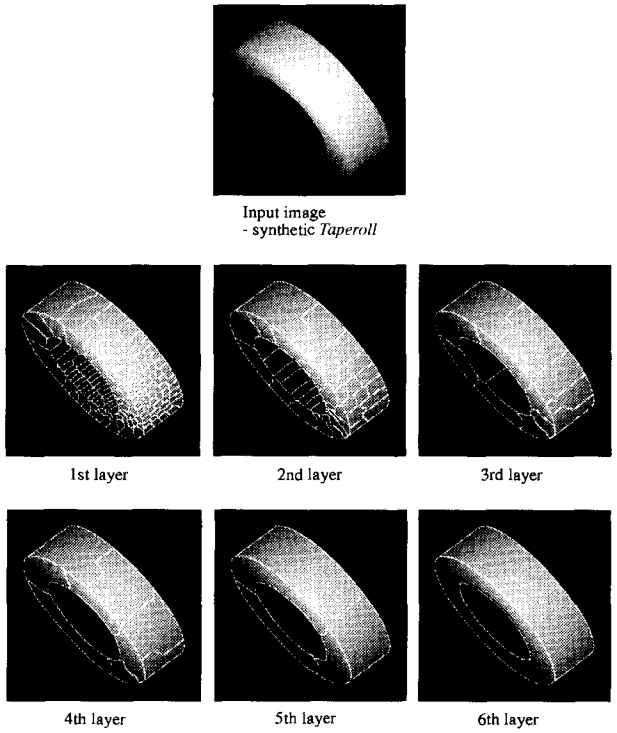


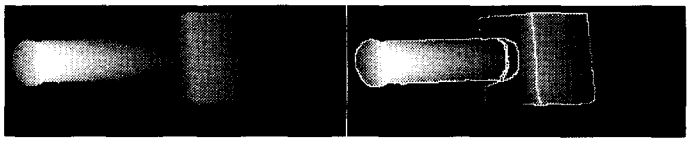

**FIGURE 10. An example of multiresolution segmentation using the synthetic *Taperoll* image. (White boundaries between pixels mapped onto different neural units are inserted for the purpose of display.)**

ner such that the image pixels mapped to a common neural unit on completion of learning form a region enclosed by white boundaries. We note that, in the first layer, the planar surface in the image is segmented into small polygonal regions whereas the cylindrical surfaces are segmented into several longitudinal strips. Although every point on a planar surface has zero $H$ and $K$ values and the same (ideally speaking) unit normal vector value, the position vector term in the weighted Euclidean distance is seen to dominate the minimum variance grouping. Likewise, although every point on the cylindrical surfaces has the same nonzero magnitude for $H$ and a zero value for $K$, the reason for the segmentation of the cylindrical surfaces into longitudinal strips is that surface points on a single longitudinal strip have identical or similar unit normal vector values. In the MLSOFM, layers at higher levels of abstraction, the polygonal segments in the case of planar surface regions and the longitudinal strips in the case of cylindrical surface regions, are merged together within their respective surface regions.

Figure 11a shows the final segmentation result for the *Column* image. The segmentation result of *Column* shows the successful segmentation of three planar surfaces of which two constitute adjacent faces of a cube and which, although spatially adjacent in the image plane, have different unit normal vectors. The third planar surface region is seen to be far apart in the image plane from the other two. The cylindrical surface is easily discriminated because its two neighboring planar surface regions have zero $H$ and $K$ values whereas the cylindrical surface has a nonzero value for $H$. The real range images from Michigan State University do not provide valid range data for the background. Even if they did, the background can be easily separated from the objects because the background is typically far behind the objects. That is to say, in a typical range image, background pixels have a much higher depth (range) value than the pixels on the object surfaces. When the background is broken into a few regions, these regions can be linked by a trivial chaining algorithm because they have the same depth value. Also note a crack near the boundary between the cylindrical surface and the cube due to the lack of data (i.e., shadows) in the input image.

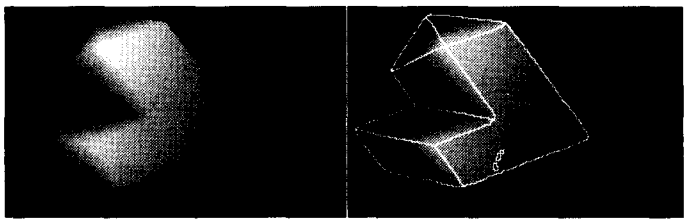
The segmented image was generated by traversing the resulting abstraction tree in a breadth-first manner from the root node, that is, a node was expanded if the sum of the variances of the range values, mean curvature magnitude values, and the unit normal vectors of the pixels in the region was larger than a threshold value. Otherwise, the node was labelled as a closed node and not expanded any further. When the one of the variances in the sum is larger than the threshold value, the pixels in the region cannot be regarded as belonging to a region that has homogeneous surface characteristics. Thus, they should be divided into several regions at a lower abstraction level. When no single variance value is larger than the threshold but the sum of the variances exceeds the threshold, the pixels in the region should also be divided. This is because the feature vectors corresponding to the pixels in the region are far too scattered in the multidimensional vector space for the pixels to be considered as belonging to a single region with homogeneous surface characteristics. Details on the threshold selection can be found in Koh (1994). One could define a more sophisticated criterion than the one mentioned here for expanding the nodes in the abstraction tree depending on the nature of the image and the scene contents. The resulting segmented image usually contains surface regions at different levels in the abstraction tree. However, because with the images we used in our experiments, all the closed nodes occurred at the same level of abstraction, the abstraction tree in its entirety was redundant. Also note that, for many applications, an entire abstraction tree providing a relational description may itself be considered the final outcome of the segmentation process. The entire abstraction tree may be directly propagated as input to the high-level vision processes instead of a single segmented image.

The segmentation results of other range images are shown in Figure 11b–j. We observed that the segmentation of planar surfaces could be accomplished easily in a few MLSOFM layers because the clustering is de-



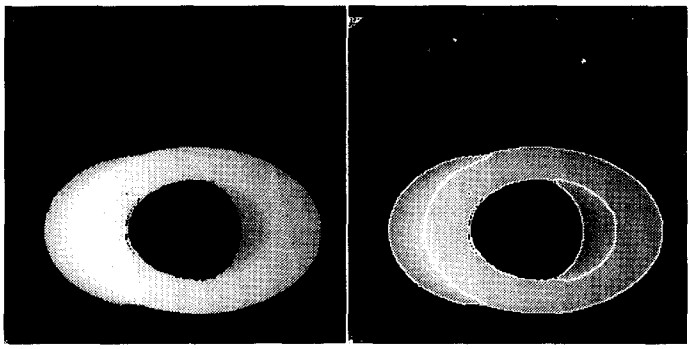
Input image - *Column* Segmented image

(a)



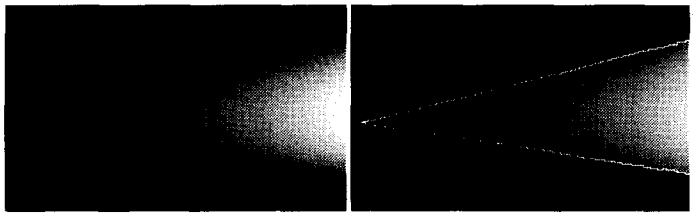
Input image - *Grnblk* Segmented image

(b)



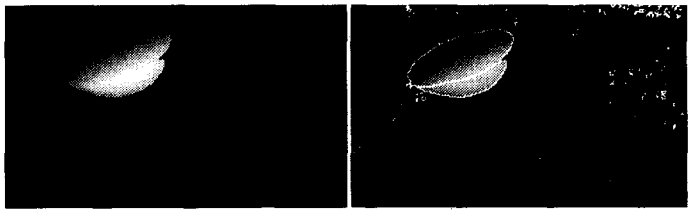
Input image - *Taperoll* Segmented image

(c)



Input image - *Cone* Segmented image

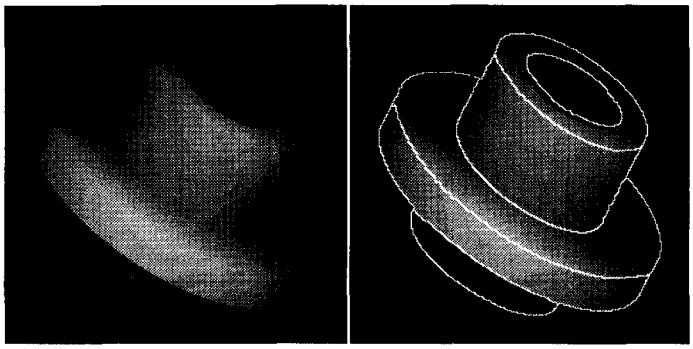
(d)



Input image - *Half-sphere* Segmented image

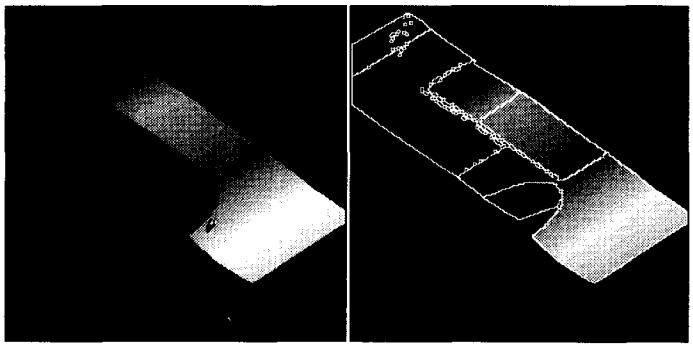
(e)

**FIGURE 11. Segmentation results for diverse types of input range images. (a) Column. (b) Grnblk, (c) Taperoll, (d) Cone, (e) Half-Sphere, (f) Agpart, (g) Curvblock, (h) Agpart + block, (i) Box + cap, (j) Cup + block. (White boundaries between pixels mapped onto different neural units are inserted for the purpose of display.)**



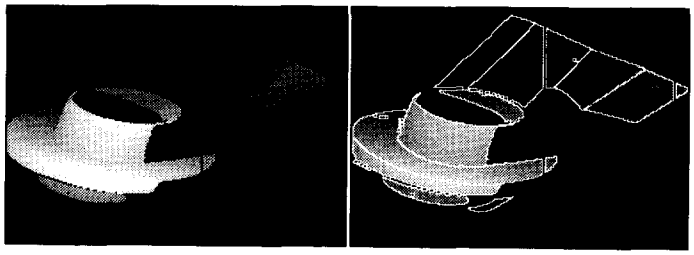
Input image - *Agpart* Segmented image

(f)



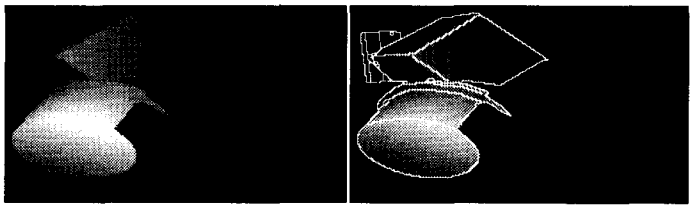
Input image - *Curvblock* Segmented image

(g)



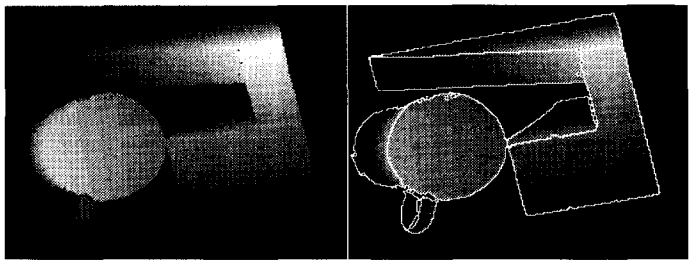
Input image - *Agpart+block* Segmented image

(h)



Input image - *Box+cap* Segmented image

(i)



Input image - *Cup+block* Segmented image

(j)

termined largely by the position vector term in the weighted Euclidean distance measure. This can be readily observed from the segmentation result of the synthetic range image of *Grnblk*. However, the segmentation of the real range image of *Grnblk* took more MLSOFM layers. This was because of the presence of noise in the real range data that led to the inevitable fragmentation of a single planar surface into several regions. We also observed from the results of the real range image *Taperoll* that cylindrical surfaces were typically divided into longitudinal strips at the lowest layer of abstraction in the MLSOFM and that these strips were successively merged into larger surface patches as one proceeded to higher layers of abstraction—a phenomenon observed in the case of synthetic cylindrical surfaces as well.

From the segmentation results of the *Cone* image and the *Half-sphere* image, we noted that the MLSOFM was capable of segmenting conical and spherical surfaces correctly. The conical surface in the *Cone* image was divided into *fan-like* regions at the lowest level of abstraction in the MLSOFM. These fan-like regions were then merged together in successive layers of the MLSOFM. The reason for conical surfaces to be divided into fan-like segments is that the points along the contours of these segments have the same (or similar) values for the surface normal. The relatively larger weight assigned to the mean curvature magnitude term in comparison to the weight assigned to the unit normal vector term in the weighted Euclidean measure (Section 4.3), as well as the multilayer structure of the MLSOFM, ensured smooth merging of subregions belonging to a cylindrical or spherical region.

During the segmentation of scenes containing complex objects such as *Agpart* and *Curvblock*, which have a greater number of planar, cylindrical, and curved surfaces than others, we observed that the segmentation of planar surfaces was, typically, accomplished faster than that of cylindrical and curved surfaces. Thus, the number of MLSOFM layers needed to successfully segment a range surface is, broadly speaking, an increasing function of the complexity of underlying surface. As expected, planar surfaces being the least complex (because all points on a planar surface have identical unit normal vectors and zero $H$ and $K$ values) were seen to be the easiest to segment. The segmentation process of the objects, *Agpart* and *Curvblock*, was observed to end successfully at the fifth layer as expected because these objects have more than four surfaces each.

As examples of multiple-object scenes, we experimented with three scenes, *Agpart + block*, *Box + cap*, and *Cup + block* in all of which, one object is seen to occlude the other. To delineate the boundaries between surface regions, the edges between these regions were marked by white lines in the image that displays the final result of the segmentation process. The final results show that the objects in the scene have been well delineated. The small noise-like circular regions in the input (real) range images and the two lines dividing the objects in the *Agpart + block* range image were due to the lack of data at certain pixel locations in the input range images (i.e., due to shadows or noise).

The above experiments positively demonstrated the utility of the MLSOFM for range image segmentation, both in concept and in practice. The performance of the MLSOFM was compared to that of a well-known segmentation algorithm by Leonardis et al. (1990) in terms of oversegmentation, misclassification, and number of missed pixels. Leonardis' algorithm, based on surface fitting, appears to suffer from oversegmentation, misclassification, and a greater number of missed pixels compared to the MLSOFM. The MLSOFM also demonstrated less sensitivity to the choice of parameters. Details of the experimental comparison can be found in Koh (1994).

## 5. CONCLUSIONS

This paper considered the use of neural networks for an important low-level computer vision problem, namely, range image segmentation. Since the problem of image segmentation can be considered to be one of vector quantization, it is natural to consider the use of the SOFM. However, it was seen that the original single-layer network was inadequate for image segmentation. The network was extended to include multiple competitive layers that were organized as a pyramid. The new extended network, which we have termed as the MLSOFM, supports range image segmentation very well, producing various levels of abstraction in the feature space. The result of the segmentation process, which includes all intermediate levels of abstraction, can be organized as an *abstraction tree*. The abstraction tree can be used for many different purposes in high-level vision. One application of the abstraction tree is to produce segmented images. Many synthetic and real range images were used in our experiments to study the effects of different types of objects and various image characteristics such as translation, rotation, and scaling on the final segmentation. The experimental results were very satisfactory in demonstrating the utility of the MLSOFM for range image segmentation. Although the immediate concern of this paper was range image segmentation using the MLSOFM, the MLSOFM can be used in any application that calls for clustering of vectors in feature space. The advantages of the MLSOFM in the general context of clustering of vectors in feature space can be summarized as: (1) the number of clusters need not be specified a priori, and (2) one obtains a multilevel abstraction of clusters in feature space.

## REFERENCES


Ahalt, S. C., Krishnamurthy, A. K., Chen, P., & Melton, D. E. (1990). Competitive learning algorithms for vector quantization description of curved objects. *Neural Networks,* **3**(3), 277–290.

Besl, P. J., & Jain, R. C. (1988). Segmentation through variable order surface fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **10**(2), 167–192.

Bilbro, G., White, M., & Snyder, W. (1987). Image segmentation with neurocomputers. In R. Eckmiller & C. von der Malsburg (Eds.), *Neural computers, NATO ASI series: Vol. F*41 (pp. 71–79). New York: Springer–Verlag.

Brady, M., Ponce, J., Yuille, A., & Asada, H. (1985). Describing surfaces. *Computer Vision, Graphics, Image Processing,* **32,** 1–28.

Daily, M. J. (1989). Color image segmentation using Markov random fields. *IEEE Proceedings of Computer Vision and Pattern Recognition Conference* (pp. 304–312). San Diego: IEEE.

DeSieno, D. (1988). Adding a conscience to competitive learning. *Proceedings of IEEE the Second International Conference on Neural Networks* (*ICNN*88) (pp. I:117–I:124).

Dyer, C. R. (1987). Multiscale image understanding. In L. Uhr (Ed.), *Parallel computer vision* (pp. 171–213). New York: Academic Press.

Fan, T. G., Medioni, G., & Nevatia, R. (1987). Segmented description of 3-D surfaces. *IEEE Journal of Robotics and Automation,* **3**(6), 527–538.

Flynn, P. J., & Jain, A. K. (1991). CAD-based computer vision: From CAD models to relational graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **13**(2), 114–132.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **6**(6), 721–741.

Grossberg, S. (1976a). Adaptive pattern classification and universal recording: I. Parallel development and coding of neural feature detectors. *Biological Cybernetic,* **23,** 121–134.

Grossberg, S. (1976b). Adaptive pattern classification and universal recording: II. Feedback, expectation, olfaction, illusions. *Biological Cybernetic,* **23,** 187–202.

Grossberg, S. (1987). Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science,* **11,** 23–63.

Grossberg, S., & Mingolla, E. (1987). A neural network architecture for preattentive vision: Multiple scale segmentation and regularization. *Proceedings of IEEE the First International Conference on Neural Networks* (*ICNN*87) (pp. IV:177–IV:184).

Haralick, R. M., Watson, L. T., & Laffey, T. J. (1983). The topographic primal sketch. *International Journal of Robotics Research,* **2**(1), 50–72.

Hecht–Nielsen, R. (1987). Counterpropagation networks. *Applied Optics,* **26**(23), 4979–4984.

Hoffman, R., & Jain, A. K. (1987). Segmentation and classification of range images. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **9**(5), 608–620.

Hurlbert, A., & Poggio, T. (1989). A network for image segmentation using color. In D. S. Touretzky (Ed.), *Advances in neural information processing systems* (pp. 297–304). San Mateo, CA: Morgan Kaufmann Publishers.

Inokuchi, S., Nita, T., Matsuday, F., & Sakurai, Y. (1982). A three-dimensional edge-region operator for range pictures. *Proceedings of 6th International Conference on Pattern Recognition* (pp. 918–920).

Ittner, D. J., & Jain, A. K. (1985). 3-D surface discrimination from local curvature measures. *Proceedings of Computer Vision and Pattern Recognition Conference* (pp. 119–123).

Kim, N. C., Hong, W. H., Suk, M., & Koh, J. (1993). Segmentation using a competitive learning neural network for image coding. *Proceedings of IEEE International Joint Conference on Neural Networks* (*IJCNN*93) (pp. III:2203–III:2206).

Koh, J. (1994). MLSOFM for image segmentation. Doctoral dissertation, Syracuse University.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics,* **43,** 59–69.

Kohonen, T. (1988). Self-organization and associative memory (2nd ed.). Berlin: Springer–Verlag.

Kohonen, T., Torkkola, K., Shozakai, M., Kangas, J., & Vent, O. (1987). Microprocessor implementation of a large vocabulary speech recognizer and phonetic typewriter for Finnish and Japanese. In J. A. Laver & M. A. Jack (Eds.), *Proceedings of European conference on speech tech.* (pp. 377–380).

Langridge, D. J. (1984). Detection of discontinuities in the first derivatives of surfaces. *Computer Vision, Graphics, Image Processing,* **27,** 291–308.

Lee, D., & Pavlidis, T. (1988). Edge detection through residual analysis. *Proceedings of Computer Vision and Pattern Recognition Conference* (pp. 215–222).

Leonardis, A., Gupta, A., & Bajcsy, R. (1990). Segmentation of range images as the search for geometric parametric models. *Proceedings of International Conference on Computer Vision* (pp. 121–125).

Lin, W., Tsao, E., & Chen, C. (1991). Constraint satisfaction neural networks for image segmentation. In T. Kohonen, K. Mkisara, O. Simula, & J. Kangas (Eds.), *Artificial neural networks* (pp. 1087–1090). New York: Elsevier Science Publishers.

Linde, Y., Buzo, A., & Gray, R. M. (1980). An algorithm for vector quantizer design. *IEEE Transactions on Communication,* **28**(1), 84–95.

Lu, Y., & Jain, R. C. (1992). Reasoning about edges in scale space. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **14**(4), 450–468.

Marr, D., & Nishihara, H. K. (1978). Visual information processing: Artificial intelligence and the sensorium of sight. *Technology Review,* **81**(1), 2–23.

Martinelli, G., Ricotti, L. P., & Ragazzini, S. (1990). Nonstationary lattice quantization by a self-organizing neural network. *Neural Networks,* **3**(4), 385–393.

Matsuyama, Y. (1988). Vector quantization with optimized grouping and parallel distributed processing. *Neural Networks,* **1**(Suppl.), 36–42.

McDermott, E., & Katagiri, S. (1988). Phoneme recognition using Kohonen networks. *Proceedings of ATR Neural Net Workshop,* July.

Mitiche, A., & Aggarwal, J. K. (1983). Detection of edges using range information. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **5**(2), 174–178.

Mokhtarian, F., & Mackworth, A. (1986). Scale-based description of planar curves and two-dimensional shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **8**(1), 34–43.

Nasrabadi, N. M., & King, R. A. (1988). Image coding using vector quantization: A review. *IEEE Transactions on Communication,* **36**(8), 957–971.

Naylor, J., & Li, K. P. (1988). Analysis of a neural network algorithm for vector quantization of speech parameters. *Proceedings of the 1st Annual INNS Meeting* (pp. 310–315).

O'Neill, B. (1966). *Elementary differential geometry.* New York: Academic Press.

Pentland, A. (1989). Part segmentation for object recognition. *Neural Computation,* **1**(1), 82–91.

Ritter, H., & Schulten, K. (1986a). On the stationary state of Kohonen's self-organizing sensory mapping. *Biological Cybernetics,* **54,** 99–106.

Ritter, H., & Schulten, K. (1986b). Topology conserving mappings for learning motor tasks. In J. S. Denker (Ed.), *Neural networks for computing* (*AIP Conference Proceedings, Snowbird, Utah*) (pp. 151:376–151:380).

Ritter, H., & Schulten, K. (1987). Extending Kohonen's self-organizing mapping algorithm to learn ballistic movements. In R. Eckmiller & C. von der Malsburg, *Neural computers* (pp. 393–406). New York: Springer-Verlag.

Ritter, H., & Schulten, K. (1988). Kohonen's self-organizing maps: Exploring their computational capabilities. *Proceedings of IEEE the Second International Conference on Neural Networks* (*ICNN*88) (pp. I:109–I:116).

Scherf, A., & Roberts, G. (1990). Segmentation using neural networks for automatic thresholding. In S. Rogers (Ed.), *SPIE Conference on Applications of Artificial Neural Networks, Orlando, Florida* (pp. 1294:118–1294:124).

Suk, M., & Koh, J. (1993). Image segmentation using multi-layer Kohonen's self-organizing feature map. *Progress in neural networks* (Vol. 4). Ablex Publishing Co.

Tenorio, M. F., & Hughes, C. S. (1987). Real time noisy image segmentation using an artificial neural network model. *Proceedings of IEEE the First International Conference on Neural Networks* (*ICNN*87) (pp. IV:357–IV:363).

Vemuri, B. C., & Aggarwal, J. K. (1986). Curvature-based representation of objects from range data. *Image and Vision Computing*, May, 107–114.

Visa, A. (1990). Texture boundary detection based on LVQ method. In *Signal processing V, theories and applications* (*The fifth European Signal Processing Conference, Barcelona, September* 18–21) (pp. II:991–II:994).

Witkin, A. P. (1983). Scale space filtering. *Proceedings of the 8th International Conference on Artificial Intelligence* (pp. 1019–1021).

Zerubia, J., & Geiger, D. (1991). Image segmentation using four direction line-processes and winner-take-all. In T. Kohonen, K. Mkisara, O. Simula, & J. Kangas (Eds.), *Artificial neural networks* (pp. 1083–1086). New York: Elsevier Science Publishers.

Zucker, S. W., & Hummel, R. A. (1973). An optimal three dimensional edge operator. *Proceedings of IEEE Conference on Pattern Recognition and Image Processing* (pp. 162–168).