An Approach to Three-Dimensional Scene Databases

Thomas C. Henderson and Chuck Hansen[1]

UUCS 87-013

15 June 1987

Department of Computer Science
University of Utah
Salt Lake City, Utah 84112

Abstract

Current image database research is concerned for the most part with the encoding and processing of two-dimensional images. However, the most successful approach to computer vision is based on 3-dimensional information, organized as either stacks of 2-D images (e.g., the intrinsic images of Barrow and Tennenbaum, or the 2 1/2 dimensional sketch of Marr) or as actual 3-dimensional data (e.g., the Multisensor Kernel System of Henderson). Efficient techniques for two-dimensional image processing have been well-developed over the last few decades and special purpose architectures are now available. However, the study of the organization, processing and analysis of three-dimensional scene data is only just beginning. We describe one approach to the representation of three-dimensional image data and evaluate several computer vision algorithms performed on the data. Finally, we describe how three-dimensional feature operations can be performed as relational database operations.

[1]This work was supported in part by NSF Grants MCS-8221750, DCR-8506393, and DMC-8502115

1. Introduction

Given the vast amount of data in digital images and the high cost of image processing operations, it is quite natural that much effort has been devoted to finding more efficient image processing techniques. However, in the analysis of three-dimensional range data, such techniques can only be used in an artificial way, in that range data is by its very nature misrepresented in a two-dimensional data structure. Our work has been aimed at developing more appropriate representations for three-dimensional data, and the development of concommitant processing techniques for such structures. In this paper, we describe some of our results in this domain.

There are two main problems with using two-dimensional images to store range data. First, the implicit neighborhoods in an image (e.g., 4-neighbors or 8-neighbors) do not capture the actual three-dimensional neighborhood information. That is, even though pixels (i,j) and (i+1,j) neighbor each other in the image, their corresponding points $(x,y,z)_{(i,j)}$ and $(x,y,z)_{(i+1,j)}$ may be very far apart. Thus, the use of special purpose architectures is limited in that consistency checking must always be performed. Second, and more importantly, it is very difficult, and usually impossible, to merge distinct views of a scene (i.e., multiple range images) into one data set. In general, this requires going to a three-dimensional array with all its incumbent inefficiencies. Moreover, if the data are sampled at different resolutions, say from a range finder and a tactile pad, it is very complicated to handle the differences in scale.

It is for these reasons that we have proposed the Multi-sensor Kernel System as an efficient and uniform mechanism for dealing with data taken from several diverse sensors [13, 14]. The system can be logically divided into three major parts: the sensor specification, low-level representation, and high-level modeling. In this paper, we discuss in detail the low-level representation and processing aspects of the system. For details on the high-level modeling, see [16, 18] and for more on sensor specifications through logical sensors, see [15, 17].

Several mechanisms have been proposed as low-level representations for three-dimensional data. In particular, Marr's primal sketch [22], Barrow and Tennenbaum's intrinsic characteristics [3], and to some extent, the region adjacency graph of Pavlidis [26], have all been proposed as a low-level organizational tool for image data analysis. We have shown how the recovery of 3-D information can be usefully organized in the spatial proximity graph [11, 13]. Most features (e.g., surface curvature, surface normal, range, texture, etc.) can be localized in 3-space using current computer vision techniques (see Ballard and Brown for an introduction [2]). Other approaches to the organization of point data include minimal spanning trees [31], relative neighborhood graphs [30], oct-trees [28], and Voronoi triangulations [1]. However, due to the fact that we also use the low-level representation to organize feature space data (with arbitrary dimension), we find that the benefits of the k-d tree outweigh its problems.

2. The Multisensor Kernel System

An overview of the configuration time view of MKS has been given by Henderson and Wu [13]. Figure 1 shows the major components of MKS. The high-level representation is either a feature vector model consisting of characteristic feature values for the object and a covariance matrix describing the uncertainty in sensor measurements or a Hough shape model consisting of a boundary model of the object. The sensor specifications are given in terms of Logical Sensors which characterize the output type of the sensor, alternative ways of producing the data, sensor characteristics and a mechanism for encoding knowledge about the sensor's performance and role in the system. Finally, the low-level

$$\begin{tikzcd}
 & \text{MKS} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
\text{High-Level Representation} & \text{Low-Level Representation} & \text{Sensor Specification}
\end{tikzcd}$$

Figure 1. Organization of MKS

representation is one of either a k-d tree or a spatial proximity graph. Figure 2 shows the details of the low-level operations. The sensors provide the raw data upon which the low-level operations are performed. The operations are requested by the high-level model.

$$\begin{tikzcd}
\text{High-Level Knowledge} & \text{3-D Scene Database} \arrow[l] & \text{Sensors} \arrow[l]
\end{tikzcd}$$

Figure 2. Organization of Data in MKS

Obviously, the system is driven by the arrival of data from the actual sensors. Currently, the data from one or more sensors can be analyzed as a sequence of snapshots of data. Moreover, rather than analyze data continually as it is received, the data is collected in a buffer until the buffer is filled, then the data in the buffer is analyzed. If more data arrives, it can either be discarded, or integrated into existing data structures; however, it is computationally expensive to add data to the existing data structures, since the k-d tree must be completely rebuilt when new data is added.

The sensor provides output vectors which are then formulated into the appropriate k-tuple for low-level processing. Currently, this is just a sequence of (x,y,z) triples each with an associated intensity value and status bit (indicating good or bad data at that point). The k-tuples provided by the formatting step are then built into a k-d tree [9, 13]. This provides an efficient method for recovering the spatial structure of the data. This step is accomplished by querying the k-d tree with each data point in turn and creating a graph linking the m nearest neighbors to each query point. Note that the graph is not necessarily symmetric.

At the heart of MKS are the following two data structures: the k-d tree and the spatial proximity graph. The spatial proximity graph simply takes a set of points and creates a graph whose nodes are the points, and whose edges connect each node to the m nearest neighbors of that node. MKS allows the spatial proximity graph to be generalized by allowing points from any k-dimensional space, and as will be discussed later, this permits one to analyze structure in any feature space chosen by the user. The spatial proximity graph is built quite efficiently in terms of the k-d tree [9] which is built directly from the data (see Henderson [12, 19] for the use of the k-d tree for feature organization).

The k-d tree is a generalized form of the simple binary tree used to achieve Order(nlogn) sorting and searching. Therefore, a k-d tree is a binary tree in which each node represents a subset of the vectors in a set of vectors and a partitioning of that subset [9]. The root of the tree represents the whole set of vectors (in our application, the vectors are (x,y,z) locations in a range image where a surface point was detected). Each nonterminal node has two successor nodes which represent the two subsets classified by the partition. The terminal nodes represent mutually exclusive small collections of the vectors in the set. These data vectors collectively form a partition of the set and are known as buckets. See Bentley [4] for the original definition of k-d trees. The version used here minimizes the expected number of vectors examined during the search for nearest neighbors. This is achieved by appropriately choosing both the discriminator key element and the partition value for each subset, and the number of vectors in each bucket.

Since information provided to a binary choice is maximal when the two alternatives are equally probable, it is equally likely that a vector will be placed on either side of the partition. Hence, irrespective of which key (i.e., which element of the vector) is selected as the discriminator, the median of the marginal distribution of key values serves well as the partition.

The search algorithm can stop searching the subset on the side of the partition opposite the query vector if the partition boundary does not intersect the ball centered at the query vector with radius equal to the dissimilarity to the m^{th} closest vector so far encountered. Consequently, the partition will intersect least the ball for that key which showed the greatest range in values before partitioning.

With these considerations in mind, the optimized k-d tree algorithm chooses at every nonterminal node the key with the largest range in values as the discriminator, and the median of the discriminator key values as the partition. (Upon analysis of performance, the terminal buckets should each contain one record in order to minimize the number of vectors examined.) The average case complexity required to build a k-d tree is of the Order(nlogn), and the m nearest neighbors for a vector query can be found in Order(logn) operations.

The user defines the k-tuple (to be used in low-level processing) as a subset of the n-tuple returned by the class of sensors providing the data to the low-level processing module. These k-tuples serve as the basic organizational element of low-level processing, and all high-level models must be defined either directly in terms of them (as when feature models are defined in terms of particular values for each element) or indirectly in terms of the spatial relations existing between the vectors (as in the Hough shape model). Finally, the user may also define a distance function to be used by the system if the standard ones such as Euclidean distance and Manhattan distance are not desired.

Given the spatial proximity graph representation of a set of points sampled from the surfaces of objects in a scene, the points in the graph can be grouped to find various features of interest in the data. In particular, we describe the following operations:

- Normal Calculation - Many other operations can be performed if the surface normal vector is known at each point,
- 3-D Edge Detection - Most high-level models use some knowledge of the 3-D edges in the scene, and
- Curvature Calculation - In order to get at the features characterizing non-polyhedral objects, it is necessary to determine the curvature (e.g., Gaussion or mean) at each point.

The analysis of the data performed by MKS is essentially a cycle of data collection, data organization, data analysis and environment manipulation. The basic interface to MKS occurs through public functions allowed on the scene database: the k-d tree and spatial proximity graph. As a result, we choose to run in our multi-sensor environment by invoking certain operations on the database. The flow of the execution of the multi-sensor framework is as follows:

1. invocation of a low-level operation on the scene database.
2. Activation of the sensors which acquire the 3-D scene data.
3. Organization of the sensed data into a k-d tree structure.
4. Construction, if necessary, of the spatial proximity graph from the k-d tree.
5. The performance of the required low-level operation.

In the following sections, we describe the low-level operations as well as their complexity and give some examples of their application to three-dimensional data.

3. Range Data

Most computer vision analysis is performed with respect to features derived from the raw intensity or surface point data. Given the enormous amount of data, it is necessary that these operations be performed efficiently and rapidly. Although highly optimized techniques have been developed for image processing, even to the point of making special purpose image processing systems cost effective, the calculation of features from multi-dimensional data and their storage can often be more efficient using tree structures rather than standard image representations. For example, we have shown that edge finding and storing is more efficient using tree methods in the application domain of remotely sensed data [19].

The advent of true three dimensional sensors is changing the processing techniques used in computer vision. Rather than attempt to reconstruct 3-D attributes from 2-D sensory data, researchers are now able to analyze actual 3-D data. The remainder of this chapter will describe 3 dimensional sensing and the sensing environment which we are using and then explain the low-level processing techniques we use to extract features from the raw range data. We will then describe how to construct higher level representations based on the information obtained from the low level processing.

3.1. Range Sensors

Three-dimensional data obtained with visual 3-D sensors can be grouped into two classes: depth map data and Cartesian data. Depth map data refers to values at every pixel as an inverse function of the distance from the sensor to the surface being sampled. This yields an image in which pixels with a high intensity represent a surface point which is closer to the sensor than pixels with a low intensity value. Since the distance increases with the offset angle from the viewing axis, these images tend to be cylindrical in nature unless corrected. Conversely, Cartesian data contains actual 3-D points in world coordinates. Thus, for every pixel, there will be a 3-tuple, (X,Y,Z), returned for every surface point rather than just a single distance value. This results in 3 times as much data to be processed as opposed to depth data.

Visual 3-D range sensors can be grouped into 3 classes: triangulation, time of flight sensors, and phase modulation sensors [21, 27]. In triangulation systems, the scene is illuminated by some means of structured light, possibly a laser or specialized pattern of light. The geometry of the light source with respect to the scene and that of the sensor, typically a video camera, with the respect to the scene is used in conjunction with trigonometric methods to recover depth from the 2-D image returned by the camera. A major draw back with 3-D triangulation systems is that depth information is computed with knowledge of the angle formed by the structured light source and the viewing angle of the camera. This results in shadows being cast on the scene. In time of flight sensors, structured light, typically laser light, is cast on the scene and the amount of time is measured between illumination and detection of that illumination. The time difference is used to compute the depth of the surface reflecting the light. Since depth is recovered as a function of the amount of time it takes the structured light to travel to the surface and back, this type of sensor doesn't suffer from shadows but requires extremely fast hardware if the scene is a relatively short distance from the sensor. This is similar to the methods used in laser-driven surveying devices. Phase modulation sensors obtain depth in a similar fashion but rather than counting time, they perform phase detection similar to radar devices. Moreover, these sensors don't suffer from shadows but resolution is only modulo the wave length or the wave length modulation. The sensoring system we use is the Technical Arts White Scanner which falls into the class of triangulation systems and returns Cartesian data [29].

The White Scanner uses a laser beam which is spread into a plane of light by an oscillating mirror and then directed via a director mirror onto the scene. The director mirror is stepped in user controlled increments of 0.055 degrees which illuminates the scene with the plane of laser light. This plane of light forms a curve on any surface which it intersects. This curve is sampled using a 2-D CCD camera yielding 240 Cartesian sample points, each containing an X,Y,Z value. In the configuration we used, the sensor returns very dense range data on the order of a sample every 0.005 inches for a planar surface normal to the angle of the plane of laser light.

Figures 3 to 8 show some samples of scenes in the Utah Range Database [10].

3.2. Storage Comparisons

Let us now consider the advantages of storing range data and associated features in graph structures. suppose that the range scanner produces an m by n image; let p = mn. Furthermore, suppose there are ap good data points in the image. (Bad data occurs when the range finder fails to compute a range value due to laser or camera shadow or improper relectance.) Let a be in the range [0,1]. Finally, suppose

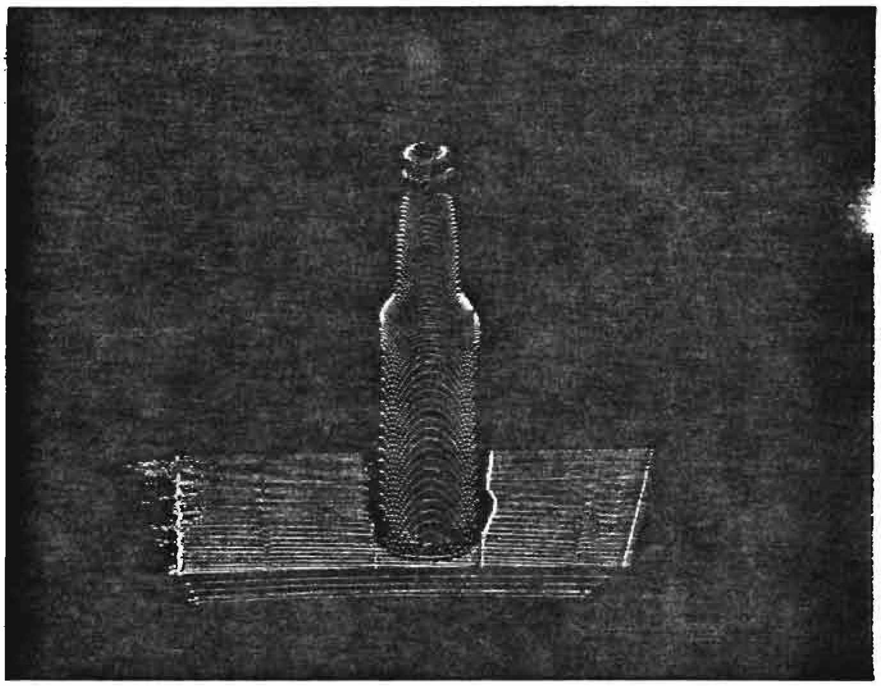

Figure 3. Scene from Range Database: bottle

there are e storage elements required for the x, y, and z values and for the features which are computed; e.g., surface normals, surface curvature, etc.

In an image format, the number of storage elements required is ep. If a k-d tree is used, then there are eap storage elements required for each leaf of the tree; also, there is a (4ap)/b storage element overhead for the nonterminals in the tree, where b is the bucket size or number of records per leaf (assuming 4 storage elements per nonterminal: median value, axis of split, and two pointers). Thus, the k-d tree is more economical whenever:

$$ep > eap + (4ap)/b$$

or

$$e/(e+(4/b)) > a. \qquad (1)$$

It is easily seen that as b increases, the left hand side of Equation (1) approaches 1; thus, the tree structure will almost always be more economical as b gets larger. Even when b=1, the left hand side is e/(e+4) which for the values of e we are considering (e.g., 20) still makes the tree structure more economical when there is 17% or more bad data. Moreover, when b=4, the usual bucket size, the tree structure is more economical when there is 5% or more bad data.

In our experience with the Utah Range Database, we have found that the amount of bad data is a function of scene complexity (essentially due to shadow areas). A summary of the data is given in Table 1. Two percentages are given; in a fixed environment (such as is found in manufacturing), there is no need to analyze the fixed background since we are only interested in recognizing and locating new

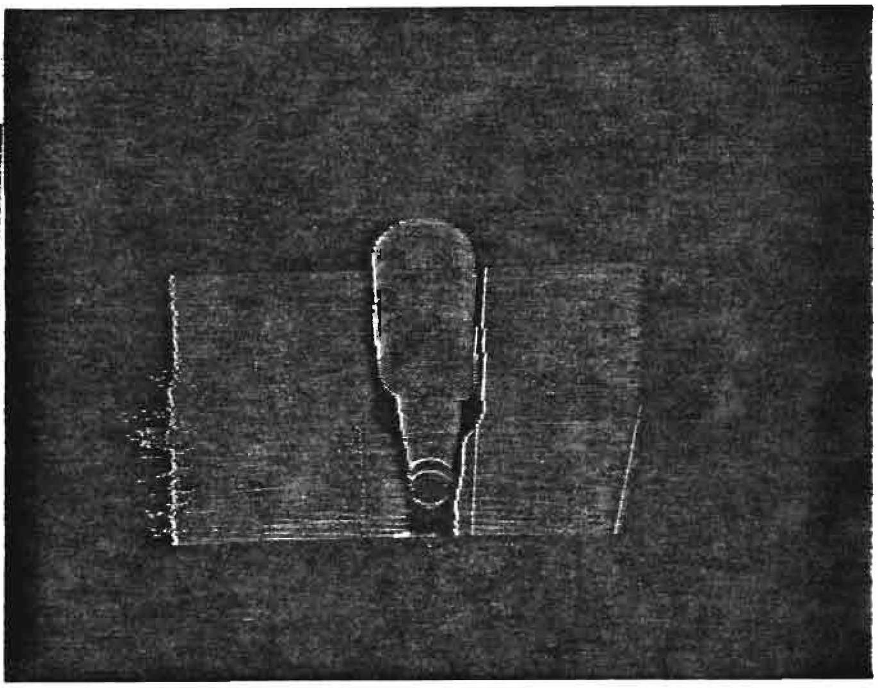

Figure 4. Scene from Range Database: bottle_1

objects in the scene. The first column of data is produced when fixed background points (i.e., points on the underlying workbench) are counted as bad data (i.e., they are not used in the processing step), while the second column gives the percentage of bad data when the fixed background points are included as good data. As can be inferred from the table, the tree storage structure is definitely more efficient.

Of course, feature calculations will most often be performed on the spatial proximity graph and not on the k-d tree in order to get constant time neighbor lookup. Thus, we must know the storage requirements of the spatial proximity graph. Since the spatial proximity graph simply stores the m nearest neighbors of each data point, it requires m(ap) storage elements.

4. Range Data Analysis

Features form the basis or primitive elements of model-based vision and must fullfill three requirements for a recognition system: low cost, reliability, and appropriateness for the particular problem. The cost requirement can be constrained by both time and space. In most computer vision applications, time is generally more critical than storage space, thus time is the primary consideration for the measure of cost. One would like to use features which are computationally efficient to obtain from sensory data. This can be facilitated by use of specialized hardware if it is available.

Features must also have the attribute of reliable detection. This encompasses both detection and false responses. When we say that a feature is reliably detected, we mean that given a known set of data which contains one or more of the features sought, the detector algorithms/hardware should locate those features in the data set. If the algorithms/hardware respond with features being present in the data when,

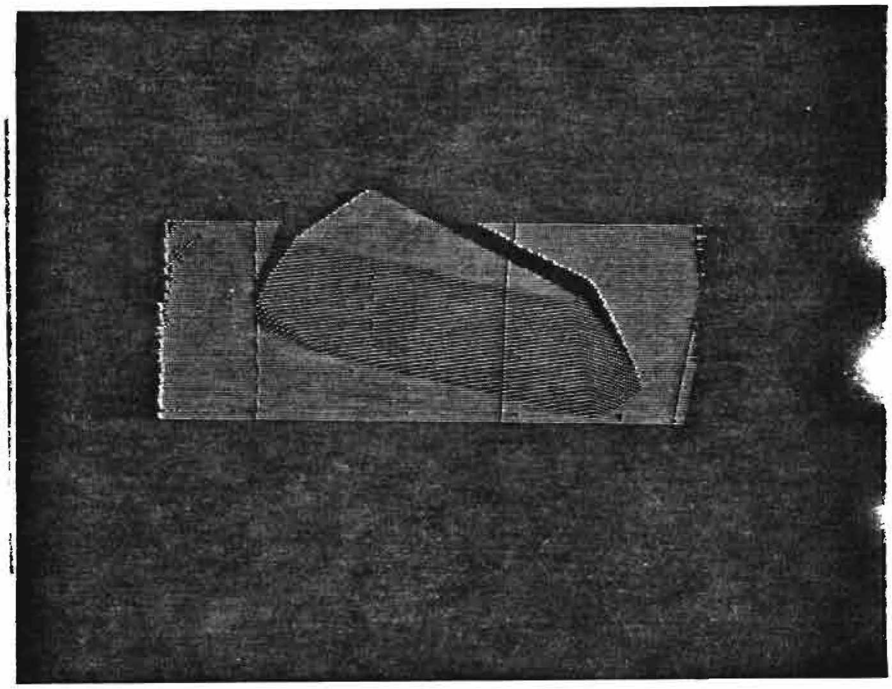

Figure 5. Scene from Range Database: poly_1

in fact, no features are be present, then reliability suffers from false responses. Ideally, we would like our feature detectors to locate only those features which we are looking for and not yield any false-alarms. Realistically, we must strike a balance where we maximize the detected features and minimize the false response rate.

Features must also provide a basis for the recognition process. They must contain enough information to correctly characterize the objects in the scene. For example, corners as features wouldn't provide a basis for the detection of spheres or circles whereas they would for polyhedral objects.

Features must satisfy these requirements simultaneously. If a feature is fast to compute and provides a basis but isn't reliable, it might not be as good as a feature which is reliable but slightly more expensive to compute. Furthermore, there is no single rule for when one should use specific features. This process is application dependent.

Features can be thought of as hierarchical in nature. That is, surface points can be considered features, yet they can be combined to form edges, another type of feature. The properties of a class of features at one level is propagated to the features which use that class. As this one travels up hierarchy, there generally is a compression of the amount of data.

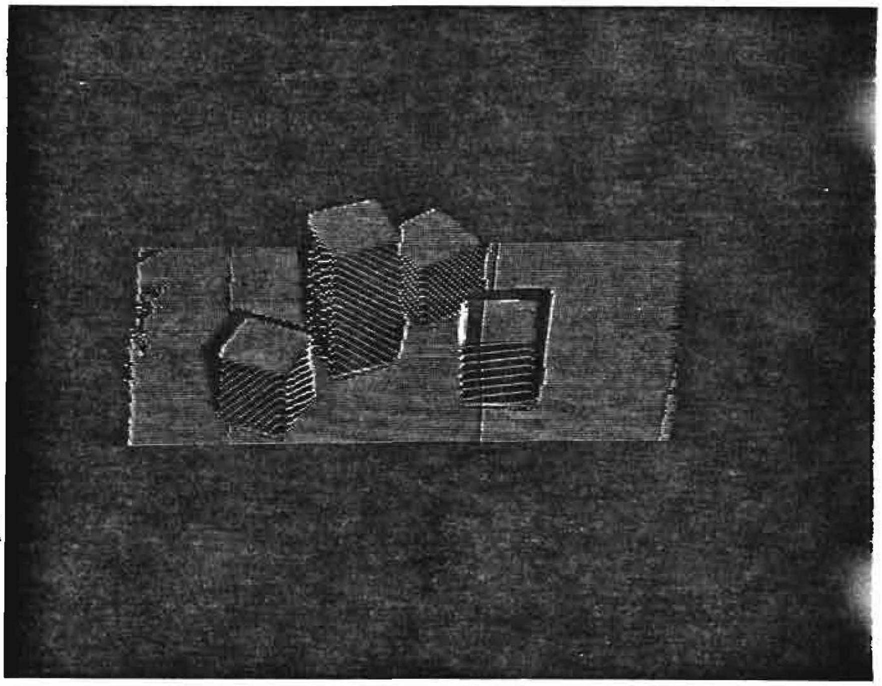

Figure 6. Scene from Range Database: scene_1

4.1. Normal Calculation

An interesting attribute of actual 3-D data is that it contains the intrinsic characteristics of surfaces which it represents. This allows us to recover shape more readily than with 2-D intensity data. Surface characteristics, such as edges, which are lost in intensity images due to the illumination process are always present wherever the surface is sampled with 3-D sensing techniques. Furthermore, some shapes which are ambiguous in intensity images are clearly discernible in 3-D data. An example is a deep bowl and shallow plate. When illuminated without shadows, these both appear similar in intensity images yet with range data, one is clearly different than the other. One would like to recover the intrinsic features which make up the surface.

Since each pixel contains the world coordinate in terms of the 3 Cartesian components, there are several low-level techniques which are useful for recovering surface information. These include surface normals and pointwise curvature.

There are several methods for computing surface normals. One would like to use a simple method such as the vector cross-product of neighboring points:

$$\vec{N} = \vec{U} \times \vec{V}$$

Where $\vec{N}$ is orthogonal to both $\vec{U}$ and $\vec{V}$.

Although computationally efficient, the reliability of this method is hampered by the noise present in the

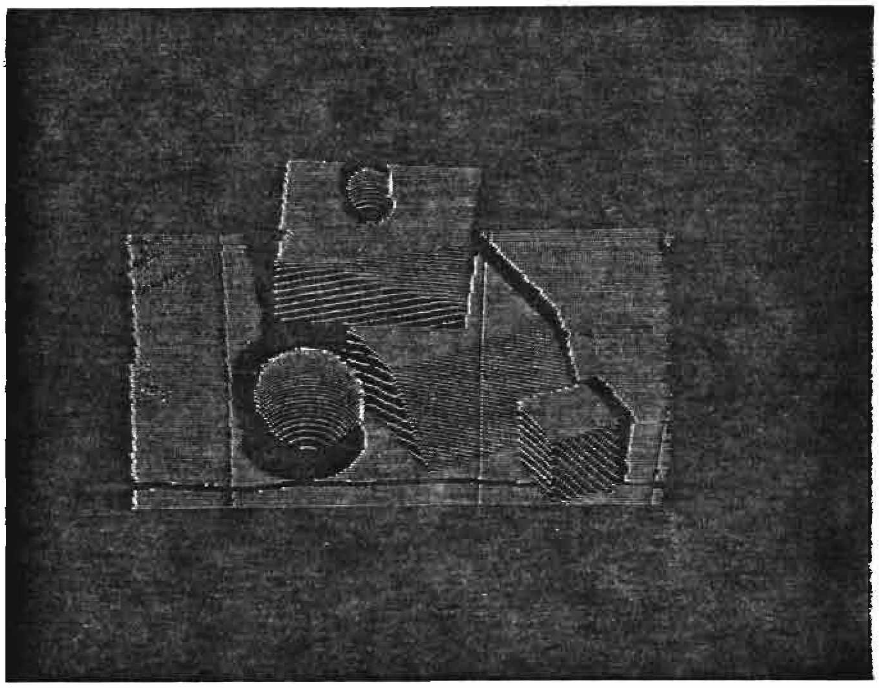

Figure 7. Scene from Range Database: scene_2

range data. One can simply smooth the data by an averaging technique, such as Guassian filtering, to remove some of this noise [7]. However, there are problems with smoothing the range data to overcome the noise. The major drawback is the loss of potentially interesting inflections on the surface of objects represented by the range data. Inflections, such as edges, change in curvatures, or shadows, are smeared by the averaging process as are the actual world space coordinates of the data points. Thus, sharp edges are reduced to smoother curves as a result of averaging operators. Median filter operators tend to sharpen an image, but there is a problem with attempting to find a median in 3 dimensions.

A second method incorporates heuristics about the signal-to-noise ratio of the particular sensor and computes the normals as a vector cross product of non-neighboring points. One can do this by selecting points with the following heuristic:

$$\vec{N} = \vec{U} \times \vec{V} \quad \textit{where} \quad \|\vec{U}\| > t \quad \text{and} \quad \|\vec{V}\| > t$$

where ||.|| is vector length and t is some threshold based on the amount of noise compared to the inner pixel distance.

Although this overcomes the signal-to-noise ratio, several other problems are incurred. One cannot be sure that the points $\vec{U}$ and $\vec{V}$ are part of the same surface if the vector length is too large. This tends to cause the normals to not reflect the actual tangent plane at the sampled surface point. Another problem is that pixels which are orthogonal in the image plane may not be orthogonal on the surface of the object. This introduces error due to the relative closeness of the two non-orthogonal points.

The method which we found to work the best is one which approximates a least-squares fit of a plane

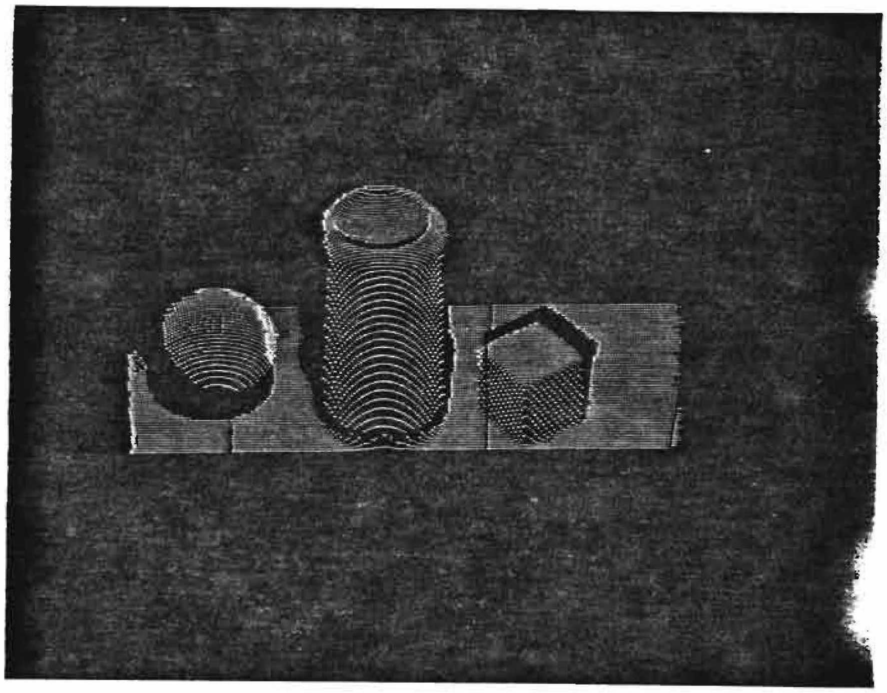

Figure 8. Scene from Range Database: scene_3

Scene	% Bad Data (with background as bad data)	%Bad Data (with background as good data)
bottle	0.423896	0.112651
bottle_1	0.253951	0.135679
poly_1	0.299228	0.254938
scene_2	0.346903	0.291482

Table 1. Percentage of Bad Data Points in Range Database Scenes

to a small window of the data. The approximation is achieved using the following well known technique [8]. We wish to fit a plane to a portion of our data, the small window, such that the error term, E, is minimized:

$$E = \sum_{i=1}^{P} (\vec{N} \cdot x_i + d)^2$$

where $\vec{N}$ is the unit normal to the plane and d is the distance of this plane from the origin. By minimizing this, we achieve the best fitting plane for the given set of points, P.

But d is just:

$$d = -\frac{\sum_{i=1}^{P} \vec{N}\cdot x_i}{P}$$

by substituting:

$$E = \sum_{i=1}^{P} (\vec{N}\cdot x_i - \sum_{i=1}^{P} \frac{\vec{N}\cdot x_i}{P})^2$$

The direction, $\vec{N}$, of the best fitting plane corresponds to the smallest eigenvalue of the matrix:

$$M = \sum_{i=1}^{P} (A_i \times A_i^t)$$

where:

$$A_i = x_i - \frac{\sum_{i=1} x_i}{P}$$

Although we have obtained good results with this method, it is computationally expensive. The speed would be improved with the appropriate hardware. We chose this method over the first two methods because of the noise in the data. Using a window as defined by P above, the data is locally smoothed thereby reducing the error. When implemented in terms of graph traversal, this algorithm becomes much more efficient in terms of execution time. Figure 9 shows the spatial proximity graph for poly_1 and Figure 10 shows the normals computed on that graph. The execution time on poly_1 was over three times shorter using the graph structure (11.3 vs. 37.5 cpu seconds).

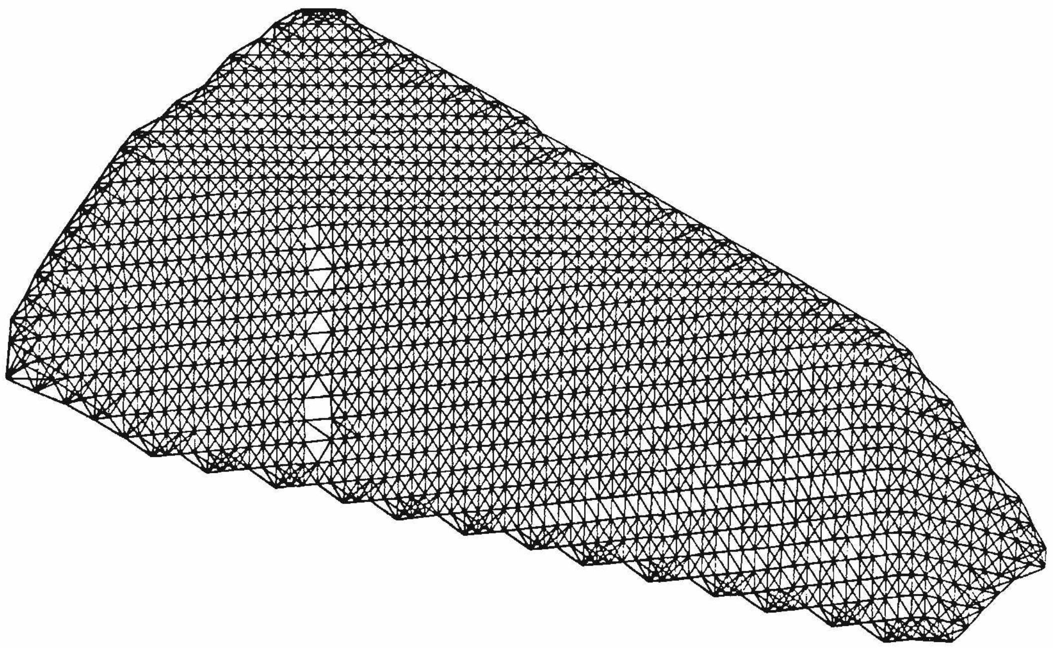

Figure 9. The Spatial Proximity Graph for poly_1

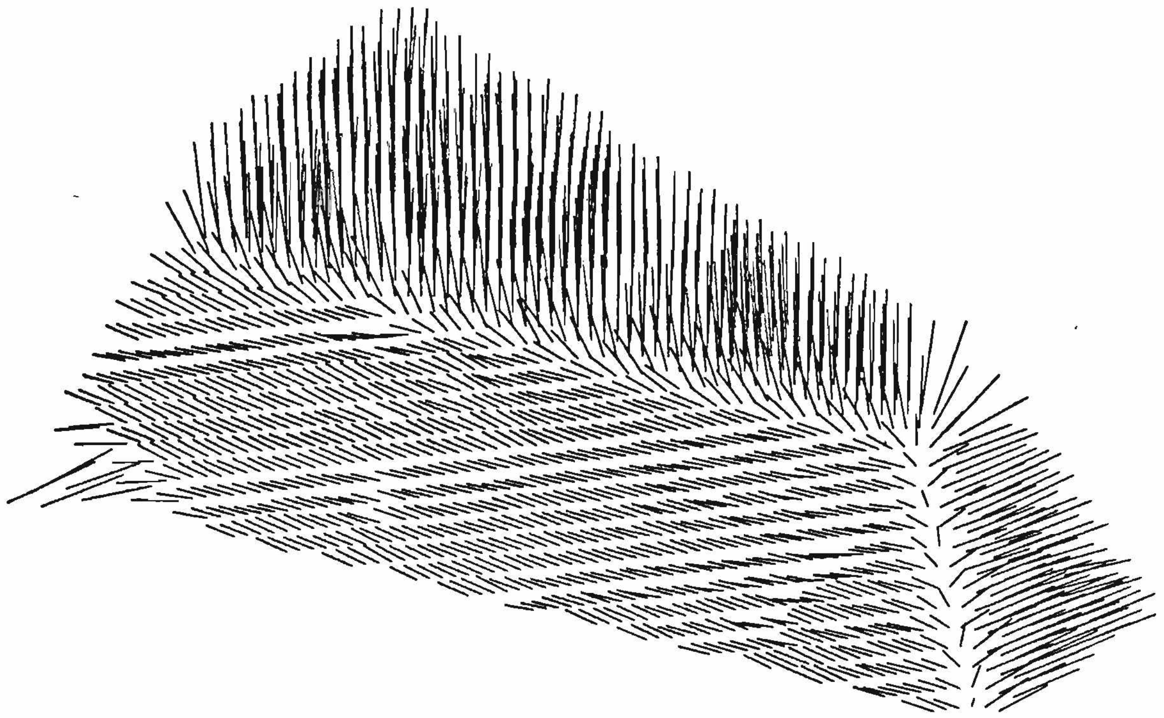

Figure 10. The Normals Calculated over the SPG

4.2. Curvature

Recently, computer vision researchers have attempted to look at surface curvature as a means for recovering the intrinsic shape of objects. The resurgence of this is due to the advent of dense range data. Curvature possesses the intrinsic characteristic of being invariant to translation and rotation thereby making it a prime candidate for describing surfaces. Curvature is defined as the acceleration of a moving trihedral, normal to the surface, following a space curve on the surface [24]. Since it is the acceleration, we can define curvature as a function of first and second derivatives. There are several types of curvatures defined in classical differential geometry textbooks which computer vision researchers have investigated. These include:

- normal curvature
- principal curvatures
- Gaussian curvature
- mean curvature

Medioni and Nevatia described a one dimensional operator to compute curvature as [23]:

$$k = \frac{f''_{\psi}}{(1+(f'_{\psi})^2)^{\frac{3}{2}}}$$

This defines curvature, k, in the ψ direction. The directions in which this metric reaches a minimum and a maximum are called the principal curvatures, k_{min} and k_{max}. These can be shown to be orthogonal. Obviously, in the discrete case, the directions must be approximated and error is introduced by the aliasing in the chosen directions. Medioni and Nevatia used the one dimensional difference operator in 4 directions to approximate the min and max local curvature.

Besl and Jain have described a method for pixel-wise computation of Gaussian and mean curvature [5]. They define the Gaussian curvature, K, and the mean curvature, M, at a point in terms of the 1st and 2nd fundamental forms:

$$K = \frac{f_{uu}f_{vv}-f_{uv}^2}{(1+f_u^2+f_v^2)^2}$$

$$M = \frac{f_{uu}+f_{vv}+f_{uu}f_v^2+f_{vv}f_u^2-2f_uf_vf_{uv}}{2(1+f_u^2+f_v^2)}$$

The Gaussian and mean curvature measures can also be defined in terms of the k_{min} and k_{max} curvatures as follows:

$$K = k_{min}{}^{*}k_{max}$$

$$M = \frac{k_{min}+k_{max}}{2}$$

Besl and Jain approximate the Monge Patch at each surface point with a 5x5 or 7x7 window. By approximating the surface with this patch, their method tends to be less prone to noise-error than Medioni and Nevatia's method.

We have investigated both methods of computing curvature in the context of our range sensing system. We have found that the using the computational technique for the principle curvatures, k_{min} and k_{max}, gives us the best results.

4.3. Segmentation

In the previous sections, we have described how to compute pointwise surface characteristics: normals and curvature. In this section, we will describe how to use these to segment scenes into symbolic representations which can be used for recognition and localization.

For most 3-D recognition schemes, features used for classification and matching can be grouped into two classes: region-based and edge-based. Since our goal is to realize an algorithmic/representation independent environment, we will formulate methods for achieving structure in both classes. We will not develop methods for global shape techniques since these are not robust to occlusion and not as strong in the 3-D paradigms. Similarly, we will exclude syntactic representations from our study although they could be derived from the data given proper algorithms.

The most obvious region-based representation is planar faces. There exist many techniques for computing planar faces given a set of 3-D data [14]. Since we have previously computed pointwise surface normals, the technique we use is a simple region-growing algorithm where the region grown is a planar region.

```
for every point,
  if it doesn't belong to a plane,
    start a new plane with this point; mark the point as part of the plane
```

```
Check the 8-neighbors and if they haven't been marked,
  then push them on the stack.
Until the stack is empty,
  pop the top of the stack
  if the normal is within a tolerance for the plane,
    mark the point as part of the plane
    Check the 8-neighbors and if they haven't been marked,
      then push them on the stack.
```

The advantage to this algorithm, is that we only have to check the distance squared of the ends of the unit normals and don't have to call the square-root function for every point. With the 3-point seed method, or others like it, we would have to measure the distance of the point from the plane which is tantamount to recomputing the unit surface normal for every point. We can keep track of the center of mass and average the normals on the fly. After the planes have been segmented, we can compute the area based on a triangulation method. If we keep track of the border pixels for each plane, we can quickly determine adjacencies and dihedral edges. Figure 11 shows the results of this algorithm run on the object shown in Figure 9.

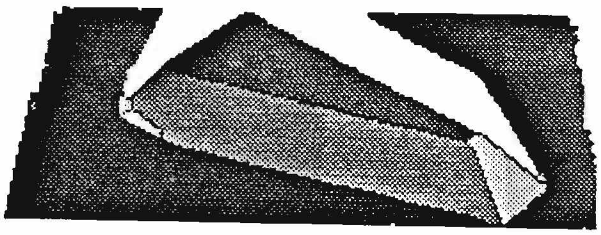

Figure 11. Planar Faces Detected in poly_1

This differs from the method of Hoffman and Jain in that they allow points to be added into a region through a clustering technique [20]. Clusters are then merged based on a generous threshold, 20 degrees, thus missing edges if the shared dihedral is less than 20 degrees.

Similarly, curvature points can be combined to form regions of constant curvature. Since we have captured the intrinsic pointwise characteristic of the surface, we can simply perform region growing using the curvature values. If we use the concise classifications described by Besl and Jain, we can easily grow regions of constant curvature.

Edges are another important feature which can be recovered from range data. Many 3-D object recognition systems are based on detecting edges of the object in a scene and then determining whether the detected edges map consistently onto the edges of the model. To find the edges in the range data, we simply traverse the spatial proximity graph, and at each point, if the error of fit of a planar surface is above a certain threshold, then we mark the point as an edge point. Figure 12 gives an example of the application of the algorithm.

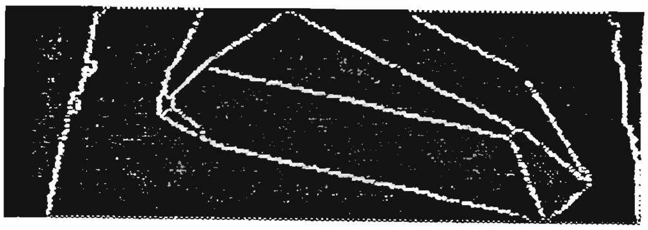

Figure 12. Edges Detected in poly_1

5. Discussion and Future Research

It is not possible to directly compare the processing costs of an image storage structure and the graph storage structures. As was mentioned above, the image structure cannot even be used for certain problems; for example, extracting features from full view CAD models [6]. Moreover, the neighborhoods in the graph structures contain exactly the points of interest for the feature calculation, whereas in the image structure the "neighbors" may not actually be physically near the point in question. Finally, there are ways to use the k-d tree structure in a sequence of operations to determine the features of interest. For example, if the tree is first built using (x,y,z) as the key, followed by selecting surface normal as the key, then the result is the planar faces. This type of operation could be easily represented in a relational query language fashion. That is, the operation just described could be expressed as:

$\sigma_F(\text{Build}(\pi_{x,y,z}(R)))$

where F stands for a formula expressing constant surface normal

More complicated relational expressions are useful for extracting other features. This approach is one avenue for future research.

Of crucial importance to building up-to-date spatial proximity graphs to organize a continuous flow of a massive amount of sensor data is the ability to dynamically insert and delete data on a k-d tree or any equivalent database storage structure that allows efficient query and searching to be performed on the data. Overmars and van Leeuwen [25] have presented some initial work on dynamic multi-dimensional data structures, and the usefulness of their results to our application must be investigated.

We have described representations and processing operations which are useful for the analysis of three-dimensional range data. Such techniques are instrumental to the successful exploitation of such data.

References

[1] Ahuja, N.
Dot Pattern Processing Using Voronoi Neighborhoods.
IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-4(3):336-343, May, 1982.

[2] Ballard, D.H. and C.M. Brown.
Computer Vision.
Prentice Hall, New York, 1982.

[3] Barrow, Harry and Jay Tennenbaum.
Recovering Intrinsic Scene Characteristics from Images.
Technical Report 157, SRI International, April, 1978.

[4] Bentley, J.L.
Multidimensional Binary Search Trees Used for Associative Searching.
CACM 18(9):509-517, September, 1975.

[5] Paul J. Besl and Ramesh C. Jain.
Invariant Surface Characteristics for 3-D Object Recognition in Range Images.
CVGIP (33):33-80, 1986.

[6] Bhanu, Bir, S.K. Lee, C.C. Ho and Thomas C. Henderson.
Range Data Processing: Representation of Surfaces by Edges.
In *Proceedings of the International Conference on Pattern Recognition*, pages 236-238. Paris, France, October, 1986.

[7] Michael Brady, Jean Ponce, Alan Yuille.
Describing Surfaces.
In *Robotics Research, the 2nd International Symposium*, pages 5-16. 1983.

[8] Faugeras, O.D. and M. Hebert.
The Representation, Recognition and Locating of 3-D Objects.
Robotics Research 5(3):27-52, 1986.

[9] Friedman, J.H., J.L. Bentley and R.A. Finkel.
An Algorithm for Finding Best Matches in Logarithmic Expected Time.
ACM Trans. on Math. Soft. 3(3):209-226, September, 1977.

[10] Hansen, C. and Thomas C. Henderson.
The UTAH Range Database.
Computer Science UUCS-86-113, University of Utah, April, 1986.

[11] Henderson, T.C.
An Efficient Segmentation Method for Range Data.
In *SPIE Conference on Robot Vision*, pages 46-47. Arlington, VA, May, 1982.

[12] Henderson, T. and E. Triendl.
The k-d Tree Representation of Edge Descriptions.
In *Proceedings of International Conference on Pattern Recognition.* October, 1982.

[13] Henderson, Thomas C. and Wu So Fai.
A Multi-sensor Integration and Data Acquisition System.
In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 274-280. IEEE, June, 1983.

[14] Henderson, T.C.
Efficient 3-D Object Representations for Industrial Vision Systems.
IEEE Transactions on Pattern Analysis and Machine Vision PAMI-5(6):609-618, November, 1983.

[15] Henderson, T.C. and E. Shilcrat.
Logical Sensor Systems.
Journal of Robotic Systems 1(2):169-193, 1984.

[16] Henderson, T.C. and Wu So Fai.
MKS: A Multi-sensor Kernel System.
IEEE Transactions on Systems, Man, and Cyberbetics SMC-14(5):784-791, September/October, 1984.

[17] Henderson, T.C., C.D. Hansen, and Bir Bhanu.
The Specification of Distributed Sensing and Control.
Journal of Robotic Systems 2(4):387-396, 1985.

[18] Henderson, T.C., Chuck Hansen and Wu So Fai.
Organizing Spatial Data for Robotic Systems.
Computers in Industry 6(5):331-344, October, 1985.

[19] Henderson, T.C. and E. Triendl.
Storing Feature Descriptions as 2-D Trees.
IEEE Transactions on Geoscience and Remote Sensing :301-303, 1986.

[20] Richard Hoffman and Anil Jain.
Segmentation and Classification of Range Images.
In *Computer Vision and Pattern Recognition, Miami Florida*, pages 446-451. 1986.

[21] Jarvis, R.A.
A Perspective on Range Finding Techniques for Computer Vision.
IEEE Transactions on Pattern Analysis and Machine Intelligence 5(2):122-139, 1983.

[22] Marr, D.
Early Processing of Visual Information.
AI Memo 450, MIT, Cambridge Mass, December, 1975.

[23] G. Medioni, R. Nevatia.
Description of 3-D Surfaces Using Curvature Properties.
In *Proc. DARPA Image Understanding Workshop, New Orleans, LA*, pages 291-299. 1984.

[24] Barrett O'Neil.
Elementary Differential Geometry.
Academic Press, 1966.

[25] Overmars, M.H. and Jan van Leeuwen.
Dynamic Multi-Dimensional Data Structures Based on Quad- and K-D Trees.
Acta Informatica 17:267-285, 1982.

[26] Pavilidis, T.
Structural Pattern Recognition.
Springer-Verlag, 1977.

[27] Denis Poussart.
Three-Dimensional Sensing for Computer Vision.
In *SPIE Tutorial, SPIE Quebec International Symposium on Optical and Optoelectronic Applied Sciences and Engineering, Quebec, Canada.* 1986.

[28] Samet, H.
Region Representation: Quadtrees from Boundary Codes.
CACM 23(3):163-170, March, 1980.

[29] *100-A Scanner User's Manual*
1982.

[30] Toussaint, G.T.
The Relative Neighborhood Graph of a Finite Planar Set.
Pattern Recognition 12:261-268, August, 1980.

[31] Zahn, C.T.
Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters.
IEEE Transactions on Computers C-20(1):68-86, 1971.