

V

Mathematical Measures in the Analysis of Three-Dimensional Binary Images

Clifford D. Krumvieda

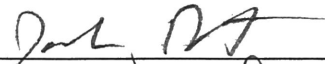
University Undergraduate Fellow, 1988-1989

Texas A&M University

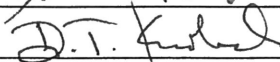
Department of Mathematics

APPROVED:

Fellows Advisor:



Honors Director:



Date:

CONTENTS

| | |
|--|------------|
| Abstract | iii |
| I. Introduction | 1 |
| II. Theoretical Background | 2 |
| A. Shapes | 2 |
| B. Measures | 2 |
| III Data | 5 |
| IV Binary Object Recognition | 8 |
| A. Discrete Boundaries in Space | 8 |
| B. Method | 9 |
| C. Details | 10 |
| D. Examples | 12 |
| Surface Area | 12 |
| Shape Measures | 13 |
| V. Three-Dimensional Binary Object Display | 18 |
| A. Preliminaries | 18 |
| B. Method | 29 |
| C. Examples and Technical Details | 23 |
| D. Plates | 24 |
| Summary | 27 |
| Acknowledgements | 28 |
| References | 29 |
| A Rationalization of a Homogeneous Congruence Measure | 31 |
| B Best Fitting Tangent Plane | 33 |
| C Programs | 35 |

ABSTRACT

Three-dimensional binary representations of continuous objects are analyzed for shape properties. Theoretical results are described that suggest a method for developing three-dimensional shape measures. New shape measures are introduced, and a method for approximating them over the boundary of a low resolution binary object is presented. Statistical results are shown that yield high differentiation between binary shapes. A method for displaying the shaded planar image of a three-dimensional binary image is described.

I. INTRODUCTION

All around us, shapes are repeated at a relatively rapid rate. Rectangular solids, for instance, can be found disguised as books, sponges and buildings; spheres can be seen in door knobs, gems, and bubbles. What do we mean when we say that two objects “have the same shape”?

From a theoretical view, we would like to define rigorously the intuitive concept of shape. From a practical view, we would like to teach a computer to recognize the shape of an arbitrary object held in its memory. There has been much work done on the two-dimensional problem [1,2,3,4,5], but much less work done in three-dimensions [6,7]. Our work is hard to compare with earlier work because the shapes we are dealing with have their only existence as three-dimensional binary objects. Reynolds *et al.* [8] use such data, obtained from computed tomography (CT) or nuclear magnetic resonance (MRI) measurements, to produce shaded displays on a flat screen. Levoy [9] uses 8-bit 3-D discrete CT data to make displays (using ray tracing and a model for reflectance-transmission) very similar to ours, but again there is no mention of *recognition* of shape from the discrete 3-D “image.” Many authors (literally in the thousands) have dealt with various aspects of shape perception and modelling starting with parametric representations, contours in 2-D or on the surface itself, from surface approximation using Bézier-Bernstein splines, from edges or webs, from range data, stereo views, shading, polygons, etc. Azriel Rosenfeld’s bibliographic review and classification “Image Analysis and Computer Vision” which appears yearly in *Computer Vision, Graphics, and Image Processing* would be an excellent place to start searching for references in other directions. The contribution covering 1988 contains over 1,600 references.

One problem with attempting to analyze three-dimensional binary shapes is that they are hard to visualize intuitively. For instance, if the computer reported that two binary objects were the same, how could the result be verified? This question led us to develop a display procedure. The program uses elementary regression techniques to display a shaded representation of a three-dimensional binary object on a two-dimensional computer screen. This development was based on recent research by psychologists [10,11,12,13] that has revealed that shading and occlusion of objects in two-dimensional pictures produce a compelling perception of three-dimensional shape.

II. THEORETICAL BACKGROUND

A. Shapes

We will define the intuitive concept of “shape” by saying that two objects *have the same shape* if one of them can be obtained from the other by a finite number of translations, rotations, and magnifications. Therefore, our concern is with the identification of objects in space independently of their spatial position, orientation, and size. We wish to do this without decomposing the objects into more primitive, understood objects. In fact, no supposition that these random objects are composed of parts is made; we are therefore making no attempt to understand the objects, only to classify them as one of a number of previously identified objects, or to indicate that no such classification is probable. To this end, we seek to define a “distance” between two objects: one which will be small when the objects seem like one another and large when they are clearly different. The distance should be independent of the orientation, position, and size of the objects being compared.

A clear analogy can be drawn between this situation and the Euclidean distance between points in space. If we can associate each object with a list of real numbers, and regard this list as the coordinates of a point in space, then classical statistical techniques can be used to deal with the objects as points (in a probably high dimensional space). Thus, the difficulty transfers to that of discovering enough different measures to distinguish between a set of objects. If we are to approach our goal, then the associations of objects to numbers should have the same invariance properties we are seeking in object recognition.

B. Measures

Definition. An *object measure* is a mapping from the set of objects to the set of real numbers.

Diameter $\delta(O)$, surface area $\sigma(O)$, and volume $\rho(O)$ are examples of natural object measures. Yet these measures are clearly fundamentally different. To illustrate this, imagine that an object—say a cube—under study has random changes in size (magnifications) which we wish to model. Then the variations in the three measurements will be different, and this cannot be corrected by scaling. For example, if each measurement is normalized by dividing by the sample mean, then the three variations will be in the

ratio 1:2:3 although the expected measurements are all 1. One way to minimize this tendency is to insist the measurements be dimensionally the same.

Definition. An object measure μ is said to be *homogeneous* (or *dimensionally linear*) if $\mu(\alpha O) = \alpha\mu(O)$ for each positive magnification α . If O is an object and α is a positive real number, let

$$\alpha O = \{\alpha x : x \in O\}$$

be the magnification of O by α .

For example, $\delta(O)$, $\sigma(O)^{1/2}$, and $\rho(O)^{1/3}$ are natural homogeneous object measures. Each is also a *congruence measure*.

Definition. A measure is said to be a *congruence measure* if it is rotation, translation, and reflection-invariant.

Definition. A *shape measure* is a magnification invariant congruence measure.

Shape measures are dimensionless. In particular, the ratio of two homogeneous congruence measures is a shape measure. Because homogeneous congruence measures are relatively easy for a computer to calculate, these ratios will be the only shape measures of interest to us. A natural denominator measure is one of the three just mentioned above, for they are never zero for a real object and are easily estimated. In discrete applications, the normalized volume estimate is the most stable because of its low ($\frac{1}{3}$) exponent.

Let S be the surface of an object with area $|S|$, centroid r_0 , normal to the surface \mathbf{n} , and element of surface area $d\sigma$. The measures

$$\left(\frac{1}{|S|} \int_S \|r - r_0\|^p |(r - r_0) \cdot \mathbf{n}|^q d\sigma \right)^{1/(p+q)}$$

are homogeneous congruence measures if $p + q > 0$ (see Appendix A); we have tested them on discrete shapes and found the methods by which the continuous integrals were approximated to be very reliable. However, some confusion remained between objects we felt were sufficiently different. One way to generate more measures (calculable by the same general method) is to relax the rigid demand for strict homogeneity.

Definition. A congruence measure is said to be *near homogeneous* if it is homogeneous when restricted to spheres. A congruence measure which is constant on spheres is called a *near shape measure*.

One way to get near shape measures is to take the ratio of a near homogeneous congruence measure to one of the three natural homogeneous congruence measures

above.

The measures

$$\log \left(\frac{1}{|S|} \int_S \exp(\|r - r_0\|) d\sigma \right)$$

and

$$\exp \left(\frac{1}{|S|} \int_S \log^+(\|r - r_0\|) d\sigma \right)$$

are very different near congruence measures. They have the form

$$\phi^{-1} \left(\frac{1}{|S|} \int_S \phi(\|r - r_0\|) d\sigma \right)$$

where ϕ is one to one and increasing on \mathbb{R}^+ .

It is easy to see how to fit $\|r - r_0\|$ and $|(r - r_0) \cdot \mathbf{n}|$ into similar expressions. In analogy with the homogeneous case,

$$\left[\log \left(\frac{1}{|S|} \int_S \exp(\|r - r_0\|^p |(r - r_0) \cdot \mathbf{n}|^q) d\sigma \right) \right]^{\frac{1}{p+q}},$$

$$\frac{1}{p+q} \exp \left(\frac{1}{|S|} \int_S \log^+(p\|r - r_0\| + q|(r - r_0) \cdot \mathbf{n}|) d\sigma \right),$$

and

$$\left[\log \left(-\frac{1}{|S|} \int_S \exp(-\|r - r_0\|^p |(r - r_0) \cdot \mathbf{n}|^q) d\sigma \right) \right]^{\frac{1}{p+q}}$$

are at least candidates for trial. These formulae exploit the functional equations of the logarithm and exponential functions in addition to the inverse relationship.

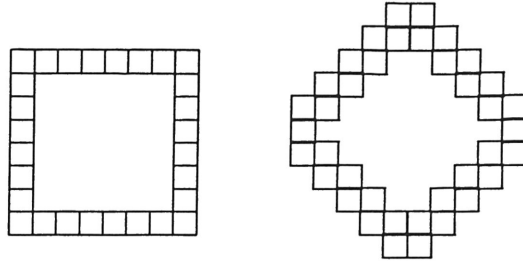


Fig. 1. Two Rotations of a Two-Dimensional Binary Square

III. DATA

Three-dimensional binary data are not too plentiful. Unlike two-dimensional data, which can be obtained with only a simple robotic vision system [5], three-dimensional physical data must be obtained through sophisticated techniques such as x-ray crystallography and nuclear magnetic resonance. To study discrete shape measures, we needed a great deal of permutable data; we decided to generate it by computer.

We wanted the data to be, if possible, easy to rotate. Raw binary data, in general, is very hard to rotate accurately because the number of “on” binary cells depends on the orientation of the object relative to the cells [5] (see Fig. 1). To eliminate the problem of rotating binary data, we decided to use a less primitive data form, one that can be easily rotated, and develop a method of generating binary data from this high-level form.

Perhaps the easiest shape to rotate is the sphere. For instance, a sphere centered at $(1, 0, 0)$ can be rotated counterclockwise about the z -axis by deleting it and constructing a new one with the same radius centered at $(0, 1, 0)$. Compare this to the problem of rotating a square, which is not symmetric about its center. To take advantage of the sphere’s simple rotational property, we chose to represent almost all of our test data as sets of intersecting spheres. That is, in a high-level form, most of our data was a finite set of center coordinates and sphere radii. This is evident in the objects of plates 1 and 2, which are pictures of a few data sets. Notice how shapes in the first three rows and in the first two columns of the fourth row are constructed of atom-like spheres (the shapes in the third and fourth columns of the fourth row are constructed of *ellipsoids*, which are also very easy to rotate). The figures exhibit the rotational properties of the data: Each object in Plate 2 is a rotation (and/or magnification) of the corresponding object in Plate 1.

Generating binary data from the sphere files was a straightforward but slow process.

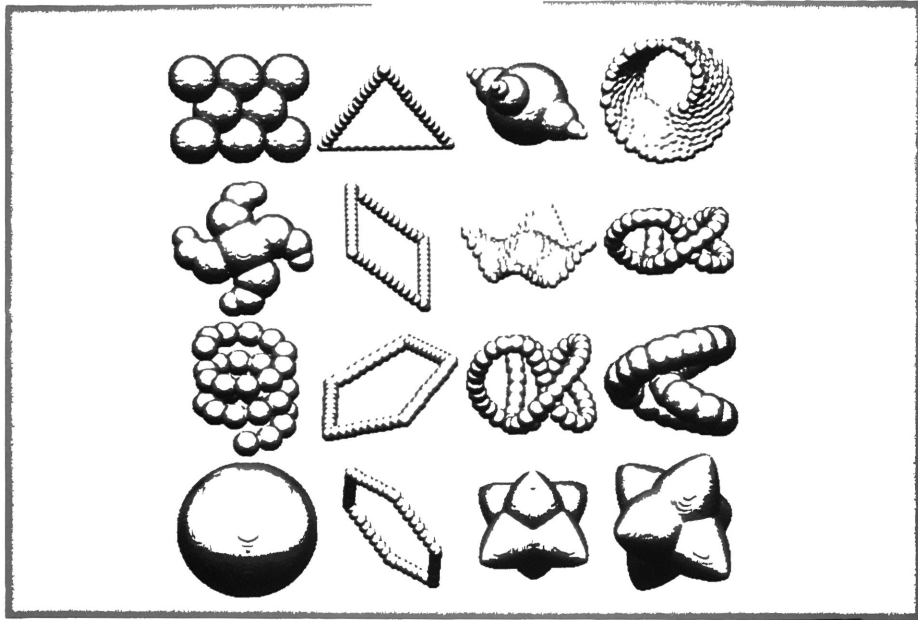


Plate 1

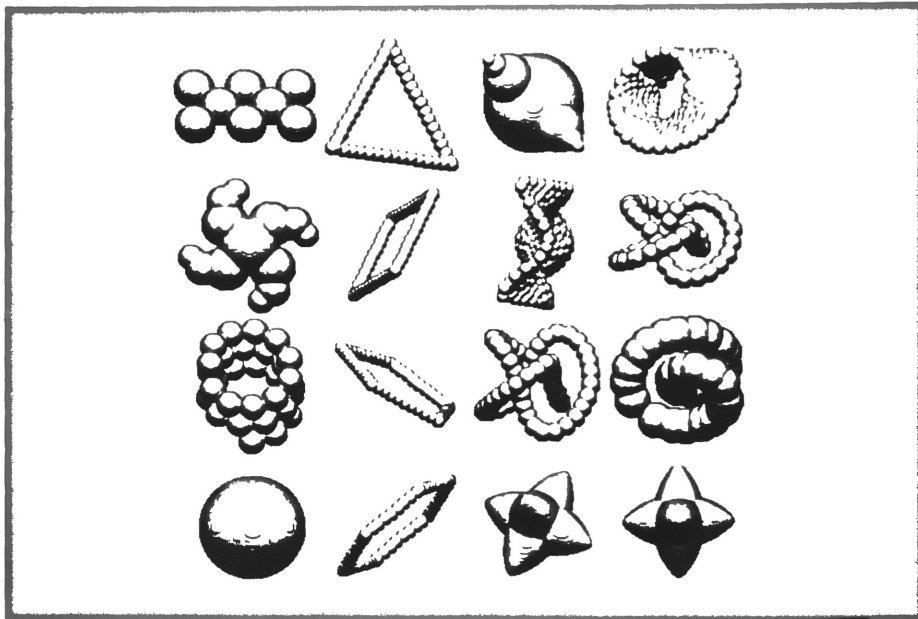


Plate 2

Each object was imbedded in a $128 \times 128 \times 128$ binary array, and so there were 2^{21} binary cells that had to be set for each object. The larger data sets—i.e., the ones constructed of many spheres—took more than two hours of VAX 8800 CPU time to generate. Fortunately, this project was a part of the 1988-1989 Cornell National Supercomputer Facility Research Experience for Undergraduates program, and we had access to the two IBM 3090-600E's in Ithaca, New York. Generating one particular shape that took two hours and 28 minutes of VAX 8800 CPU time took only four minutes and fifteen seconds of IBM 3090-600E CPU time. Another shape, which took a little less than twelve minutes to generate on the supercomputer would not have been created on the VAX.

We used several methods to generate the high-level sphere data files. One means involved laying spheres along a three-dimensional parametric curve. The program we wrote to do this, DISPF.FOR, is included in Appendix C. The program that rotated the sphere files, ROTCALL5.FOR, and the program that generated the binary data from these files, DISPIT2.FOR, are also in this Appendix.

Note that neither the recognition nor the display procedure took advantage of the fact that the binary data was generated from spheres, but worked directly with the binary data. This makes them very flexible and applicable to data that was not computer generated.

IV. BINARY OBJECT RECOGNITION

In the discrete case, part of the problem in calculating shape measures is the need to calculate integrals of the form

$$\int_S F(\mathbf{n}, \mathbf{r}) d\sigma,$$

where F is a function of \mathbf{r} and \mathbf{n} on the surface S , \mathbf{r} is a boundary element, \mathbf{n} is the outward unit normal, and $d\sigma$ is the element of surface area. The estimation of such an integral is difficult because the object is not continuous even though the underlying shape being represented may be C^∞ . Two related questions must be answered before the integral can be estimated numerically:

- How can one estimate the unit normal or, equivalently, the tangent plane?
- Even a continuous closed surface is often difficult to parameterize so as to compute $d\sigma$; what can one do to get the discrete element of surface area?

In the plane, the perimeter of the shape plays a crucial rôle in the shape recognition problem. Improvements on the methods given by Bryant and Bryant [5] have led to real-time recognition of shapes in 128×512 binary images using quickly computed functionals of the chain-coded boundary. The methods are fast because the boundary is linearly ordered and very small compared to the object; furthermore, the boundary can (and should) be sampled to produce both an accurate estimate of the length and partial magnification invariance. None of these advantages is present in space. The boundary is large, not linearly ordered in a natural way, and is difficult (or impossible) to sample consistently.

A. *Discrete Boundaries in Space*

The *boundary* of a discrete 3-D object is the set of points in the object with one of their six nearest neighbors not in the object. The shape is imbedded in a three dimensional array; down one of the axes the binary pattern is assumed packed in computer words. The boundary is found using logical operations on the bit pattern representing the shape (rather than looking at neighbors) [14]; it is present as a 3-D array and as a list of coordinates. For each point in the boundary list, one needs an estimate of the unit normal and of the differential of surface area. The general idea is to find an approximate tangent plane at each boundary point, to estimate the element of surface

area by looking at the direction cosines of the unit normal, and to establish a consistent outward direction (if required).

Before we begin, let us review briefly how surface integrals are computed by hand. One is given a surface, say $z = f(x, y)$, with parameters x and y lying in a plane. The integral of a scalar function g over the surface might be computed as the plane integral of

$$g(x, y, f(x, y)) d\sigma = g(x, y, f(x, y)) \frac{dx dy}{|\cos \gamma|},$$

with γ being the angle from the unit normal to the z -axis. That is, $d\sigma = dx dy / |\cos \gamma|$.

It is intuitively clear that a discrete realization of this technique will be best when $|\cos \gamma|$ is large—close to 1. That requires we allow the parameterization *to depend on the surface point*. We select for the parameterization direction the one with largest direction cosine. Levoy [9] uses a similar technique, but instead estimates the gradient of the density not available in this binary setting. Under the assumption that a tangent plane approximation of the form $ax + by + cz = d$ exists near the point (x_0, y_0, z_0) , our first idea was to solve the constrained least squares problem

$$\text{minimize } \sum [ax_i + by_i + cz_i - d]^2 \quad \text{subject to } a^2 + b^2 + c^2 = 1,$$

where the sum is extended over the “training” points collected. The normal equations of the resulting Lagrange multipliers problem are only mildly nonlinear, but we have not found a computationally efficient solution to the problem taking this approach (considering the hundreds of thousands of times the procedure must be invoked). The use of modified steepest descent methods is satisfactory except for the computer time taken; in fact, if one uses as a starting point the solution at a neighboring point, then standard iterative methods converge rapidly. However, another approach, described now, produces better approximations immediately. In the discrete case, and with a surface which is the boundary of an entire solid and thus not routinely parameterized by a single plane, our idea is to allow the parameters to be dependent on the point: we select as parameters the best of the three coordinate planes, which will vary from point to point.

B. Method

A preliminary step has found the boundary of the discrete shape. The array which contains the boundary points is now searched. We show how to approximate the tangent

plane to the surface and the discrete version μ_0 of $d\sigma$ at each point $\mathbf{r}_0 = (x_0, y_0, z_0)$ in the boundary. The idea is to solve three unconstrained linear least squares problems looking down each of the three coordinate directions and determine which direction gives the best fitting plane. One can predict which solution is best without carrying out all the calculations. An example of one of the problems (the x -axis look problem) is:

Let $(x_i, y_i, z_i), i = 1, \dots, k$ be the k neighbors of this point. We are looking for a plane of the form $(x - x_0) = c_2(y - y_0) + c_3(z - z_0)$, and we wish to

$$\text{minimize} \quad f(c_2, c_3) = \sum [(x_i - x_0) - c_2(y_i - y_0) - c_3(z_i - z_0)]^2$$

which represents the sum of the square differences from the set of neighbor points to any potential tangent plane. If we take the partial derivatives of f with respect to c_2 and c_3 , we obtain

$$\begin{aligned} \frac{\partial f}{\partial c_2} &= 2 \sum x_i z_i - 2c_3 \sum y_i z_i - 2c_2 \sum z_i^2 \\ \frac{\partial f}{\partial c_3} &= 2 \sum x_i y_i - 2c_2 \sum y_i^2 - 2c_3 \sum y_i z_i. \end{aligned}$$

We can set these to zero and apply Kramer's rule to obtain a quick solution.

The element of area is estimated by taking the parametric representation of the surface to be the coordinate plane which best fits the points collected as viewed normal to that plane. As will be seen, few computations are required.

C. Details

- *Step 1.* Search the 26 points in the 3×3 discrete cube centered at \mathbf{r}_0 for boundary points; let the coordinates be $\{(x_i, y_i, z_i) : i = 1, \dots, k\}$. (If $k < 6$ it is unlikely that a tangent plane exists, possibly because the sampling density is too low, but proceed anyway. Such points are bound to be rare if the shape is adequately sampled, and they are not missed in the application to surface integral approximation or display. After all, we are sampling the surface at the 27 points in the cube centered at the point (it counts too), and one would expect 9 points. In the tests presented later the case $k < 8$ was never observed.)

- *Step 2.* Form the following six sums (all over 1 to k).

$$\begin{aligned} a &= \sum(x_i - x_0)^2 & d &= \sum(x_i - x_0)(y_i - y_0) \\ b &= \sum(y_i - y_0)^2 & e &= \sum(x_i - x_0)(z_i - z_0) \\ c &= \sum(z_i - z_0)^2 & f &= \sum(y_i - y_0)(z_i - z_0) \end{aligned}$$

- *Step 3.* Evaluate the three 2×2 determinants

$$A = \begin{vmatrix} b & f \\ f & c \end{vmatrix} \quad B = \begin{vmatrix} a & e \\ e & c \end{vmatrix} \quad C = \begin{vmatrix} a & d \\ d & b \end{vmatrix}.$$

- *Step 4.* Select the largest of these three integers. By Schwartz's inequality each is non-negative. If all three are zero no tangent plane or normal line exists, and the differential of surface area μ is taken to be zero. (This event has never been observed.)
- 4.a A is largest. In this case the best fitting (least squares) tangent plane will be obtained by looking down the x -axis. (The proof of this assertion is found in the Appendix B.) Let

$$c_2 = \begin{vmatrix} d & f \\ e & c \end{vmatrix} / A \quad c_3 = \begin{vmatrix} b & d \\ f & e \end{vmatrix} / A.$$

Then the direction cosines (α, β, γ) of the unit normal are

$$\left(1/\sqrt{1 + c_2^2 + c_3^2}, -c_2/\sqrt{1 + c_2^2 + c_3^2}, -c_3/\sqrt{1 + c_2^2 + c_3^2} \right).$$

Let $\mu = 1/\alpha$.

- 4.b B is largest. Then the best fitting tangent plane will be obtained by looking down the y -axis. Let

$$c_1 = \begin{vmatrix} d & e \\ f & c \end{vmatrix} / B \quad c_3 = \begin{vmatrix} a & d \\ e & f \end{vmatrix} / B.$$

The direction cosines (α, β, γ) of the unit normal are

$$\left(-c_1/\sqrt{1 + c_1^2 + c_3^2}, 1/\sqrt{1 + c_1^2 + c_3^2}, -c_3/\sqrt{1 + c_1^2 + c_3^2} \right).$$

Let $\mu = 1/\beta$.

- 4.c C is largest. The best fitting tangent plane will be obtained by looking down the z -axis. Let

$$c_1 = \begin{vmatrix} e & d \\ f & b \end{vmatrix} / C \quad c_2 = \begin{vmatrix} a & e \\ d & f \end{vmatrix} / C.$$

The direction cosines (α, β, γ) of the unit normal are

$$\left(-c_1 / \sqrt{1 + c_1^2 + c_2^2}, -c_2 / \sqrt{1 + c_1^2 + c_2^2}, 1 / \sqrt{1 + c_1^2 + c_2^2} \right).$$

Let $\mu = 1/\gamma$.

D. Examples

Surface Area

One rather sharp test of the method is the estimate of surface area given by $\sum \mu_m$. If the discrete shape is derived from a C^∞ continuous shape, then the analytic surface area should be approximated by this sum, and the approximation should improve as the mesh becomes finer (or, equivalently, the relative error should decrease as the size of the sphere increases). We generated four spheres of radii 16 through 62 as described in Table 1. We estimate the surface area (labelled “Area” in Table 1) using the formula

$$S = 4\pi \left(\frac{3V}{4\pi} \right)^{2/3},$$

where V is the estimate of the volume obtained by counting the number of points in the shape and subtracting half the number of boundary points. This is an extremely stable estimate of the volume which is still quickly computed. In the table one finds the nominal radius, the surface area as estimated, the count of 1-boundary points, the estimate $\sum \mu_m$, and the number $4\pi r^2$ using the nominal radius. Percentage deviations from the calculated volume are indicated. The discrete spheres, imbedded in a $128 \times 128 \times 128$ array, are given by

$$\{(i, j, k) : (i - 64)^2 + (j - 64)^2 + (k - 64)^2 < r^2\}.$$

The count of internal 1-boundary points is, as expected, low. However, even if the number of points in the external 1-boundary were used, this estimate would remain low. The 2-boundary (not shown in the table) is a gross overestimate of the surface area.

TABLE I. ESTIMATES OF SURFACE AREA FOR FOUR SPHERES.

| Radius | Area | Estimate from: | | | Percent errors: | | |
|--------|-------|----------------|--------------|--------|-----------------|--------------|--------|
| | | count | $\sum \mu_m$ | radius | count | $\sum \mu_m$ | radius |
| 16 | 3044 | 2546 | 2998 | 3217 | -16.4 | -1.5 | 5.6 |
| 31 | 11726 | 9774 | 11642 | 12076 | -16.6 | -0.7 | 2.9 |
| 47 | 27246 | 22730 | 27051 | 27759 | -16.5 | -0.7 | 1.8 |
| 62 | 47641 | 39702 | 47320 | 48305 | -16.6 | -0.6 | 1.3 |

Shape Measures

In Appendix A we have shown that

$$\frac{1}{V^{1/3}} \left(\frac{1}{|S|} \int_S \|r - r_0\|^p |(r - r_0) \cdot \mathbf{n}|^q d\sigma \right)^{1/(p+q)}$$

is a continuous shape measure. In the discrete case, we merely approximate the various parts of the formula. The integrals are approximated by the sums

$$\frac{1}{V^{1/3}} \left(\frac{1}{|S|} \sum \| \mathbf{r}_m - \mathbf{r}_0 \|^q |(\mathbf{r}_m - \mathbf{r}_0) \cdot \mathbf{n}_m|^p \mu_m \right)^{\frac{1}{p+q}}$$

where all the estimates of the quantities involved are described above. We have tested the measures for the spheres of nominal radius 64: remarkable agreement with the theoretical value $(\frac{4}{3}\pi)^{1/3}$ is obtained. For values of p and q with $1 \leq p + q \leq 4$, $p > 0$, and $q > -4$, the agreement is within 1% (and often 0.1%).

Similarly, we approximated the continuous near shape measures introduced in Section II of this paper, and generated 36 different measures of our discrete data. In Fig. 2, there is a plot of measure 1,

$$\frac{S^{\frac{1}{2}}}{V^{\frac{1}{3}}},$$

vs. measure 21,

$$\frac{1}{|S|} \int_S \|r - r_0\|^4 (|r - r_0| \cdot \mathbf{n})^{-3} d\sigma.$$

Each character in the graph represents a different binary object; objects representing the same shape are denoted with the same letter. In Fig. 3, there is a similar plot of

Figure 2. Measure 1 vs. Measure 21
for 61 shapes

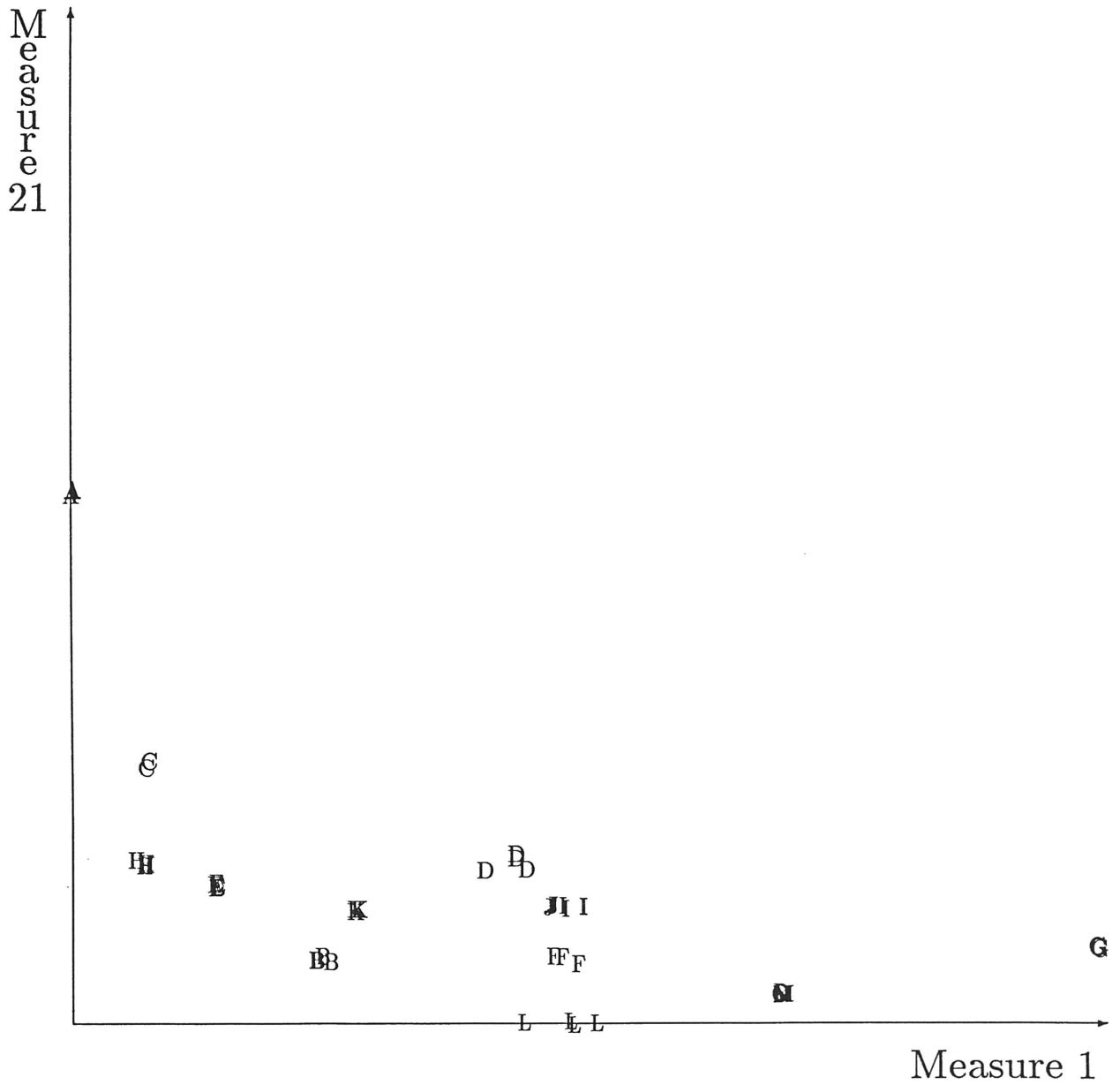
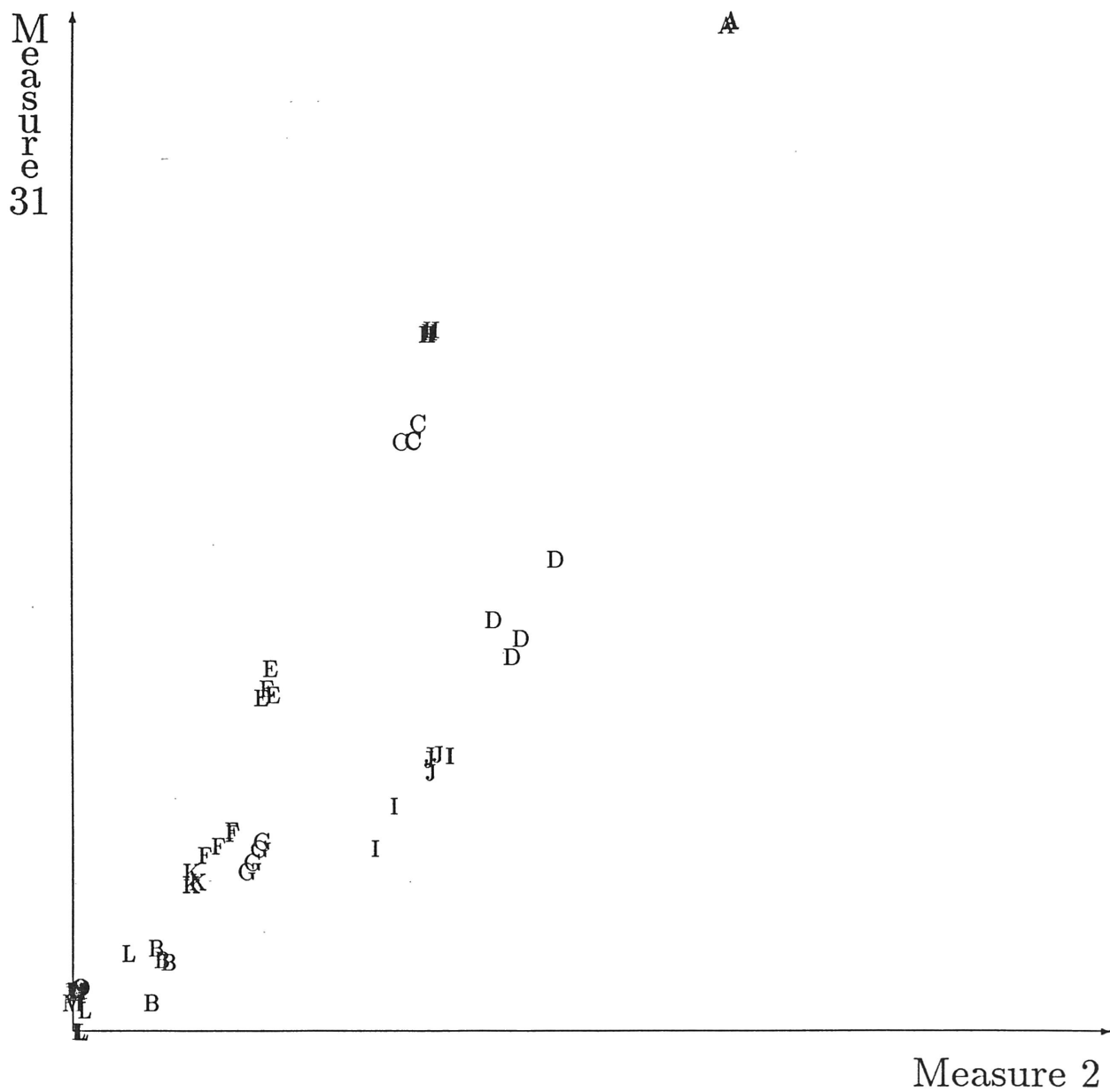


Figure 3. Measure 2 vs. Measure 31
for 61 shapes



measure 2,

$$\frac{1}{|S|} \int_S (|r - r_0| \cdot \mathbf{n}) d\sigma,$$

vs. measure 31,

$$\frac{1}{2} \left\{ \exp \left[\frac{1}{|S|} \int_S \log (2|(r - r_0) \cdot \mathbf{n}|) d\sigma \right] \right\}.$$

This plot seems to suggest some form of correlation in the measures, and brings up the question of statistical independence. We have used some of the recently studied dimensionality reduction (feature selection) techniques investigated and developed by Siedlecki *et al.* [15,16] and by Bryant and Guseman [17] to estimate the dimensionality of a particular set of shapes. Results using the principal components approach show that the dimensionality of the set of measures of the shapes we have tested is about three. Figure 4 is a plot of the dimensionally reduced data for band 1 vs. band 2, and Fig. 5 displays band 1 vs. band 3. Fig. 6 is a spatial plot of all three significant bands. These plots show excellent separation among our test shapes; the only objects that seem confused are those denoted by the letters I and J. Referring to plates 1 and 2, I denotes the shape in row 3 and column 3, and J denotes the shape in row 2 and column 4. These two shapes are intuitively very similar. In fact, the two shapes differ in only one direction; in that direction, I is stretched relative to J by a factor of $\frac{3}{2}$.

Finally, one might at first consider our work in connection with using time as the third dimension to view a moving 2-D shape as a 3-D object. However, it is clear that a rotated view of such a 3-D shape, even if a reasonable scaling of the time dimension could be found, would not be like the unrotated 3-D patterns in a real sense. For example, a stationary disc over a finite time period would be a cylinder. When rotated normal to its (time) axis, it becomes void (in time), then a line which grows to a rectangle and back to vanish. Magnifications would not be impossible, but it is hard to see a slowly moving small object as similar to a rapidly moving large (similar) object. Only translations are valid, and translation-invariant measures are easily obtained. Yet the idea is interesting.

Figure 4. Reduced Dimensionality Data
Band 1 vs. Band 2 for 61 Shapes

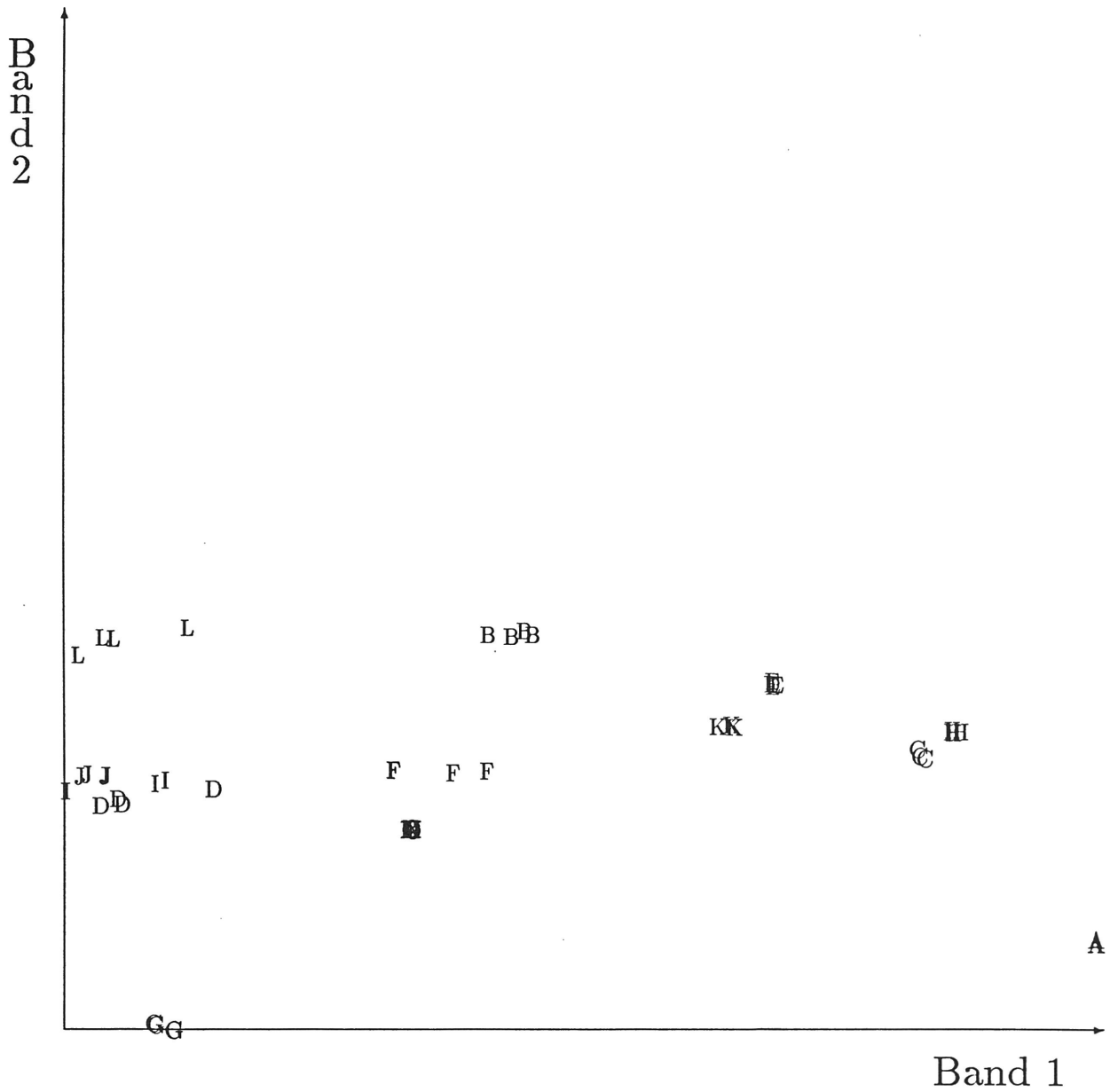


Figure 5. Reduced Dimensionality Data
Band 1 vs. Band 3 for 61 Shapes

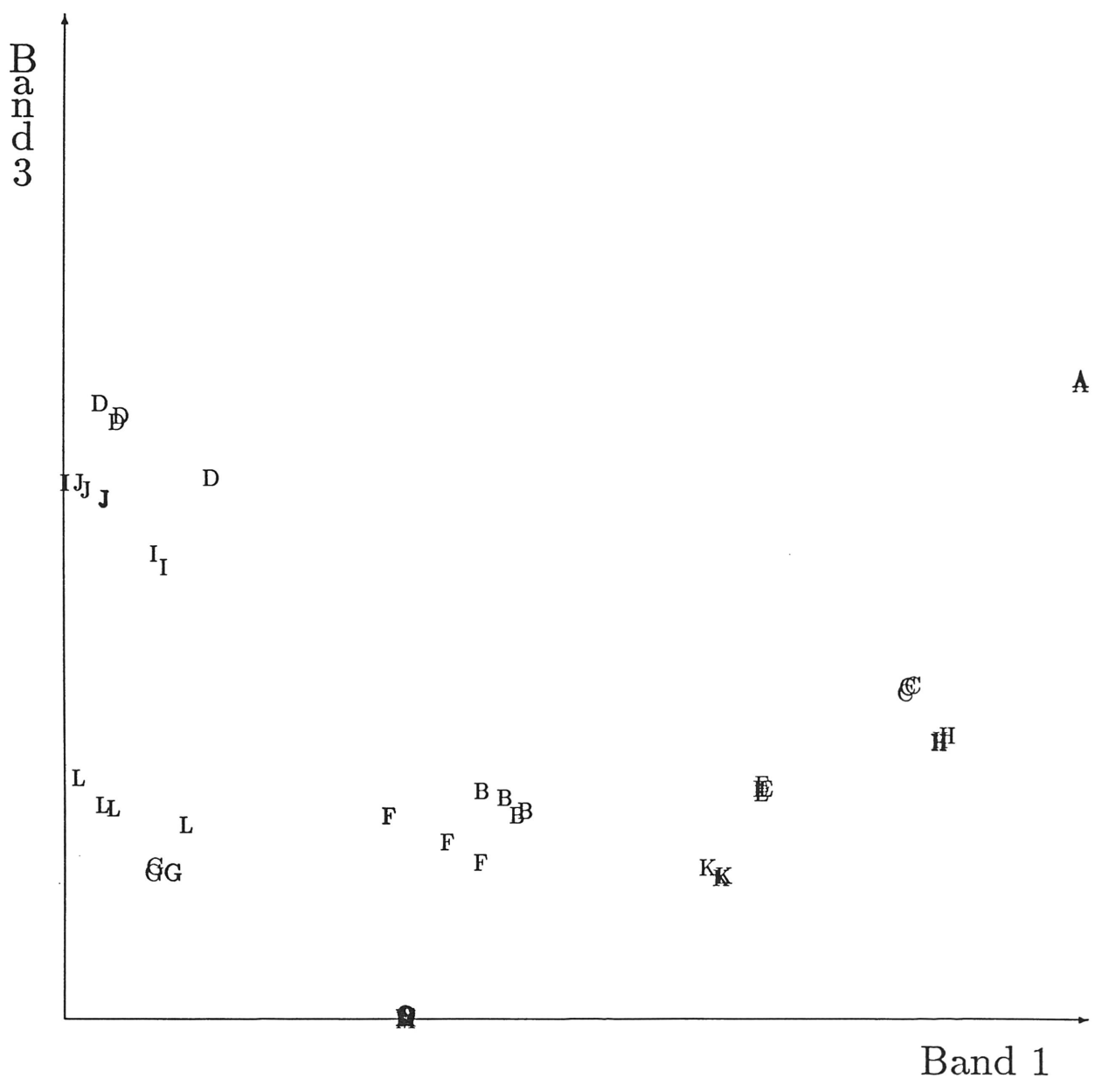
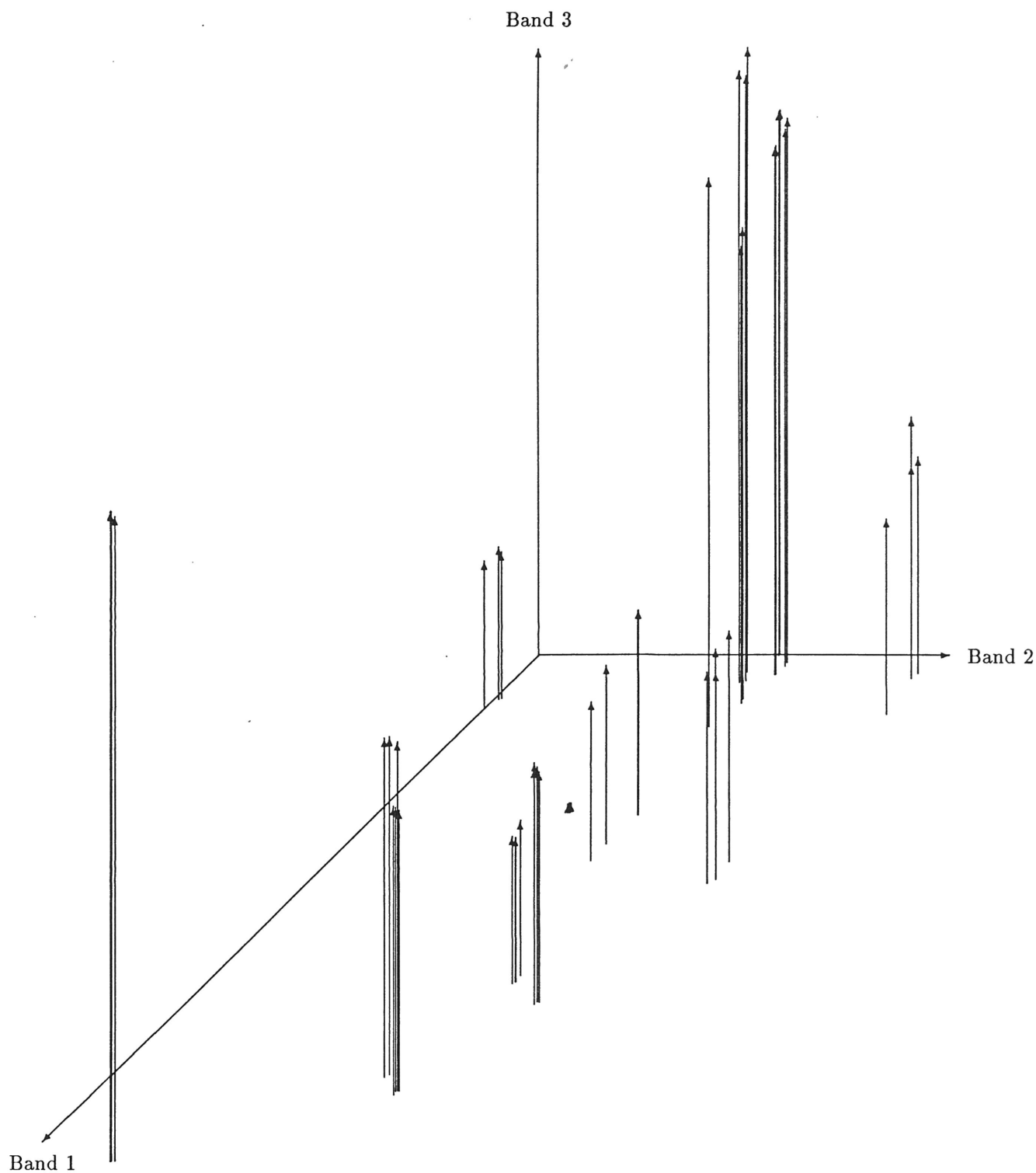


Figure 6. Reduced Dimensionality Data
Bands 1, 2 and 3 for 61 Shapes



V. THREE-DIMENSIONAL BINARY OBJECT DISPLAY

A. Preliminaries

While analyzing shape properties of three-dimensional binary objects, we found that it was necessary to gain an intuitive feeling for our data; we needed to see what we were working with. However, the display of 3-D discrete objects is difficult because the objects are not continuous. “Wire cage” displays of even moderately complex shapes are hard to visualize and the solution of the hidden line problem is time consuming. The wire cage outline of a complex low resolution discrete shape fails to convey an adequate feeling of the presumed underlying real world shape being modelled.

Ray-tracing produces interesting displays in which one often sees the reflection of one part of the image in another. However, the technique is computationally intensive and current techniques require parametric descriptions of the objects: It is not at all clear how it could be applied to a random three-dimensional binary scene. Levoy [9] has developed a method which uses 3-D volume data in which each volume element is a number (such as might be obtained using computed tomography), producing a shaded image. He uses a combination of ray-tracing and interpolation and thereby succeeds in displaying weak or fuzzy surfaces. Our interest is in producing comparable products but from low resolution binary data.

Recent research [10,11,12,13] by psychologists has revealed that shading and occlusion of objects in a two-dimensional picture can produce a compelling perception of three-dimensional shape. While this has been understood in a qualitative way since the middle ages, these studies seek to isolate the source of the perception. Illumination from the sun, although scattered by the atmosphere and reflected by nearby objects, is generally overhead. An animal in such an environment able to use subtle variations in shading to enhance perception must have an evolutionary advantage. For this reason, to convey an impression of three-dimensionality by a shaded planar image, the best direction of illumination is from the top of the object. We are dealing with the problem of presenting the shape to natural vision, a system which has evolved to deal with natural scenes. The trick is therefore to give the vision system a picture it would see were the shape illuminated from overhead relative to the viewer.

In producing the display it is not necessary to trace rays of light; in particular, if part of the shape would cast a shadow in another, then the absence of this shadow

does not significantly hinder visualization of the shape in three dimensions, and often assists visualization if the object would have large shadows. The psychologists also learned that perspective plays a much less important rôle than might be imagined, so that viewing can be modeled along parallel rays—that is, down a fixed direction which can be one of the three coordinate lines. These considerations can reduce computation time.

The experimental work carried out by psychologists used simple piecewise C^∞ shapes such as spheres on various backgrounds. The shapes we are investigating are binary and are not described parametrically. However, we thought it might be possible to use elementary approximation theoretic techniques to estimate parameters sufficient to determine the reflectance a discrete surface might have if it were continuous. Coupled with a simple reflectance model, we were able to display essentially arbitrary binary objects. The results exceeded our expectations.

B. Method

The shape is imbedded in a three-dimensional binary array; a cell is “in” the shape if its array value is 1. Down one of the axes the binary pattern is assumed packed in computer words. Think of this as the x -axis in a rectangular coordinate system; it is this axis which is taken to be the viewing direction. That is, the observer is at $(\infty, 0, \pi/2)$ in spherical coordinates looking parallel to the x -axis. The illumination direction is user-selectable, from (∞, θ, ϕ) in spherical coordinates; it is directly overhead (i.e., $\theta = 0, \phi = 0$) in the examples shown here. The general idea is to find the boundary (see [14]), find an approximate tangent plane at each boundary point which is visible from the view direction, and, using a reflectance model, calculate the shading observed.

The *boundary* of a discrete 3-D object is the set of points in the object with one of their six nearest neighbors not in the object. Call a boundary point *visible* if it is the first one encountered along the viewing line. Proceeding from a visible boundary point, collect points in the boundary near the point being observed. *Near* means one of the 36 nearest neighbors of the point which are not on the viewing line but which are boundary points. More precisely, suppose the point has (integer) coordinates (i, j, k) , with the first being the viewing direction coordinate. The points are the 24 nearest neighbors not on the viewing line with coordinates differing by at most one, plus the 10 points $(i, j \pm 2, k)$, $(i, j, k \pm 2)$, and $(i \pm 1, j \pm 2, k)$, $(i \pm 1, j, k \pm 2)$. In the shapes

studied here, eight to fourteen nearby boundary points were found for each boundary point.

We seek the best approximation of the form $x = ay + bz + d$ near the point (x, y, z) (with the viewing direction being the x -axis). That is, find the best (least squares) values of a and b which fit the “training” points collected. Let the points nearby be $\{(x_i, y_i, z_i) : i = 1, \dots, n\}$. The error in the approximation at a point is given by $e_i = x_i - (ay_i + bz_i + d)$. We seek to minimize the 2-norm

$$E(e_1, \dots, e_n) = \left(\sum_{i=1}^n e_i^2 \right)^{1/2},$$

which is a function of the three parameters a , b , and d . It is, of course, necessary that the gradient ∇E vanish when E has a local minimum (see [18]). These equations are called the *normal equations* for geometrical reasons; they are linear. The solution of the normal equations amounts the following linear algebra problem: let (all sums extend from 1 to n)

$$A = \begin{pmatrix} \sum y_i^2 & \sum y_i z_i & \sum y_i \\ \sum y_i z_i & \sum z_i^2 & \sum z_i \\ \sum y_i & \sum z_i & n \end{pmatrix},$$

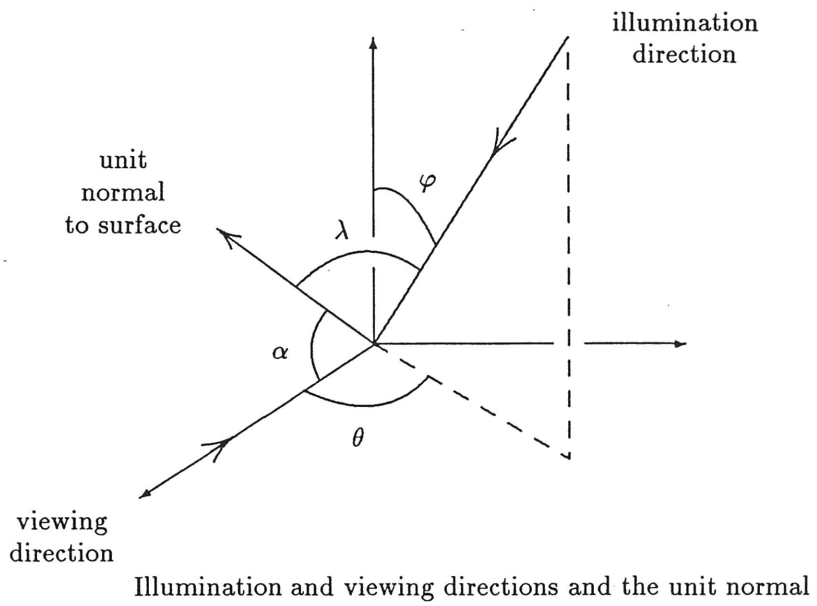
$$\mathbf{v} = \begin{pmatrix} \sum x_i y_i \\ \sum x_i z_i \\ \sum x_i \end{pmatrix}.$$

All arithmetic in accumulating the sums is exact.

We solve the equation $A\mathbf{u} = \mathbf{v}$ using Gaussian elimination ($\mathbf{u} = (a, b, d)^T$). As is typical of naïvely posed least squares problems, the matrix A tends to be ill-conditioned, but a highly accurate solution is not required. If the linear algebra portion of the least squares process indicates that no tangent plane of the requested form exists, mark the point for later processing during the filtering procedure (in the examples presented here this was never observed).

From the tangent plane the unit normal $(\cos \alpha, \cos \beta, \cos \gamma)$ is easily determined; it is the vector $(1, -a, -b)^T / (1 + a^2 + b^2)^{1/2}$. Refer to Fig. 7: the x -axis in this figure is the look direction. The relevant angles are the angle from the observer and from the illumination to the unit normal to the surface: The cosine of the angle λ from the normal to the illumination is now easily determined:

$$\cos \lambda = \cos \alpha \sin \phi \cos \theta + \cos \beta \sin \phi \sin \theta + \cos \gamma \cos \theta.$$



Two reflectance models are illustrated here: one, which we call *diffuse*, is $(1 + \cos \lambda) \cos \alpha$; another (*glossy*) is $[(1 + \cos \lambda) \cos \alpha]^3$. While these models are arbitrary, the $(1 + \cos \lambda)$ factor might be thought of as diffuse illumination from the source to the tangent plane (λ is then the zenith angle), and $\cos \alpha$ represents the energy intercepted by the observer. Raising these quantities to a power greater than 1 effectively simulates glossy surfaces, for large values are then relatively larger than small values. After further processing, the reflectances are scaled to fill the range 0-255 before being sent to the display device.

In addition to the two-dimensional map of reflectances, we keep a map of the x -values on the surface. This will be used in the following two steps.

An optional step allows the low resolution grey scale image to be filtered. While the least squares approximation is fast and never fails, and the neighborhood searched is large, the low resolution is bound to induce problems of inadequate sampling for highly irregular objects. While it seems strange at first, extremely smooth objects also lead to misleading displays. Filtering helps (as will be seen in the examples). A 5×5 filter is used; the weights are given by the following matrix, derived from the function

$$\cos \left(\frac{\pi}{6} [(y - y_0)^2 + (z - z_0)^2]^{1/2} \right)$$

which has its first zeros at $(y_0 \pm 3, z_0)$, $(y_0, z_0 \pm 3)$. The matrix used is

$$\begin{pmatrix} 0.090 & 0.389 & 0.500 & 0.389 & 0.090 \\ 0.389 & 0.738 & 0.866 & 0.738 & 0.389 \\ 0.500 & 0.866 & 1.000 & 0.866 & 0.500 \\ 0.389 & 0.738 & 0.866 & 0.738 & 0.389 \\ 0.090 & 0.389 & 0.500 & 0.389 & 0.090 \end{pmatrix}.$$

The user selects the number of times this filter is applied to the array of reflectances. In the examples given here, the array is filtered zero, one, two, and three times in the illustrations (proceeding left to right). The filter is only applied to points which are visible boundary points in planes close to the point at which it is being applied. (In effect all boundary points in the $5 \times 5 \times 5$ cube centered at the point are allowed.) While the filter smoothes the image, it can remove fine detail which may be of importance. In any case it is optional.

As a final step, we slightly filter and scale the array of reflectances. A simple 3×3 filter is used, with weights 1.0 (the center), 0.50 (the four nearest neighbors), and 0.20 for the next nearest four. This filter is relatively insensitive to ringing in spite of its

small size. Only those illuminations belonging to the same part of the surface (as determined by the saved x -values) are used. Use the value 0 for missing neighbors; this causes slight (and desirable) edge darkening, helping to reinforce the illusion that one part of the shape is occluding another (or the background). One purpose of the filter is to induce this edge darkening; in addition, the least squares procedure is least reliable on the edges where the actual tangent plane of the continuous shape is nearly parallel to the x -axis. Edge darkening is not induced by the 5×5 filter.

C. Examples and Technical Details

The example shapes are $128 \times 128 \times 128$ binary arrays. Recall that the shapes are constructed of spheres so that they can be easily rotated and magnified by rotating and magnifying the parameters (i.e., the centers and radii) exactly and regenerating the shapes.

One needs to be aware that this method does not produce the same effect as actually rotating the discrete shape and somehow resampling. For instance, one distracting feature of the images are the strange “circles” that appear along the viewing angle on the spheres. It is important to realize that these circles are actually present in the discrete data; that is, they are not a side effect of the display program. All discrete points in any given ring (and the central disk) lie in a plane parallel to the y - z plane; they are a consequence of the discretization process. Rotating a ball and regenerating it would not rotate these ringed flat spots, for they would remain fixed along the viewing line. Contrast-involved side effects are even more distracting: the disk appears to be concave even though the shape is convex. While the illumination in the disk is really constant, the illuminated sphere gets darker as one looks down past the bright spot. In fact, if the flat spot is large enough to be easily resolved, then the vision system will interpret it to be a concave dimple. Smoothing does not correct this; smoothing often *enhances* the perception of concavity.

As an example, consider a discrete sphere of radius 50 centered at the origin (the set of lattice points $\{(i, j, k) : i^2 + j^2 + k^2 < 2500\}$), and examine the point $(49, 0, 0)$. It is a boundary point because it is in the object and the neighboring point $(50, 0, 0)$ is not in the object. The plane circular area $\{(49, j, k) : j^2 + k^2 < 99\}$ containing this point is in the boundary of this sphere; it is a relatively large flat spot consisting of 305 points with diameter about 19. Surrounding it is a circular annulus of 316 boundary

points $\{(48, j, k) : 97 \leq j^2 + k^2 < 196\}$, which appears to be a band of width slightly over 4. Two more bands are just barely visible, of 324 points with width about 4 and 316 points with width about 3. In the circle, and down the center of the first ring, the tangent plane found by our procedure will have parameters a and b equal to zero, so that $\cos \alpha = 1$, $\cos \lambda = 0$ (assuming directly overhead illumination). A filter designed to remove the bands will necessarily be at least 5×5 . Not much can be done about the center disk, but the first band can be removed by filtering. On the other hand, smaller balls suffer much less from this visual artifact.

This is an instance of a problem one encounters in the process of taking a continuous object, sampling to produce a discrete object, and then viewing a reconstruction as a continuous object: *Viewing resolution can be too fine, given a fixed sampling resolution.* There are conflicting views on how to handle this problem. The concave spots and rings perceived on the larger spheres are diverting. However, the underlying data which produce the artifact are present in the discrete shape, and it should be remembered that discretization is not a one-to-one process; many continuous shapes may be represented by the same discrete model. One view would have the shading reflect only the data that is present without adding any assumptions on the smoothness of the “real world” shape. Alternatively, the crinkly, hammered, appearances of the unfiltered shapes may or may not distract from the interpretation of the artificial image, depending entirely on the point of view of the user: one interested in global structure might prefer the filtered image.

D. Plates

The shaded versions of the shapes are shown in plates 3 and 4; a 512×512 grey scale image was made from the shaded versions using eight bit (0-255) grey levels. Since the underlying binary objects are $128 \times 128 \times 128$, sixteen 128×128 shaded “images” of the binary shape fit in the plate. The plates were photographed directly from the video monitor.

All computations were performed on a low-end VAXstation 2000 without floating-point hardware; a little over a minute was required to make the grey scale image of one $128 \times 128 \times 128$ binary image. The grey scale images, produced by mosaicing sixteen 128×128 displays, were displayed on a Silicon Graphics, Inc. IRIS graphics workstation. In addition to the high quality display, a quick look at the shaded shape

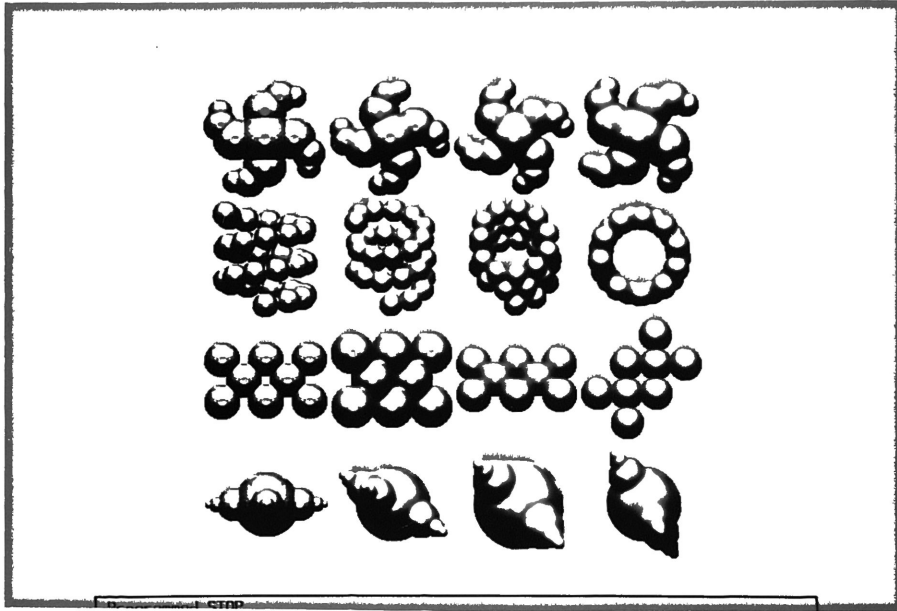


Plate 3

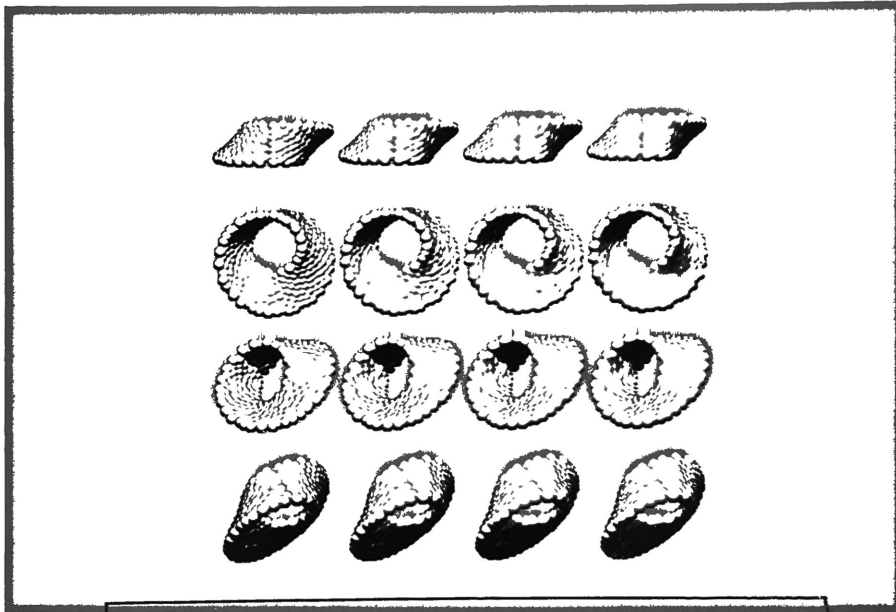
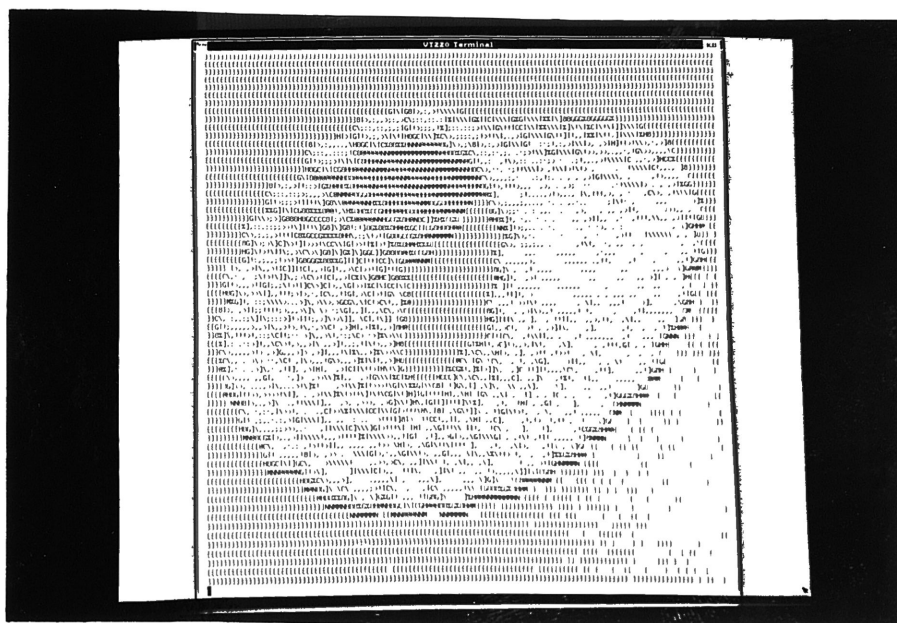


Plate 4

is displayed on the workstation console using a width of 132 characters and length of 64. A useful view of the shape is obtained by printing every-other-line with the sixteen characters (including blank) "Hhes%gc|\!|,;.:", the first characters corresponding to the lower reflectance values. The background is represented by the texture made of alternating lines of "{" and "}". A screen photograph of that display is included as plate 5.



SUMMARY

Theoretical results have been presented that quantify the intuitive concept of shape and provide a method for developing new shape measures. These measures can be used by a computer to compare the shapes of objects in its memory. An arbitrary method for representing objects as binary arrays has been used, and a method for the approximation of surface integrals over the boundary of a low-resolution binary object has been presented. These integrals, an approximation to the unit normal, and an approximation to the differential of surface area make it possible to compute several shape measures and near shape measures of binary objects. These measures were tested and analyzed for independence and significance. A method for producing a shaded two-dimensional representation of a three-dimensional binary object was discussed.

ACKNOWLEDGEMENTS

We used the Cornell National Supercomputer Facility's IBM 3090-600E supercomputers to generate complex images that would otherwise have not been feasible. The author is part of the CNSF's 1988-1989 Research Program for Undergraduates and Faculty, and deeply appreciates the opportunities afforded them by the Facility. The CNSF is a resource of the Center for Theory and Simulation in Science and Engineering. The Center receives major funding from the National Science Foundation and IBM Corporation, with additional support from New York State and members of the Corporate Research Institute. Susan Mehringer and Helen Doerr of the CNSF were particularly helpful.

Alan Bryant took the screen photographs and entered into several conversations on the problem. The referees of our two papers, "Display of Discrete Three-Dimensional Binary Objects: I-Shading," and "Shape Measures and Discrete Surface Integrals in Space," made many suggestions that strengthened this presentation. Finally, my mentor, Dr. Jack Bryant was an endless source of inspiration, provocation, and productive frustration throughout my Honors Fellow's project. Decisions I have made under his wing will be with me long after my interest in three-dimensional binary objects has faded.

REFERENCES

- [1] J. Sklanski. Recognition of convex blobs. *Pattern Recognition*, 2:3–10, 1970.
- [2] L. D. Harmon, M. K. Khan, R. Lasch, and P. F. Ramig. Machine identification of human faces. *Pattern Recognition*, 13:97–110, 1981.
- [3] S. K. Parui and D. Dutta Majumder. Symmetry analysis by computer. *Pattern Recognition*, 16:63–67, 1983.
- [4] L. Gupta and M. D. Srinath. Contour sequence moments for the classification of closed planar shapes. *Pattern Recognition*, 20:267–272, 1987.
- [5] A. Bryant and J. Bryant. Recognizing shapes in planar binary images. *Pattern Recognition*, 22, 1989 (to appear).
- [6] T. Okada and S. Tsuchiya. Object recognition by grasping. *Pattern Recognition*, 9:111–119, 1977.
- [7] T. M. Silberberg, L. Davis, and D. Harwood. An iterative Hough procedure for three-dimensional object recognition. *Pattern Recognition*, 17:621–629, 1984.
- [8] R. A. Reynolds, D. Gordon, and L.-S. Chen. A dynamic screen technique for shaded graphics display of slice-represented objects. *Computer Vision, Graphics, and Image Processing*, 38:275–298, 198.
- [9] M. Levoy. Display of surfaces from volume data. *IEEE Comput. Graphics & Appl.*, 29–37, May 1988.
- [10] A. Yonas, M. Kuskowski, and Susan Sternfels. The role of frames of reference in the development of responsiveness to shading. *Child Development*, 50:495–500, 1979.
- [11] J. T. Todd and E. Mingolla. Perception of surface curvature and direction of illumination from patterns of shading. *J. Experimental Psychology: Human Perception and Performance*, 9:585–595, 1983.
- [12] V. S. Ramachandran. Perceiving shape from shading. *Nature*, 331:133–166, 1988.

- [13] V. S. Ramachandran. Perceiving shape from shading. *Scientific American*, 259(2):76–83, August 1988.
- [14] A. Bryant and J. Bryant. Following boundaries of discrete binary objects in space. *Pattern Recognition*, 1989 (submitted).
- [15] W. Siedlecki, K. Siedlecka, and J. Sklansky. An overview of mapping techniques for exploratory pattern analysis. *Pattern Recognition*, 21:411–429, 1988.
- [16] W. Siedlecki, K. Siedlecka, and J. Sklansky. Experiments on mapping techniques for exploratory pattern analysis. *Pattern Recognition*, 21:431–438, 1988.
- [17] J. Bryant and L. F. Guseman, Jr. Distance preserving linear feature selection. *Pattern Recognition*, 11:347–352, 1979.
- [18] S. D. Conte and C. di Boor. *Elementary Numerical Analysis. International series in pure and applied mathematics*, McGraw-Hill, New York, third edition, 1980.

A RATIONALIZATION OF A HOMOGENEOUS CONGRUENCE MEASURE

We show here that

$$\left[\frac{1}{|S|} \int_S \|r - r_0\|^p |(r - r_0) \cdot \mathbf{n}|^q d\sigma \right]^{1/(p+q)}$$

is a homogeneous congruence measure.

Consider the top illustration in Fig. 8. Here, S is a piece of a continuous shape, and r_0 is its centroid. Let P be some point on the surface of S , and \mathbf{r} be the vector from r_0 to P . Let \mathbf{n} be the unit normal of S at P . Now, consider the bottom illustration in Fig. 8. S has not changed shape, but has rotated clockwise 60 degrees. Notice that $\|\mathbf{r}\|$ and $\mathbf{r} \cdot \mathbf{n}$ do not change as S rotates. Since this holds for arbitrary P on the surface of S ,

$$\int_S \|\mathbf{r}\| |\mathbf{r} \cdot \mathbf{n}| d\sigma$$

is a congruence measure (we take $|\mathbf{r} \cdot \mathbf{n}|$ for computational convenience only. The result holds if the absolute value marks are removed). Notice that, if f and g are arbitrary functions defined on the positive real numbers,

$$\int_S f(\|\mathbf{r}\|) \cdot g(|\mathbf{r} \cdot \mathbf{n}|) d\sigma$$

is a congruence measure. Thus

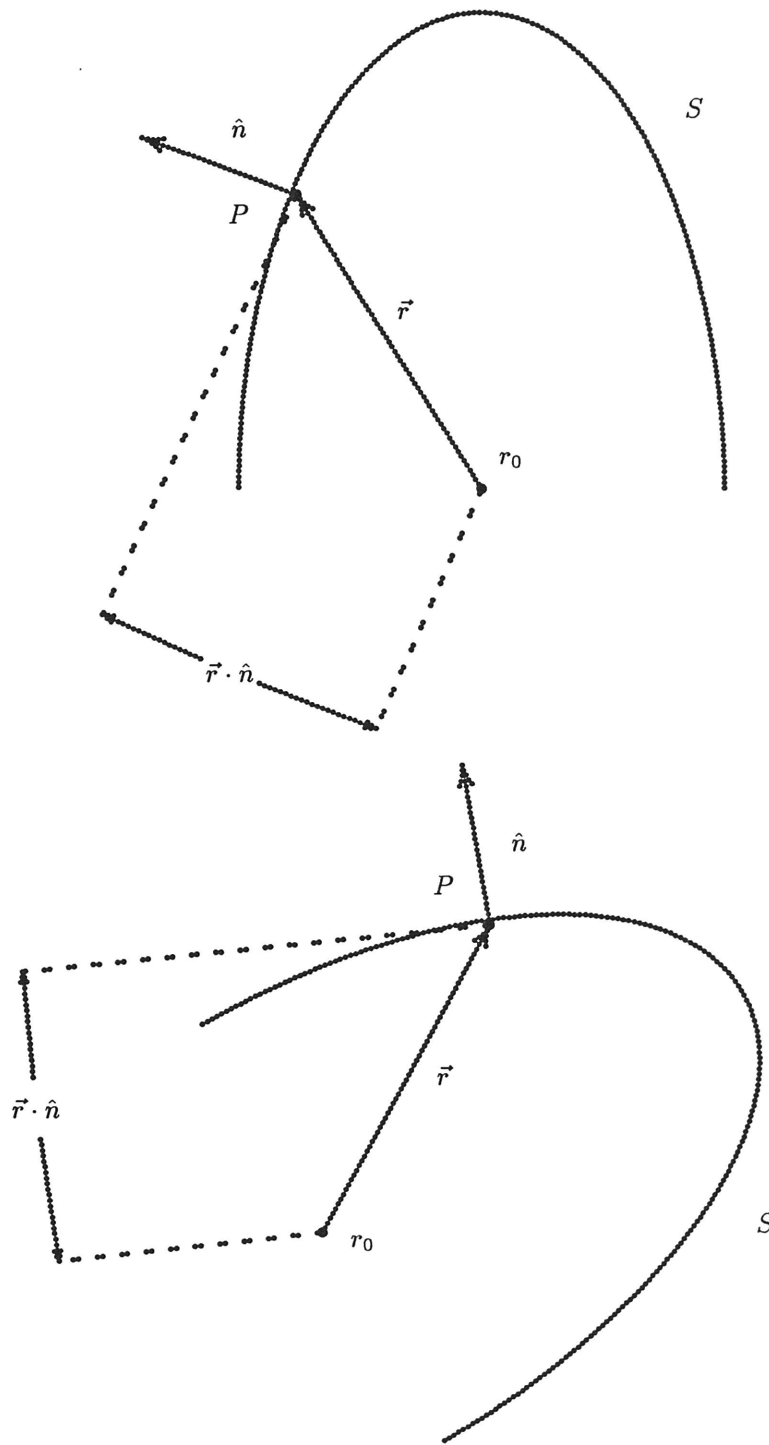
$$\int_S (\|\mathbf{r}\|)^p (|\mathbf{r} \cdot \mathbf{n}|)^q d\sigma$$

is a congruence measure; simple unit analysis shows that

$$\left[\frac{1}{|S|} \int_S (\|\mathbf{r}\|)^p (|\mathbf{r} \cdot \mathbf{n}|)^q d\sigma \right]^{1/(p+q)}$$

has linear dimension and is, therefore, a homogeneous congruence measure (of course, using different notation, $\|\mathbf{r}\| = \|r - r_0\|$).

Figure 1. Two Rotations of a Shape



B BEST FITTING TANGENT PLANE

We prove here that the best fitting tangent plane is found looking in the direction with maximum 2×2 determinant. It is enough to suppose the point (x_0, y_0, z_0) is the origin. We accordingly consider the problem of approximating $\{(x_i, y_i, z_i) : i = 1, \dots, n\}$ with one of the planes of the form $x = by + cz$, $y = ax + cz$ or $z = ax + by$. Let $\mathbf{x} = (x_1, \dots, x_n)$, $\mathbf{y} = (y_1, \dots, y_n)$, and $\mathbf{z} = (z_1, \dots, z_n) \in R^n$. The three determinants described above in Step 3 become (where $\|\mathbf{x}\|^2 = \sum x_i^2$, $\mathbf{x} \cdot \mathbf{y} = \sum x_i y_i$)

$$A = \begin{vmatrix} \|\mathbf{y}\|^2 & \mathbf{y} \cdot \mathbf{z} \\ \mathbf{y} \cdot \mathbf{z} & \|\mathbf{z}\|^2 \end{vmatrix} \quad B = \begin{vmatrix} \|\mathbf{x}\|^2 & \mathbf{x} \cdot \mathbf{z} \\ \mathbf{x} \cdot \mathbf{z} & \|\mathbf{z}\|^2 \end{vmatrix} \quad C = \begin{vmatrix} \|\mathbf{x}\|^2 & \mathbf{x} \cdot \mathbf{y} \\ \mathbf{x} \cdot \mathbf{y} & \|\mathbf{y}\|^2 \end{vmatrix}.$$

Consider, for example, the approximation $x = by + cz$. The least squares problem is easily seen to be

$$\begin{bmatrix} \|\mathbf{y}\|^2 & \mathbf{y} \cdot \mathbf{z} \\ \mathbf{y} \cdot \mathbf{z} & \|\mathbf{z}\|^2 \end{bmatrix} \begin{bmatrix} b \\ c \end{bmatrix} = \begin{bmatrix} \mathbf{x} \cdot \mathbf{y} \\ \mathbf{x} \cdot \mathbf{z} \end{bmatrix}.$$

After solving for b and c and evaluating the error we obtain

$$Error^2 = \frac{\|\mathbf{x}\|^2 \|\mathbf{y}\|^2 \|\mathbf{z}\|^2 + 2 \mathbf{x} \cdot \mathbf{y} \mathbf{x} \cdot \mathbf{z} \mathbf{y} \cdot \mathbf{z} - \|\mathbf{x}\|^2 \mathbf{y} \cdot \mathbf{z}^2 - \|\mathbf{y}\|^2 \mathbf{x} \cdot \mathbf{z}^2 - \|\mathbf{z}\|^2 \mathbf{x} \cdot \mathbf{y}^2}{A}.$$

Notice that the numerator is symmetric in \mathbf{x} , \mathbf{y} , and \mathbf{z} ; accordingly, the minimal error is obtained when the non-negative determinant in the denominator is largest.

Remark:

The result is easily generalized to approximation in n -dimensions. Let $\{\mathbf{x}_i : i = 1, \dots, m\}$ be a set of vectors in R^n . For each $j \leq m$ let A_j be the $j \times j$ matrix

$$A_j = (\mathbf{x}_\mu \cdot \mathbf{x}_\nu)_{j \times j}.$$

Consider the approximation problem (for $2 \leq j \leq m$)

$$\text{minimize} \quad \|\mathbf{x}_j - \sum_{i=1}^{j-1} \alpha_i \mathbf{x}_i\|^2$$

A tedious but straightforward calculation leads to the error equation (provided $\det A_{j-1} > 0$)

$$Error^2 = \det A_j / \det A_{j-1}$$

The matrix A_j is clearly positive definite, for if $\mathbf{u} \in R^n$ then

$$\mathbf{u}^T A_j \mathbf{u} = \sum_{i=1}^j (\mathbf{u} \cdot \mathbf{x}_i)^2 \geq 0.$$

Therefore, $\det A_j \geq 0$. It is enough to assume $\det A_j > 0$, for if $\det A_j = 0$ then the set $\{\mathbf{x}_i : i = 1, \dots, j\}$ is linearly dependent and we could omit some of the \mathbf{x}_i without changing the error in linear approximations.

C PROGRAMS

These are the programs we developed for this project. The first, DISPF.FOR, makes a sphere file that maps out a parametric three-dimensional curve. The second, ROTCALL5.FOR, rotates an existant sphere file through user-supplied angles. The third, DISPIT2.FOR, generates binary data from sphere files. The last, EXTR.FOR, produces a displayable image from the binary data.

```

PROGRAM DISPF

C   This program is supposed to write a data file containing
C   enough data to allow a dispit version of any given
C   three dimensional parametric function.

IMPLICIT NONE
REAL Overlap
INTEGER PointCount
REAL OldX, OldY, OldZ
REAL Radius
REAL T, Inc
REAL TMax
CHARACTER*20 OutputFile

REAL X, Y, Z
REAL Dist

Dist(X,Y,Z) = SQRT(X**2+Y**2+Z**2)
INCLUDE 'THEFUNCTIONS.DISPF'

WRITE (*,*) 'Put the parametric functions in THEFUNCTIONS.DISPF'
WRITE (*,*) 'and compile DISPF.FOR; then'
WRITE (*,*) 'input T0, TMax, inc0, the radius,'
WRITE (*,*) 'and the Overlap Factor (0=None,1=All)'
READ (*,*) T, TMax, Inc, Radius, Overlap
WRITE (*,*) 'And where do you want these points written?'
READ (*,1) OutputFile
1  FORMAT (A20)
   OPEN (UNIT = 12, FILE = OutputFile, STATUS = 'NEW')

PointCount = 0
OldX = X(T)
OldY = Y(T)
OldZ = Z(T)
10  CONTINUE
   IF (T .LT. TMax) THEN
     PointCount = PointCount + 1
     WRITE (12,*) OldX, OldY, OldZ, Radius
     20  CONTINUE
     IF (Dist ((OldX-X(T)),(OldY-Y(T)),(OldZ-Z(T)))
        &      .LT. (1.-Overlap)*Radius) THEN
       T = T + Inc
       GO TO 20
     ENDIF
     OldX = X(T)
     OldY = Y(T)
     OldZ = Z(T)
     GO TO 10
   ENDIF

WRITE (*,*) 'I wrote ', PointCount, ' points.'
STOP
END

```

```

PROGRAM RotCall5

C   This puppy tries to read the data from a disk.
C   And tries to implement psi.
C   And immediately calls orientate and scaleit.
C   And allows two sets of rotations and a specified final ball radius.

IMPLICIT NONE
real X,Y,Z,X1,Y1,Z1,X2,Y2,Z2,Radius
real Theta1, Phi1, Psi1, Theta2, Phi2, Psi2
REAL SpecRad
CHARACTER*50 DataFile

REAL DegToRad
DegToRad (Theta1) = 0.017453293*Theta1

write (*,*) 'Yo. Enter the name of the data file now.'
read (*,1) DataFile
1  FORMAT (A20)
OPEN (UNIT = 11, FILE = DataFile, STATUS = 'OLD')
OPEN (UNIT = 12, FILE = 'ROTATE.OUT', STATUS = 'NEW')

write (*,*) 'Joko, and what are the first Theta, Phi, and Psi?'
READ (*,*) Theta1, Phi1, Psi1
WRITE (*,*) 'Uh huh. And the second Theta, Phi, and Psi?'
READ (*,*) Theta2, Phi2, Psi2
WRITE (*,*) 'Hmmm. Specified Radius (zero if apathetic)?'
READ (*,*) SpecRad

Theta1 = DegToRad (Theta1)
Phi1 = DegToRad (Phi1)
Psi1 = DegToRad (Psi1)

Theta2 = DegToRad (Theta2)
Phi2 = DegToRad (Phi2)
Psi2 = DegToRad (Psi2)

5000 CONTINUE
READ (11, *, END = 10000) X, Y, Z, Radius

CALL Rotate (X, Y, Z, Theta1, Phi1, Psi1, X1, Y1, Z1)
CALL Rotate (X1, Y1, Z1, Theta2, Phi2, Psi2, X2, Y2, Z2)

write (12 ,*) X2, Y2, Z2, Radius

GO TO 5000
10000 CONTINUE

CLOSE (11)
CLOSE (12)

CALL ORIENTATE
CALL SCACALL4(SpecRad)

STOP
END

SUBROUTINE Rotate (X0,Y0,Z0,Theta,Phi,Psi,X1,Y1,Z1)

C   Another attempt at Theta, Phi, Psi rotation.

IMPLICIT NONE
REAL X0,Y0,Z0,X1,Y1,Z1
REAL Theta, Phi, Psi

REAL Pi
PARAMETER (Pi = 3.141592654)

REAL Rho,R1,Phi1,Theta1
REAL Theta0,Phi0,Psi0

Theta0 = 0
Phi0 = 0
Psi0 = 0

IF (X0 .NE. 0 .OR. Y0 .NE. 0)
&   Theta0 = ATAN2 (Y0,X0) + Theta

```



```

Rho = SQRT (X0**2 + Y0**2)
X1 = Rho * COS (Theta0)
Y1 = Rho * SIN (Theta0)

IF (X1 .NE. 0 .OR. Z0 .NE. 0)
&   Phi0 = ATAN2 (Z0,X1) + Phi
Rho = SQRT (X1**2 + Z0**2)
X1 = Rho * COS (Phi0)
Z1 = Rho * SIN (Phi0)

IF (Y1 .NE. 0 .OR. Z1 .NE. 0)
&   Psi0 = ATAN2 (Z1,Y1) + Psi
Rho = SQRT (Y1**2 + Z1**2)
Y1 = Rho * COS (Psi0)
Z1 = Rho * SIN (Psi0)

RETURN
END

SUBROUTINE Standardize (Theta, Phi)

IMPLICIT NONE
REAL Theta, Phi

REAL Pi
PARAMETER (Pi = 3.141592654)

1000 CONTINUE
IF (Phi .GT. Pi) THEN
  Phi = Phi - 2 * Pi
  GOTO 1000
ELSE IF (Phi .LT. -Pi) THEN
  Phi = Phi + 2 * Pi
  GOTO 1000
ENDIF

IF (Phi .GT. Pi/2) THEN
  Phi = Pi - Phi
  Theta = Theta - Pi
ELSE IF (Phi .LT. -Pi/2) THEN
  Phi = -Pi - Phi
  Theta = Theta - Pi
ENDIF

RETURN
END
SUBROUTINE ScaCall4(SpecRad)

C   This one implements a specified radius

IMPLICIT NONE
INTEGER MaxSpheres
PARAMETER (MaxSpheres = 1000)

REAL Centers(MaxSpheres,3), Radii(MaxSpheres)
REAL NewC(MaxSpheres,3),NewR(MaxSpheres)
INTEGER i, j, NumSpheres
REAL SpecRad

OPEN (unit=13,file='Orient.out',status='old')
OPEN (unit=14,file='Scaleit.out',status='new')

I = 1
5   CONTINUE
  READ (13,*, END = 10000) (Centers(I,J),J=1,3), Radii(I)
  I = I + 1
  GO TO 5
10000 CONTINUE
  NumSpheres = I - 1

  WRITE (*,*) 'SCALEIT: Read ', NumSpheres, ' spheres.'
  call ScaleIt2 (Centers, Radii, NumSpheres, SpecRad)
55  CONTINUE
  call CenterIt (Centers, Radii, NumSpheres, NewC, NewR)
  DO 20 i = 1, NumSpheres
    WRITE (14,*) (NewC(i,j),j=1,3),NewR(i)
20  CONTINUE

```

```

write (14,*) 0,0,0,0

CLOSE (13)
CLOSE (14)

STOP
END
SUBROUTINE ScaleIt2 (Centers,Radii,NumSpheres,SpecRad)

IMPLICIT NONE
INTEGER MaxSpheres
PARAMETER (MaxSpheres = 1000)
INTEGER NumSpheres
REAL Centers(MaxSpheres,3), Radii(MaxSpheres)
REAL SpecRad

INTEGER I, J
REAL MinRatio
INTEGER NRows, NCols, NPlas
PARAMETER (NRows = 128)
PARAMETER (NCols = 128)
PARAMETER (NPlas = 128)

REAL Maxs(3), Mins(3)

IF (NumSpheres .GT. 0) THEN
  IF (SpecRad .EQ. 0) THEN
    DO 10 I = 1,3
      Mins(i) = Centers(1,I) - Radii(1)
      Maxs(i) = Centers(1,I) + Radii(1)
      DO 15 J = 2, NumSpheres
        Mins(I) = MIN (Mins(I), Centers(J,I) - Radii(J))
        Maxs(I) = MAX (Maxs(I), Centers(J,I) + Radii(J))
15      CONTINUE
10      CONTINUE

      MinRatio = MIN (REAL(NRows-4)/(Maxs(1)-Mins(1)),
&                  REAL(NCols-4)/(Maxs(2)-Mins(2)),
&                  REAL(NPlas-4)/(Maxs(3)-Mins(3)))
      ELSE
        MinRatio = SpecRad/Radii(1)
      ENDIF

      DO 20 J = 1, NumSpheres
        Radii(J) = Radii(J)*MinRatio
        DO 20 I = 1,3
          Centers(J,I) = (Centers(J,I)-Mins(I))*MinRatio+2.0
20      CONTINUE

      ENDIF

      RETURN
    END
  SUBROUTINE Orientate

  IMPLICIT NONE
  INTEGER MaxSpheres
  PARAMETER (MaxSpheres = 1000)

  CHARACTER*20 DataFile
  INTEGER NumSpheres
  REAL Centers(MaxSpheres,3), Radii(MaxSpheres)
  INTEGER I, J

  OPEN (UNIT = 11, FILE = 'Rotate.out', STATUS = 'Old')
  OPEN (UNIT = 12, FILE = 'Orient.out', STATUS = 'New')

  I = 1
10  CONTINUE
    READ (11,*, END = 10000) (Centers(I,J),J=1,3), Radii(I)
    I = I + 1
    GO TO 10
10000 CONTINUE

    NumSpheres = I - 1
    write (*,*) 'ORIENTATE: Read ', NumSpheres, ' spheres.'
    DO 20 I = 1, NumSpheres

```

```

        Centers(I,1) = -Centers(I,1)
        Centers(I,3) = -Centers(I,3)
        write (12,*) (Centers(I,J),J=1,3), Radii(I)
20    CONTINUE

    CLOSE (11)
    CLOSE (12)

    RETURN
    END
    SUBROUTINE CenterIt(Centers, Radii, NumSpheres, NewC, NewR)

C    This routine doesn't work properly; it must be fixed.

    IMPLICIT NONE
    INTEGER MaxSpheres
    PARAMETER (MaxSpheres = 1000)
    INTEGER NumSpheres
    REAL Centers(MaxSpheres,3), Radii(MaxSpheres)
    REAL NewC(MaxSpheres,3), NewR(MaxSpheres)

    INTEGER NRows, NCols, NPlas
    PARAMETER (NRows = 128)
    PARAMETER (NCols = 128)
    PARAMETER (NPlas = 128)

    INTEGER I, J
    REAL Mins(3), Maxs(3), FreeSpace(3)

    IF (NumSpheres .GT. 0) THEN
        DO 30 I = 1, 3
            Mins(i) = Centers(1,I) - Radii(1)
            Maxs(i) = Centers(1,I) + Radii(1)
            DO 25 J = 2, NumSpheres
                Mins(i) = MIN (Mins(i), Centers(J,I) - Radii(J))
                Maxs(i) = MAX (Maxs(i), Centers(J,I) + Radii(J))
25            CONTINUE
30        CONTINUE

            FreeSpace(1) = REAL(NRows) - (Maxs(1) - Mins(1))
            FreeSpace(2) = REAL(NCols) - (Maxs(2) - Mins(2))
            FreeSpace(3) = REAL(NPlas) - (Maxs(3) - Mins(3))

            DO 40 I = 1, NumSpheres
                NewR(I) = Radii(I)
                DO 40 J = 1, 3
                    NewC(I,J) = .5*FreeSpace(J) - Mins(J)
                    & + Centers(I,J)
40            CONTINUE
        ENDIF

    RETURN
    END

```

```

PROGRAM DISPIT2

IMPLICIT NONE

C Program to display a shape on a gray-scale display
C SCALE is the number of levels of gray available

INTEGER SCALE
PARAMETER (SCALE=255)
C PI is pi
REAL PI
PARAMETER (PI=3.14159)

C THETA is the polar coordinate offset
REAL THETA

C PHI IS THE ANGLE FROM AZIMUTH OF ILLUM
REAL PHI

c U1,U2,U3 is a unit vector in the direction of the illum angle
REAL U1,U2,U3

C NBITS -- number of bits in an integer = number of data bus bits
C NROW -- number of rows
C NROWB -- NROW/NBITS
C NCOL -- number of columns
C NPLA -- number of planes
C EDGES -- a binary 3-D image of the boundary
C IMAGE -- the binary set of objects

INTEGER NBITS,NROW,NROWB,NCOL,NPLA,EDGES,IMAGE,BIGBIT
PARAMETER (NBITS=32,NROW=128,NROWB=NROW/NBITS)
PARAMETER (NCOL=128,NPLA=128)

C BIGBIT - an integer with the first bit on and the rest off
PARAMETER (BIGBIT=-2**(NBITS-2)-2**(NBITS-2))

C D,FLAG,SHFLAG are used to extract the boundary using fast logical operations
C I,J,K,L are DO indexes
INTEGER D,FLAG,SHFLAG,I,J,K,L

C DISP is the grey scale image to make; it is real until scale factors
C and filtering are complete.
REAL DISP

C PLANE -- this array contains the plane a point is in; used to avoid filtering
C using reflectances from actually different objects with close
C projections.
INTEGER*2 PLANE

C FILTER -- the number of times to 5x5 filter the reflectances
INTEGER FILTER
COMMON/GSCALE/ DISP(NCOL,NPLA),PLANE(NCOL,NPLA)
COMMON/ARRAY/ EDGES(O:NROWB-1,NCOL,NPLA),
+ IMAGE(O:NROWB-1,NCOL,NPLA)

C DTAFGL -- determines whether one reads the image from a file or
C derives it internally from parameters which are input at run time.
LOGICAL DTAFGL
CHARACTER*40 INNAME

C SHINFL is the reflectance parameter
INTEGER SHINFL
PRINT*, ' THETA, PHI (IN DEGREES)'
READ*,THETA,PHI
THETA = THETA*PI/180

```

```

    PHI = PHI*PI/180
    PRINT*, ' GENERATE (T) OR READ (F) DATA'
    READ*,DTAFLG
    PRINT*, ' INPUT FILE NAME'
    READ15,INNAME
    OPEN (10,FILE=INNAME,DEFAULTFILE='FORO10.DIS',
    &      FORM='UNFORMATTED',STATUS='NEW')
15  FORMAT(A)
    PRINT*, ' TYPE OF REFLECTANCE: (1) EGG SHELL TO (4) GLOSSY'
    READ*,SHINFL
    PRINT*, ' HOW MANY TIMES TO FILTER THE 3-D IMAGE?'
    READ*,FILTER

C This is the unit vector in the direction of the illumination angle

    U1 = SIN(PHI)*COS(THETA)
    U2 = SIN(PHI)*SIN(THETA)
    U3 = COS(PHI)

C Get the image

    CALL MAKEIT(DTAFLG,INNAME)

C Process it--that is, find the edges

C First the image shifted up one plane
    DO 30 K = 2,NPLA
      DO 30 J = 1,NCOL
        DO 30 I = 0,NROWB-1
          30      EDGES(I,J,K) = IAND(IMAGE(I,J,K-1),IMAGE(I,J,K))

C Next AND with the image shifted down one plane
    DO 60 K = 1,NPLA-1
      DO 60 J = 1,NCOL
        DO 60 I = 0,NROWB-1
          60      EDGES(I,J,K) = IAND(IMAGE(I,J,K+1),EDGES(I,J,K))

C Next the image shifted up one column
    DO 90 K = 1,NPLA
      DO 90 J = 2,NCOL
        DO 90 I = 0,NROWB-1
          90      EDGES(I,J,K) = IAND(EDGES(I,J,K),IMAGE(I,J-1,K))

C Next AND with the image shifted down one column
    DO 120 K = 1,NPLA
      DO 120 J = 1,NCOL-1
        DO 120 I = 0,NROWB-1
          120     EDGES(I,J,K) = IAND(IMAGE(I,J+1,K),EDGES(I,J,K))

C Now AND with the image shifted right one in the row direction.
    DO 150 K = 1,NPLA
      DO 150 J = 1,NCOL
        FLAG = 0
        DO 150 I = 0,NROWB-1
          D = IMAGE(I,J,K)
          SHFLAG = IAND(D,1)
          D = ISHFT(D,-1)
          IF (FLAG.GT.0) D = IOR(D,BIGBIT)
          FLAG = SHFLAG
          150     EDGES(I,J,K) = IAND(D,EDGES(I,J,K))

C Finally AND with the image shifted left one.
    DO 180 K = 1,NPLA
      DO 180 J = 1,NCOL
        FLAG = 0
        DO 180 I = NROWB-1,0,-1
          D = IMAGE(I,J,K)
          SHFLAG = ISHFT(D,-NBITS+1)
          D = ISHFT(D,1)+FLAG
          FLAG = SHFLAG
          180     EDGES(I,J,K) = IAND(D,EDGES(I,J,K))

C Now EOR with the original, leaving an internal 2-connected boundary
    DO 210 K = 1,NPLA
      DO 210 J = 1,NCOL
        DO 210 I = 0,NROWB-1
          210     EDGES(I,J,K) = IEOR(IMAGE(I,J,K),EDGES(I,J,K))

```


C NBR is the pointer to offsets which generate new neighbors.

```
INTEGER NUMBBR,I,J,K,II,JJ,KK,IT,JT,KT,NBR
```

C Arrays X,Y,Z collect the coordinates of the point. The remaining C variables are adequately described in the main program.

```
INTEGER X(32),Y(32),Z(32)
INTEGER NBITS,NROW,NROWB,NCOL,NPLA,EDGES,IMAGE
PARAMETER (NBITS=32,NROW=128,NROWB=NROW/NBITS)
PARAMETER (NCOL=128,NPLA=128)
REAL DISP
INTEGER*2 PLANE
COMMON/GSCALE/ DISP(NCOL,NPLA),PLANE(NCOL,NPLA)
```

C FLAG is set by LSFIT if the linear algebra problem is not solvable.
LOGICAL FLAG

C This is for the VAX: will have to test where to grab least signif 8
C bits on IBM and FPS if this is to be transported. It is a
C non-problem, but annoying anyway.

```
INTEGER INTEQ
LOGICAL*1 STUFF
EQUIVALENCE (INTEQ,STUFF)
COMMON/ARRAY/ EDGES(0:NROWB-1,NCOL,NPLA),
+ IMAGE(0:NROWB-1,NCOL,NPLA)
INTEGER IX,IY,IZ
```

C POINT is a statement function that works like the corresponding
C function in BASIC, except in 3-D

```
LOGICAL POINT
POINT(IX,IY,IZ) = BTEST(EDGES(IX/NBITS,IY,IZ),
+ NBITS-1-MOD(IX,NBITS))
```

C Find out where the bit is in the word. Also make local copies of the
C subroutine parameters.

```
II = I*NBITS
JJ = J
KK = K
10 IF (POINT(II,JJ,KK)) GO TO 20
II = II+1
GO TO 10
```

C Now we're started, having found the bit causing the problem.

```
20 NUMBBR = 1
X(NUMBBR) = II
Y(NUMBBR) = JJ
Z(NUMBBR) = KK
```

C Look around, trying to collect enough. the first 24 neighbors are
C guaranteed to not be off the array because of the border of zeroes.

```
DO 25 NBR = 1,24
IT = II+OFFSX(NBR)
JT = JJ+OFFSY(NBR)
KT = KK+OFFSZ(NBR)
IF (POINT(IT,JT,KT)) THEN
NUMBBR = NUMBBR+1
X(NUMBBR) = IT
Y(NUMBBR) = JT
Z(NUMBBR) = KT
END IF
25 CONTINUE
```

C If we have accumulated 9 or more then we have an adequate sample.

```
IF (NUMBBR.GT.8) GO TO 31
NBR = 25
```

C From now on we have to watch whether off the array since we are looking
C 2 away. No big deal--only 12 critters involved.

```
30 IF (NBR.LE.36) THEN
```

```

IT = II+OFFSX(NBR)
  IF (IT.GT.NROW-2.OR.IT.LT.1) THEN
    NBR = NBR+1
    GO TO 30
  END IF
JT = JJ+OFFSY(NBR)
  IF (JT.GT.NCOL-1.OR.JT.LT.2) THEN
    NBR = NBR+1
    GO TO 30
  END IF
KT = KK+OFFSZ(NBR)
  IF (KT.GT.NPLA-1.OR.KT.LT.2) THEN
    NBR = NBR+1
    GO TO 30
  END IF
  IF (POINT(IT,JT,KT)) THEN
    NUMBBR = NUMBBR+1
    X(NUMBBR) = IT
    Y(NUMBBR) = JT
    Z(NUMBBR) = KT
    NBR = NBR+1
    GO TO 30
  END IF
END IF

```

C Watch out for tiny shapes and other weird stuff.

```

IF (NUMBBR.LT.5) THEN
  DISP(J,K) = -2.0
  PLANE(J,K) = II
  RETURN
END IF

```

C Find the direction cosines of the normal to the best fitting (least squares) tangent plane. Logical variable FLAG is set when the linear algebra problem is deemed impossible.

```

31 CALL LSFIT(X,Y,Z,NUMBBR,CALPHA,CBETA,CGAMMA,FLAG)
  IF (FLAG) THEN
    DISP(J,K) = -12.0
    PLANE(J,K) = II
    RETURN
  END IF

```

C Now calculate the radiation viewed: this is for diffuse lighting and a matte finish surface.

```

VIEW = CALPHA*(1.0+U1*CALPHA+U2*CBETA+U3*CGAMMA)

```

C This is for a glossy finish.

```

IF (SHINFL.GT.1) VIEW = VIEW**SHINFL
DISP(J,K) = VIEW

```

C Remember the plane for filtering to follow.

```

PLANE(J,K) = II
RETURN
END

```

```

SUBROUTINE LSFIT(X,Y,Z,NUMBBR,C1,C2,C3,FLAG)

```

C Find the best fitting tangent plane to the binary data under the assumption that the X-Y-Z coordinates collected are good representatives of the boundary surface. The mathematics of the problem is trivial; the program simply computes parameters needed to solve the normal equations and calls an appropriate program. One minor change: weight the "center" point twice that of any other.

```

IMPLICIT NONE
LOGICAL FLAG
REAL C1,C2,C3,A,B
INTEGER X(*),Y(*),Z(*),NUMBBR,I
REAL MATRIX(4,3)
INTEGER SSY,SSZ,SYX,SYZ,SZX,SX,SY,SZ
SSY = Y(1)**2
SSZ = Z(1)**2

```



```

SYX = Y(1)*X(1)
SYZ = Y(1)*Z(1)
SZX = Z(1)*X(1)
SY = Y(1)
SZ = Z(1)
SX = X(1)
DO 10 I = 1,NUMBBR
  SSY = SSY+Y(I)**2
  SSZ = SSZ+Z(I)**2
  SYX = SYX+Y(I)*X(I)
  SYZ = SYZ+Y(I)*Z(I)
  SZX = SZX+Z(I)*X(I)
  SY = SY+Y(I)
  SZ = SZ+Z(I)
  SX = SX+X(I)

```

10 CONTINUE

C Now float the sums and do the linear algebra.

```

MATRIX(1,1) = SSY
MATRIX(2,1) = SYZ
MATRIX(3,1) = SY
MATRIX(1,2) = SYZ
MATRIX(2,2) = SSZ
MATRIX(3,2) = SZ
MATRIX(1,3) = SY
MATRIX(2,3) = SZ
MATRIX(3,3) = NUMBBR+1
MATRIX(4,1) = SYX
MATRIX(4,2) = SZX
MATRIX(4,3) = SX
CALL SOLVE(MATRIX,A,B,FLAG)

```

C FLAG is set if SOLVE had trouble.

```
IF (FLAG) RETURN
```

C This solves the least squares problem giving $X = AY+BZ+\langle\text{Don't care}\rangle$

```

C1 = SQRT(1./(1.+A**2+B**2))
C2 = -A*C1
C3 = -B*C1
RETURN
END

```

```
SUBROUTINE SOLVE(M,XA,XB,FLAG)
```

C Simple Gaussian elimination...I doubt if you can beat it for a 3X3
C problem, but I'd be glad to talk about it if you think differently.
C for example, one could write out the entire loop 70 and write M as
C a 1-D array, but I won't do that for the 1 or 2 seconds a run it will save.

```

IMPLICIT NONE
LOGICAL FLAG
REAL M(4,3),XA,XB,XC
INTEGER I,J,K
DO 70 K = 1,2
  DO 70 I = K+1,3
    XA = M(K,I)/M(K,K)
    M(K,I) = XA
  DO 70 J = K+1,4
    M(J,I) = M(J,I)-XA*M(J,K)

```

```

70 CONTINUE
FLAG = ABS(M(3,3)).LT.1.E-7.OR.ABS(M(2,2)).LT.1.E-6
IF (FLAG) RETURN
XC = M(4,3)/M(3,3)
XB = (M(4,2)-M(3,2)*XC)/M(2,2)
XA = (M(4,1)-M(2,1)*XB-M(3,1)*XC)/M(1,1)
RETURN
END

```

```

SUBROUTINE FLTR
IMPLICIT NONE

```

C This preliminary filter applies a 5x5 filter inside the raw reflectances
C without trying to induce edge effects. This is an attempt to filter the
C shape, but clearly isn't wanted all the time.

```

INTEGER NBITS, NROW, NROWB, NCOL, NPLA, EDGES, IMAGE
PARAMETER (NBITS=32, NROW=128, NROWB=NROW/NBITS)
PARAMETER (NCOL=128, NPLA=128)
REAL DISP, THREER(NCOL, 5)
LOGICAL*1 LINE(NCOL)
INTEGER*2 CPLANE, PLANE(NCOL, 5), IP
COMMON/SAVMEM/PLANE, THREER
COMMON/GSCALE/ DISP(NCOL, NPLA), CPLANE(NCOL, NPLA)
COMMON/ARRAY/ EDGES(0:NROWB-1, NCOL, NPLA),
+   IMAGE(0:NROWB-1, NCOL, NPLA)
INTEGER I1, I2, I3, I4, I5, IT, I, J, L
REAL W1, W2, W3, W4, W5, SUM, WEIGHT
PARAMETER (W1=0.866, W2=0.5, W3=.738, W4=.389, W5=.09)

C   PATTERN:
C       W5 W4 W2 W4 W5
C       W4 W3 W1 W3 W4
C       W2 W1 @@ W1 W2
C       W5 W4 W2 W4 W5
C       W4 W3 W1 W3 W4

INTEGER*2 P, Q
LOGICAL TEST
TEST(P, Q) = IABS(P-Q).LE.2

C Circular buffer pointers
I1 = 1
I2 = 2
I3 = 3
I4 = 4
I5 = 5

C Read five lines of the image
DO 10 I = 1, 5
  DO 10 J = 1, NCOL
    THREER(J, I) = DISP(J, I)
    PLANE(J, I) = CPLANE(J, I)
10 CONTINUE

C This is the big loop
DO 30 L = 6, NPLA

C Filter the current line. Be sure to test:
C   If the point is in the boundary of the shape
C   If the point is in the same part of the shape

  DO 40 J = 3, NCOL-2
    IF (THREER(J, I3).EQ.-2.0) THEN
      GO TO 35
    ELSE IF (THREER(J, I3).LT.-10.0) THEN

C These points were too hard on the equation solver, but are valid,
C so try to fix them up.

      SUM = 0.
      WEIGHT = 0.
    ELSE

C Here we have a solid citizen point, so count it.

      SUM = THREER(J, I3)
      WEIGHT = 1.
    END IF
    IP = PLANE(J, I3)
    IF (THREER(J-2, I1).GT.0.AND.TEST(PLANE(J-2, I1), IP)) THEN
      SUM = SUM+THREER(J-2, I1)*W5
      WEIGHT = WEIGHT+W5
    END IF
    IF (THREER(J-2, I5).GT.0.AND.TEST(PLANE(J-2, I5), IP)) THEN
      SUM = SUM+THREER(J-2, I5)*W5
      WEIGHT = WEIGHT+W5
    END IF
    IF (THREER(J+2, I1).GT.0.AND.TEST(PLANE(J+2, I1), IP)) THEN
      SUM = SUM+THREER(J+2, I1)*W5
      WEIGHT = WEIGHT+W5
    END IF
    IF (THREER(J+2, I5).GT.0.AND.TEST(PLANE(J+2, I5), IP)) THEN

```

```

        SUM = SUM+THREER(J+2,I5)*W5
        WEIGHT = WEIGHT+W5
    END IF
    IF (THREER(J+1,I1).GT.O.AND.TEST(PLANE(J+1,I1),IP))THEN
        SUM = SUM+THREER(J+1,I1)*W4
        WEIGHT = WEIGHT+W4
    END IF
    IF (THREER(J+1,I5).GT.O.AND.TEST(PLANE(J+1,I5),IP))THEN
        SUM = SUM+THREER(J+1,I5)*W4
        WEIGHT = WEIGHT+W4
    END IF
    IF (THREER(J-1,I1).GT.O.AND.TEST(PLANE(J-1,I1),IP))THEN
        SUM = SUM+THREER(J-1,I1)*W4
        WEIGHT = WEIGHT+W4
    END IF
    IF (THREER(J-2,I2).GT.O.AND.TEST(PLANE(J-2,I2),IP))THEN
        SUM = SUM+THREER(J-2,I2)*W4
        WEIGHT = WEIGHT+W4
    END IF
    IF (THREER(J-2,I4).GT.O.AND.TEST(PLANE(J-2,I4),IP))THEN
        SUM = SUM+THREER(J-2,I4)*W4
        WEIGHT = WEIGHT+W4
    END IF
    IF (THREER(J+2,I2).GT.O.AND.TEST(PLANE(J+2,I2),IP))THEN
        SUM = SUM+THREER(J+2,I2)*W4
        WEIGHT = WEIGHT+W4
    END IF
    IF (THREER(J+2,I4).GT.O.AND.TEST(PLANE(J+2,I4),IP))THEN
        SUM = SUM+THREER(J+2,I4)*W4
        WEIGHT = WEIGHT+W4
    END IF
    IF (THREER(J-1,I1).GT.O.AND.TEST(PLANE(J-1,I1),IP))THEN
        SUM = SUM+THREER(J-1,I1)*W4
        WEIGHT = WEIGHT+W4
    END IF
    IF (THREER(J+1,I1).GT.O.AND.TEST(PLANE(J+1,I1),IP))THEN
        SUM = SUM+THREER(J+1,I1)*W4
        WEIGHT = WEIGHT+W4
    END IF
    IF (THREER(J-1,I5).GT.O.AND.TEST(PLANE(J-1,I5),IP))THEN
        SUM = SUM+THREER(J-1,I5)*W4
        WEIGHT = WEIGHT+W4
    END IF
    IF (THREER(J-2,I3).GT.O.AND.TEST(PLANE(J-2,I3),IP))THEN
        SUM = SUM+THREER(J-2,I3)*W2
        WEIGHT = WEIGHT+W2
    END IF
    IF (THREER(J,I1).GT.O.AND.TEST(PLANE(J,I1),IP))THEN
        SUM = SUM+THREER(J,I1)*W2
        WEIGHT = WEIGHT+W2
    END IF
    IF (THREER(J,I5).GT.O.AND.TEST(PLANE(J,I5),IP))THEN
        SUM = SUM+THREER(J,I5)*W2
        WEIGHT = WEIGHT+W2
    END IF
    IF (THREER(J+2,I3).GT.O.AND.TEST(PLANE(J+2,I3),IP))THEN
        SUM = SUM+THREER(J+2,I3)*W2
        WEIGHT = WEIGHT+W2
    END IF
    IF (THREER(J-1,I2).GT.O.AND.TEST(PLANE(J-1,I2),IP))THEN
        SUM = SUM+THREER(J-1,I2)*W3
        WEIGHT = WEIGHT+W3
    END IF
    IF (THREER(J+1,I2).GT.O.AND.TEST(PLANE(J+1,I2),IP))THEN
        SUM = SUM+THREER(J+1,I2)*W3
        WEIGHT = WEIGHT+W3
    END IF
    IF (THREER(J+1,I4).GT.O.AND.TEST(PLANE(J+1,I4),IP))THEN
        SUM = SUM+THREER(J+1,I4)*W3
        WEIGHT = WEIGHT+W3
    END IF
    IF (THREER(J-1,I4).GT.O.AND.TEST(PLANE(J-1,I4),IP))THEN
        SUM = SUM+THREER(J-1,I4)*W3
        WEIGHT = WEIGHT+W3
    END IF
    IF (THREER(J-1,I3).GT.O.AND.TEST(PLANE(J-1,I3),IP))THEN
        SUM = SUM+THREER(J-1,I3)*W1

```

```

WEIGHT = WEIGHT+W1
  END IF
IF (THREER(J+1,I3).GT.0.AND.TEST(PLANE(J+1,I3),IP))THEN
SUM = SUM+THREER(J+1,I3)*W1
WEIGHT = WEIGHT+W1
  END IF
IF (THREER(J,I2).GT.0.AND.TEST(PLANE(J,I2),IP))THEN
SUM = SUM+THREER(J,I2)*W1
WEIGHT = WEIGHT+W1
  END IF
IF (THREER(J,I4).GT.0.AND.TEST(PLANE(J,I4),IP))THEN
SUM = SUM+THREER(J,I4)*W1
WEIGHT = WEIGHT+W1
  END IF
  DISP(J,L-3) = SUM/WEIGHT
35   CONTINUE
40   CONTINUE

C Read another line
DO 50 J = 1,NCOL
  THREER(J,I1) = DISP(J,L)
  PLANE(J,I1) = CPLANE(J,L)
50   CONTINUE

C Rotate circular buffer
IT = I1
I1 = I2
I2 = I3
I3 = I4
I4 = I5
I5 = IT
30 CONTINUE

RETURN
END
SUBROUTINE SFLTR(SCALE)
IMPLICIT NONE
  INTEGER NBITS,NROW,NROWB,NCOL,NPLA,EDGES,IMAGE
  PARAMETER (NBITS=32,NROW=128,NROWB=NROW/NBITS)
  PARAMETER (NCOL=128,NPLA=128)

C DISP is the grey scale image to make. DISP and CPLANE could be disk
C files with no loss in speed. Everything in the filtering part is arranged
C that way (for sequential files).

REAL DISP,MAX,MIN,THREER(NCOL,3)
LOGICAL*1 LINE(NCOL)
INTEGER*2 CPLANE,PLANE(NCOL,3),IP
COMMON/SAVMEM/PLANE,THREER
COMMON/GSCALE/ DISP(NCOL,NPLA),CPLANE(NCOL,NPLA)

C This is for the VAX: will have to test where to grab least signif 8
C bits on IBM and FPS if this is to be transported.

INTEGER INTEQ,I,J,L,SCALE
LOGICAL*1 STUFF
EQUIVALENCE (INTEQ,STUFF)

COMMON/ARRAY/ EDGES(0:NROWB-1,NCOL,NPLA),
+ IMAGE(0:NROWB-1,NCOL,NPLA)
INTEGER I1,I2,I3,IT
REAL M,B
REAL W1,W2,SUM,WEIGHT
PARAMETER (W1=0.50,W2=0.20)
PARAMETER (WEIGHT=1.0+4*(W1+W2))
INTEGER*2 P,Q
LOGICAL TEST
TEST(P,Q) = IABS(P-Q).LE.1

C First find the MAX and MIN observed after initial filtering.

MAX = -2.0
MIN = 2.0
DO 1 I = 1,NPLA,2
  DO 1 J = I,NCOL,2
M = DISP(I,J)
IF (M.GT.-2.) THEN

```

```

      IF (MAX.LT.M) THEN
MAX = M
GO TO 1
      ELSE IF (MIN.GT.M) THEN
MIN = M
      END IF
END IF
      1 CONTINUE

C Linear conversion factors
M = REAL(SCALE)/(MAX-MIN)
B = -MIN*M

C Circular buffer pointers
I1 = 1
I2 = 2
I3 = 3

C Read three lines of the image
DO 10 I = 1,3
  DO 10 J = 1,NCOL
    IF (DISP(J,I).NE.-2.0) THEN
      IF (DISP(J,I).GT.-10.0) THEN
        THREER(J,I) = M*DISP(J,I)+B
      ELSE
        THREER(J,I) = -12.0
      END IF
    ELSE
      THREER(J,I) = -2.0
    END IF
    PLANE(J,I) = CPLANE(J,I)
  10 CONTINUE

C Copy the first line without filtering (should be all -2.0)
DO 20 J = 1,NCOL
  INTEQ = THREER(J,I1)
  IF (INTEQ.LE.0) THEN
INTEQ = 0
  END IF
  LINE(J) = STUFF
  20 CONTINUE

C Write the eldest line
WRITE(10) LINE

C This is the big loop
DO 30 L = 4,NPLA

C Filter the current line. Be sure to test:
C   If the point is in the boundary of the shape
C   If the point is in the same part of the shape

  DO 40 J = 1,NCOL
    IF (THREER(J,I2).EQ.-2.0) THEN
      INTEQ = 0
      GO TO 35
    ELSE IF (THREER(J,I2).LT.-10.0) THEN

C These points were too hard on the equation solver, but are valid,
C so try to fix them up.

      SUM = 0.
      ELSE

C Here we have a solid citizen point, so count it.

      SUM = THREER(J,I2)
    END IF
    IP = PLANE(J,I2)
    IF (THREER(J,I1).GT.0.AND.TEST(PLANE(J,I1),IP)) THEN
      SUM = SUM+THREER(J,I1)*W1
    END IF
    IF (THREER(J,I3).GT.0.AND.TEST(PLANE(J,I3),IP)) THEN
      SUM = SUM+THREER(J,I3)*W1
    END IF
    IF (THREER(J-1,I2).GT.0.AND.TEST(PLANE(J-1,I2),IP)) THEN
      SUM = SUM+THREER(J-1,I2)*W1

```

```

      END IF
      IF (THREER(J+1,I2).GT.0.AND.TEST(PLANE(J+1,I2),IP))THEN
      SUM = SUM+THREER(J+1,I2)*W1
      END IF
      IF (THREER(J-1,I1).GT.0.AND.TEST(PLANE(J-1,I1),IP))THEN
      SUM = SUM+THREER(J-1,I1)*W2
      END IF
      IF (THREER(J-1,I3).GT.0.AND.TEST(PLANE(J-1,I3),IP))THEN
      SUM = SUM+THREER(J-1,I3)*W2
      END IF
      IF (THREER(J+1,I1).GT.0.AND.TEST(PLANE(J+1,I1),IP))THEN
      SUM = SUM+THREER(J+1,I1)*W2
      END IF
      IF (THREER(J+1,I3).GT.0.AND.TEST(PLANE(J+1,I3),IP))THEN
      SUM = SUM+THREER(J+1,I3)*W2
      END IF
      SUM = SUM/WEIGHT
      INTEQ = SUM
      IF (INTEQ.LT.1) INTEQ = 1
      IF (INTEQ.GT.SCALE) INTEQ = SCALE

      35      LINE(J) = STUFF
      40      CONTINUE
      WRITE(10) LINE

C Read another line
      DO 50 J = 1,NCOL
      IF (DISP(J,L).NE.-2.0) THEN
      THREER(J,I1) = M*DISP(J,L)+B
      ELSE
      THREER(J,I1) = -2.0
      END IF
      PLANE(J,I1) = CPLANE(J,L)
      50      CONTINUE

C Rotate circular buffer
      IT = I1
      I1 = I2
      I2 = I3
      I3 = IT
      30 CONTINUE

C Finally, write the last critter (unfiltered)
      DO 60 J = 1,NCOL
      INTEQ = THREER(J,I3)
      IF (INTEQ.LE.0) THEN
      INTEQ = 0
      ELSE IF (INTEQ.GT.SCALE) THEN
      INTEQ = SCALE
      END IF
      LINE(J) = STUFF
      60 CONTINUE
      RETURN
      END

SUBROUTINE MAKEIT(DTAFLG,INNAME)
      IMPLICIT NONE
      INTEGER IS,I,J,K,IBSET,IX,IY,IZ,L,IC,LL
      INTEGER NBITS,NROW,NROWB,NCOL,NPLA,EDGES,IMAGE
      PARAMETER (NBITS=32,NROW=128,NROWB=NROW/NBITS)
      PARAMETER (NCOL=128,NPLA=128)
      REAL C(300,3),R(300),II,JJ,KK
      INTEGER MIDX,MIDY,MIDZ,IPSET
      CHARACTER*40 INNAME
      LOGICAL DTAFLG
      COMMON/ARRAY/ EDGES(0:NROWB-1,NCOL,NPLA),
      + IMAGE(0:NROWB-1,NCOL,NPLA)
      IPSET(IX,IY,IZ) = IBSET(IMAGE(IX/NBITS,IY,IZ),
      + NBITS-1-MOD(IX,NBITS))
      IF (DTAFLG) THEN
      CALL GETPAR(C,R,IC)
      PRINT*,' READ',IC,' BALLS'
      DO 3 I = 1,IC
      3      R(I) = R(I)**2
      DO 10 J = 2,NCOL-1
      DO 10 K = 2,NPLA-1
      DO 10 I = 1,NROW-2

```

```

        DO 5 L = 1,IC
          JJ = (REAL(J)-C(L,2))**2
        IF (JJ.GE.R(L)) GO TO 5
        KK = (REAL(K)-C(L,3))**2
        IF (JJ+KK.GE.R(L)) GO TO 5
          II = (REAL(I)-C(L,1))**2
        IF (JJ+KK+II.GE.R(L)) GO TO 5
        IMAGE(I/NBITS,J,K) = IPSET(I,J,K)
        GO TO 10
      5   CONTINUE
      10  CONTINUE
        PRINT*, ' MADE IT'
        OPEN(11,FILE=INNAME,STATUS='NEW',FORM='UNFORMATTED')
          CALL WRITEDATA (11,IMAGE)
        PRINT*, ' STASHED IT ON DISK, UNIT 11'
      ELSE
        OPEN(11,FILE=INNAME,STATUS='OLD',FORM='UNFORMATTED')
          CALL READDATA (11,IMAGE)
        PRINT*, ' READ IT FROM DISK, UNIT 11'
          END IF
        RETURN
      END

SUBROUTINE GETPAR(C,R,IC)
  IMPLICIT NONE
  INTEGER IC,I,J
  REAL C(300,3),R(300)
  IC = 1
  DO 10 I = 1,300
    PRINT*, ' CENTER, RADIUS '
    READ*,(C(I,J),J=1,3),R(I)
    IF (C(I,1).EQ.0) RETURN
    IC = IC+1
  10 CONTINUE
  RETURN
  END

SUBROUTINE SHOWME
  C For a VT200-level terminal--quick and dirty look program.
  CHARACTER*32 TC/'\NN@H8%GC]|\!>;,:.'
  CHARACTER*1 P(0:31),POP
  INTEGER I,J,IPOP,MAX
  EQUIVALENCE (POP,IPOP)
  CHARACTER*1 LINE(128),DISP(128)
  LOGICAL IFODD
  IFODD = .TRUE.
  MAX = 0
  REWIND(10)
  DO 10 I = 1,32
    P(I-1) = TC(I:I)
  10 CONTINUE
  1 READ(10,END=9) LINE
  DO 2 J = 1,128
    POP = LINE(J)
    MAX = MAXO(MAX,IPOP)
  2 CONTINUE
  GO TO 1
  9 REWIND(10)
  IF (MAX.EQ.0) STOP ' NO DATA IN FILE'
  20 READ(10,END=99) LINE
  READ(10,END=99) LINE
  DO 30 J = 1,128
    POP = LINE(J)
  IF (IPOP.EQ.0) THEN
    IF (IFODD) THEN
      DISP(J) = '{'
    ELSE
      DISP(J) = '}'
    END IF
  ELSE
    DISP(J) = P(MOD(IPOP*17/MAX,32)+1)
  END IF
  30 CONTINUE
  PRINT40,(DISP(J),J=1,128)
  40 FORMAT(3X,128A1)
  IFODD = .NOT.IFODD
  GO TO 20

```

```

99 RETURN
END

SUBROUTINE READDATA (UNITNO,IMAGE)

C Reads data in its compressed form

IMPLICIT NONE

INTEGER NROWB,NCOL,NBITS
PARAMETER (NROWB = 4)
PARAMETER (NCOL = 128)
PARAMETER (NBITS = 32)

INTEGER UNITNO
INTEGER IMAGE(NROWB,NCOL,NCOL)
INTEGER I,J,K,L
INTEGER COUNT1
INTEGER MAP(NROWB*NCOL*NCOL/NBITS)
INTEGER NONZERO(NROWB*NCOL*NCOL)
INTEGER NONZEROCOUNT
CHARACTER*20 FILENAME

READ (UNITNO) COUNT1

WRITE (*,*) 'READING ZERO MAP...'
READ (UNITNO) (MAP(I),I=1,NCOL*NCOL*NROWB/NBITS)

WRITE (*,*) 'READING NONZERO ENTRIES...'
READ (UNITNO) (NONZERO(I),I=1,COUNT1)

WRITE (*,*) 'REBUILDING THE DATA...'
NONZEROCOUNT = 0
DO 70 L = 1, NCOL*NCOL*NROWB
  IF (BTEST(MAP((L-1)/NBITS+1),MOD((L-1),NBITS))) THEN
    I = (L-1)/(NCOL*NCOL)
    J = (L-I*NCOL*NCOL-1)/NCOL
    K = L-I*NCOL*NCOL-J*NCOL
    I = I + 1
    J = J + 1
    NONZEROCOUNT = NONZEROCOUNT + 1
    IMAGE(I,J,K) = NONZERO(NONZEROCOUNT)
  ENDIF
70 CONTINUE

WRITE (*,*) 'DECOMPRESSION COMPLETE.'

RETURN
END

SUBROUTINE WRITEDATA (UNITNO,IMAGE)

C Writes data in its compressed form

IMPLICIT NONE

INTEGER NROWB,NCOL,NBITS
PARAMETER (NROWB = 4)
PARAMETER (NCOL = 128)
PARAMETER (NBITS = 32)

INTEGER UNITNO
INTEGER IMAGE(0:NROWB-1,NCOL,NCOL)
INTEGER I,J,K
INTEGER COUNT1
INTEGER MAP(NROWB*NCOL*NCOL/NBITS)
INTEGER NONZERO(NROWB*NCOL*NCOL)
CHARACTER*20 FILENAME

COUNT1 = 0

WRITE (*,*) 'COMPUTING ZERO MAP...'
DO 50 I = 0,NROWB-1
  DO 50 J = 1,NCOL
    DO 50 K = 1,NCOL
      IF (IMAGE(I,J,K) .NE. 0) THEN
C A non-zero entry

```



```

C          Count it
          COUNT1 = COUNT1 + 1
C          Set map entry
          MAP(I*NCOL*NCOL/NBITS+(J-1)*NCOL/NBITS
&          +INT((K-1)/32)+1)
&          = IBSET(MAP(I*NCOL*NCOL/NBITS+(J-1)*NCOL/NBITS
&          +INT((K-1)/NBITS)+1),MOD(NCOL*(J-1)+K-1,NBITS))
          NONZERO(COUNT1) = IMAGE(I,J,K)
        ENDIF
50 CONTINUE

        WRITE (*,*) 'WRITING ZERO MAP...'
        WRITE (UNITNO) COUNT1
        WRITE (UNITNO) (MAP(I),I=1,NCOL*NCOL*NROWB/NBITS)
60 CONTINUE

        WRITE (*,*) 'WRITING NONZERO DATA...'
        WRITE (UNITNO) (NONZERO(I),I=1,COUNT1)

        WRITE (*,*) 'COMPRESSION COMPLETE.'

        RETURN
        END

```

```

PROGRAM EXTR

IMPLICIT NONE

INTEGER NBITS,NROW,NROWB,NCOL,NPLA,EDGES,IMAGE,BIGBIT,MOREE
PARAMETER (NBITS=32,NROW=128,NROWB=NROW/NBITS)
PARAMETER (NCOL=128,NPLA=128)
PARAMETER (BIGBIT=-2**(NBITS-2)-2**(NBITS-2))
INTEGER D,FLAG,SHFLAG,I,J,K,ITIME,IDUMMY,L,LIB$INIT_TIMER
COMMON/ARRAY/ EDGES(0:NROWB-1,NCOL,NPLA),
+ IMAGE(0:NROWB-1,NCOL,NPLA),MOREE(0:NROWB-1,NCOL,NPLA)
INTEGER MIDX,MIDY,MIDZ,IX,IY,IZ,NPOINT,LIB$SHOW_TIMER,IT
LOGICAL FLAGE,IPOINT
IPOINT(IX,IY,IZ) = BTEST(IMAGE(IX/NBITS,IY,IZ),
+ NBITS-1-MOD(IX,NBITS))
ITIME = 0
IDUMMY = LIB$INIT_TIMER(ITIME)
OPEN(20,FILE='[SHAPES.DAT]STATS',STATUS='OLD',ERR=99)
5 READ(20,*,END=10)
GO TO 5
99 OPEN(20,FILE='[SHAPES.DAT]STATS',STATUS='NEW')
10 CONTINUE
IDUMMY = LIB$SHOW_TIMER(ITIME)
ITIME = 0
IDUMMY = LIB$INIT_TIMER(ITIME)
PRINT*, ' MAKE (T) OR READ (F) DATA '
READ*,FLAGE
CALL MAKEIT(FLAGE)
PRINT*, ' MADE AN IMAGE '
IDUMMY = LIB$SHOW_TIMER(ITIME)
ITIME = 0
IDUMMY = LIB$INIT_TIMER(ITIME)
C NBITS - Number of bits in an integer
C NROW - Number of rows in the discrete image
C NROWB - Number of integers per row
C NCOL - Number of columns
C NPLA - Number of planes
C IMAGE - The discrete image
C EDGES - The boundary points in IMAGE
C
C Find the edges using logical operations.
C
CALL ZEROIT(EDGES,NROWB*NCOL)

C First the image shifted up one plane. NPOINT is the number of points
C in the shape.
NPOINT = 0
DO 30 K = 2,NPLA-1
DO 30 J = 2,NCOL-1
DO 30 I = 0,NROWB-1
IT = IMAGE(I,J,K)
IF (IT.NE.0) THEN
IF (IT.EQ.-1) THEN
NPOINT = NPOINT+NBITS
ELSE
DO 25 L = 0,NBITS-1
IF (BTEST(IT,L)) NPOINT = NPOINT+1
25 CONTINUE
END IF
EDGES(I,J,K) = IAND(IMAGE(I,J,K-1),IT)
ELSE
EDGES(I,J,K) = 0
END IF
30 CONTINUE
C Next AND with the image shifted down one plane
DO 60 K = 2,NPLA-1
DO 60 J = 1,NCOL
DO 60 I = 0,NROWB-1
60 EDGES(I,J,K) = IAND(IMAGE(I,J,K+1),EDGES(I,J,K))

C Next the image shifted up one
DO 90 K = 2,NPLA-1
DO 90 J = 2,NCOL-1
DO 90 I = 0,NROWB-1
90 EDGES(I,J,K) = IAND(EDGES(I,J,K),IMAGE(I,J-1,K))

C Next AND with the image shifted down one plane

```

```

DO 120 K = 2,NPLA-1
DO 120 J = 2,NCOL-1
DO 120 I = 0,NROWB-1
120      EDGES(I,J,K) = IAND(IMAGE(I,J+1,K),EDGES(I,J,K))

C Now AND with the image shifted right one.
DO 150 K = 2,NPLA-1
DO 150 J = 2,NCOL-1
FLAG = 0
DO 150 I = 0,NROWB-1
D = IMAGE(I,J,K)
SHFLAG = IAND(D,1)
D = ISHFT(D,-1)
IF (FLAG.GT.0) D = IOR(D,BIGBIT)
FLAG = SHFLAG
EDGES(I,J,K) = IAND(D,EDGES(I,J,K))

150      CONTINUE

C Finally AND with the image shifted left one.
DO 180 K = 2,NPLA-1
DO 180 J = 2,NCOL-1
FLAG = 0
DO 180 I = NROWB-1,0,-1
D = IMAGE(I,J,K)
SHFLAG = ISHFT(D,-NBITS+1)
D = ISHFT(D,1)+FLAG
FLAG = SHFLAG
EDGES(I,J,K) = IAND(D,EDGES(I,J,K))

180      CONTINUE

C Now EOR with the original, leaving an internal 2-connected boundary
DO 210 K = 2,NPLA-1
DO 210 J = 2,NCOL-1
DO 210 I = 0,NROWB-1
IT = Ieor(IMAGE(I,J,K),EDGES(I,J,K))
MOREE(I,J,K) = IT
210      EDGES(I,J,K) = IT

C
C Now all the boundaries have been found. Proceed to find,
C collect, and analyze the boundaries collected.
C
PRINT*, ' BOUNDARIES MARKED AND POINTS COUNTED '
IDUMMY = LIB$SHOW_TIMER(ITIME)
DO 240 K = 1,NPLA
DO 240 J = 1,NCOL
DO 240 I = 0,NROWB-1
7      IF (EDGES(I,J,K).NE.0) THEN
ITIME = 0
IDUMMY = LIB$INIT_TIMER(ITIME)
CALL EXTRACT3(I,J,K,NPOINT)
PRINT*, ' GOT ONE '
IDUMMY = LIB$SHOW_TIMER(ITIME)
GO TO 7
END IF
240      CONTINUE
CLOSE(20,DISP='KEEP')
END

SUBROUTINE EXTRACT3(I,J,K,NPOINT)

C Purpose: To search a three dimensional 2-neighbor connected set for
C a list of all members. Of course, the list is in no particular
C order. If you like, you can think of the set as being the boundary
C of a 1-neighbor connected discrete object. The list is not kept
C as an array, but as three separate arrays X(*), Y(*), Z(*).
C Revision: they are packed in one integer now.

C Method: Two pointers are maintained: CURRENT and LAST. CURRENT points
C to the point whose neighbors are being examined. LAST points to
C the last one stored. As long as LAST.GE.CURRENT, more points remain
C which require their 18 neighbors to be checked. On termination,
C LAST points to the last one collected.

C Arguments: I The coordinate of the word containing the packed
C first bit.

C J,K Coordinates of column and plane in the array containing

```

C the word.

C INTR calls: POINT A LOGICAL function which is .TRUE. iff the point
C (I,J,K) is a point in the set (on).

C PRESET Resets (turns off) the point at (I,J,K).

C Special considerations: The actual array is packed by bits into a 3
C dimensional INTEGER array. To find the absolute
C starting point, LOGICAL operations are performed.
C The necessary information is passed to PRESET and
C POINT in COMMON BLOCK /ARRAYS/. It is assumed that
C any necessary bounds checking is performed by the
C external modules. Alternatively, the image can
C be strictly inside the 3-d rectangle.

IMPLICIT NONE

C Search pattern:

C Lower plane Ground plane Upper plane

```
C 19 14 20 6 7 8 23 13 24
C 15 16 17 1 * 2 10 11 12 <--> Storage order
C 22 18 21 3 4 5 26 9 25
```

C Offsets for search of the 2-boundary (with 18 neighbors differing by
C at most 1 in at most two coordinates). XX,YY,ZZ are the offsets to boundary
C points for the least squares part. The first 18 are neighbors; the next
C 8 are also used in the least squares tangent fitting problem.

```
INTEGER OFFSET_X(26),XX(26)
DATA OFFSET_X/-1, 1,-1, 0, 1,-1, 0, 1,
+ 0,-1, 0, 1, 0,
+ 0,-1, 0, 1, 0,
+ -1, 1, 1, -1, -1, 1, 1, -1/
```

```
INTEGER OFFSET_Y(26),YY(26)
DATA OFFSET_Y/ 0, 0, 1, 1, 1,-1,-1,-1,
+ 1, 0, 0, 0,-1,
+ -1, 0, 0, 0, 1,
+ 1, 1,-1,-1, 1, 1,-1,-1/
```

```
INTEGER OFFSET_Z(26),ZZ(26)
DATA OFFSET_Z/ 0, 0, 0, 0, 0, 0, 0, 0, 0,
+ 1, 1, 1, 1, 1,
+ -1,-1,-1,-1,-1,
+ -1,-1,-1,-1, 1, 1, 1, 1/
```

LOGICAL POINT,MOINT

INTEGER CURRENT, LAST, I, J, K, II, JJ, KK, IT, JT, KT, NBR

INTEGER MAXSIZE, STASH(18), IPTR, PTR, MIDX, MIDY, MIDZ

PARAMETER (MAXSIZE = 80000)

INTEGER X(MAXSIZE)

REAL U1, U2, U3, MMU, DOT, DIST, AREA, VMT, SII, TI, SU, SIII, TII

INTEGER NFACT

PARAMETER (NFACT=35)

REAL OLAST, NORMS(O:NFACT), SIIII

REAL SUM, N3, T, MAX, FUDGE(O:NFACT), PPQ(NFACT)

```
DATA FUDGE/4550.0,16120.0,16120.0,16120.0,16120.0,
+ 16120.0,16120.0,16120.0,16120.0,16120.0,16120.0,
+ 16120.0,16120.0,16120.0,16120.0,16120.0,16120.0,
+ 16120.0,16120.0,16120.0,16120.0,15*16120.0/
```

```
DATA PPQ/1.,2.,3.,4.,2.,3.,4.,3.,4.,4.,1.,2.,3.,4.,1.,1.,
+ 2.,3.,2.,1./
```

REAL ALPHA, BETA, GAMMA, MU

COMMON/TANGNT/ALPHA(MAXSIZE), BETA(MAXSIZE), GAMMA(MAXSIZE),
+ MU(MAXSIZE)

INTEGER NBITS, NROW, NROWB, NCOL, NPLA, EDGES, IMAGE, MOREE

PARAMETER (NBITS=32, NROW=128, NROWB=NROW/NBITS)

PARAMETER (NCOL=128, NPLA=128)

COMMON/ARRAY/ EDGES(O:NROWB-1, NCOL, NPLA),

+ IMAGE(O:NROWB-1, NCOL, NPLA), MOREE(O:NROWB-1, NCOL, NPLA)

INTEGER IX, IY, IZ, PRESET, PACK, UNPX, UNPY, UNPZ, MRESET

INTEGER BITS13, BITS03, MASKO, NPOINT, NAST, L

PARAMETER (BITS03=NBITS/3, BITS13=BITS03*2)

PARAMETER (MASKO=2**BITS03-1)

```

INTEGER XBAR,YBAR,ZBAR,IACTOR,IUM,ISQUA
REAL FACTOR,RI,SQUA,SI,SUMM,SUMMS,S128,R128
C Statement functions:
POINT(IX,IY,IZ) = BTEST(EDGES(IX/NBITS,IY,IZ),
+ NBITS-1-MOD(IX,NBITS))
MOINT(IX,IY,IZ) = BTEST(MOREE(IX/NBITS,IY,IZ),
+ NBITS-1-MOD(IX,NBITS))
PRESET(IX,IY,IZ) = IBCLR(EDGES(IX/NBITS,IY,IZ),
+ NBITS-1-MOD(IX,NBITS))
PACK(IX,IY,IZ) = IOR(ISHFT(IY,BITS03),ISHFT(IZ,BITS13))+IX
UNPX(IX) = IAND(IX,MASK0)
UNPY(IX) = UNPX(ISHFT(IX,-BITS03))
UNPZ(IX) = ISHFT(IX,-BITS13)
DIST(IX,IY,IZ) = SQRT(REAL(IX**2+IY**2+IZ**2))
C Pointer to the point currently being examined:
CURRENT = 1
C Pointer to the last new point found and stored:
LAST = 1
C Find out where the bit is in the word. Also make local copies of the
C subroutine parameters.
II = I*NBITS
JJ = J
KK = K
10 IF (POINT(II,JJ,KK)) GO TO 20
II = II+1
GO TO 10
20 EDGES(II/NBITS,JJ,KK) = PRESET(II,JJ,KK)
X(LAST) = PACK(II,JJ,KK)
XBAR = II
YBAR = JJ
ZBAR = KK
30 IF (LAST.GE.CURRENT) THEN
WAST = 0
C Look for a neighbor that is 'on'. When it is, store it and turn it off.
DO 40 NBR = 1,18
IT = II+OFFSET_X(NBR)
JT = JJ+OFFSET_Y(NBR)
KT = KK+OFFSET_Z(NBR)
IF (POINT(IT,JT,KT)) THEN
LAST = LAST+1
IF (LAST.GT.MAXSIZE) STOP ' TOO BIG'
X(LAST) = PACK(IT,JT,KT)
EDGES(IT/NBITS,JT,KT) = PRESET(IT,JT,KT)
XBAR = XBAR+IT
YBAR = YBAR+JT
ZBAR = ZBAR+KT
END IF
40 CONTINUE
DO 45 NBR = 1,26
IT = II+OFFSET_X(NBR)
JT = JJ+OFFSET_Y(NBR)
KT = KK+OFFSET_Z(NBR)
IF (MOINT(IT,JT,KT)) THEN
WAST = WAST+1
XX(WAST) = OFFSET_X(NBR)
YY(WAST) = OFFSET_Y(NBR)
ZZ(WAST) = OFFSET_Z(NBR)
END IF
45 CONTINUE
CALL LSFIT(XX,YY,ZZ,WAST,CURRENT)
CURRENT = CURRENT+1
II = UNPX(X(CURRENT))
JJ = UNPY(X(CURRENT))
KK = UNPZ(X(CURRENT))
GO TO 30
END IF
XBAR = XBAR/LAST
YBAR = YBAR/LAST
ZBAR = ZBAR/LAST
C The estimate of the volume should be modified to account for boundary not
C all there
NPOINT = NPOINT-LAST/2
VMT = FLOAT(NPOINT)**(-1./3.)
C Compute the area. This should be done in floating point for giant shapes.
AREA = 0.
C Now three surface integrals.
FACTOR = 0.

```

```

SUM = 0.
SQUA = 0.
MAX = 0.
DO 5210 L = 1, NFACT
  NORMS(L) = 0.0
5210 CONTINUE
DO 520 L = 1, LAST
  IX = UNPX(X(L)) - XBAR
  IY = UNPY(X(L)) - YBAR
  IZ = UNPZ(X(L)) - ZBAR
  SUMM = DIST(IX, IY, IZ)
  S128 = SUMM/128
  AREA = AREA + MU(L)
  RI = ABS(IX * ALPHA(L) +
    + IY * BETA(L) +
    + IZ * GAMMA(L))
  R128 = RI/128
  SI = RI * MU(L)
  NORMS(1) = NORMS(1) + SI
  NORMS(21) = NORMS(21) + EXP(S128) * MU(L)
  NORMS(22) = NORMS(22) + EXP(R128) * MU(L)
  NORMS(23) = NORMS(23) + EXP(S128**2 * R128**2) * MU(L)
  NORMS(24) = NORMS(24) + EXP(S128**2) * MU(L)
  NORMS(25) = NORMS(25) + EXP(R128**2) * MU(L)
  NORMS(31) = NORMS(31) + EXP(-S128) * MU(L)
  NORMS(32) = NORMS(32) + EXP(-R128) * MU(L)
  NORMS(33) = NORMS(33) + EXP(-S128**2 * R128**2) * MU(L)
  NORMS(34) = NORMS(34) + EXP(-S128**2) * MU(L)
  NORMS(35) = NORMS(35) + EXP(-R128**2) * MU(L)
  SII = SI * RI
  NORMS(2) = NORMS(2) + SII
  SIII = SII * RI
  NORMS(3) = NORMS(3) + SIII
  SIIII = SIII * RI
  NORMS(4) = NORMS(4) + SIIII
  IF (SUMM.NE.0.) THEN
    NORMS(26) = NORMS(26) + LOG(SUMM) * MU(L)
    NORMS(28) = NORMS(28) + LOG(2 * (RI + SUMM)) * MU(L)
    NORMS(29) = NORMS(29) + LOG(2 * SUMM) * MU(L)
    IF (RI.GT.1.) THEN
      NORMS(27) = NORMS(27) + LOG(RI) * MU(L)
      NORMS(30) = NORMS(30) + LOG(2 * RI) * MU(L)
    END IF
    SUMMS = SUMM**2
    TI = SI * SUMM
    NORMS(5) = NORMS(5) + TI
    TII = TI * SUMM
    NORMS(6) = NORMS(6) + TII
    NORMS(7) = NORMS(7) + TII * SUMM
    NORMS(8) = NORMS(8) + SUMM * SII
    NORMS(9) = NORMS(9) + SUMM * SIII
    NORMS(10) = NORMS(10) + SII * SUMMS
    SU = SUMM * MU(L)
    NORMS(11) = NORMS(11) + SU
    SU = SU * SUMM
    NORMS(12) = NORMS(12) + SU
    SU = SU * SUMM
    NORMS(13) = NORMS(13) + SU
    NORMS(14) = NORMS(14) + SU * SUMM
  NORMS(15) = NORMS(15) + SII / SUMM
  NORMS(16) = NORMS(16) + SIII / SUMMS
  NORMS(17) = NORMS(17) + SIIII / SUMM
  NORMS(18) = NORMS(18) + SIIII / SUMM
  NORMS(19) = NORMS(19) + SIIII / SUMMS
  NORMS(20) = NORMS(20) + SIIII / SUMMS / SUMM
  END IF
520 CONTINUE
NORMS(31) = -LOG(NORMS(31)/AREA) * 128.
NORMS(32) = -LOG(NORMS(32)/AREA) * 128.
NORMS(33) = (-LOG(NORMS(33)/AREA)) ** (1/4.) * 128.
NORMS(34) = (-LOG(NORMS(34)/AREA)) ** .5 * 128.
NORMS(35) = (-LOG(NORMS(35)/AREA)) ** .5 * 128.
DO 5220 L = 1, NFACT - 15
  NORMS(L) = FUDGE(L) * (NORMS(L)/AREA) ** (1./PPQ(L)) * VMT
5220 CONTINUE
  NORMS(21) = LOG(NORMS(21)/AREA) * 128
NORMS(22) = LOG(NORMS(22)/AREA) * 128

```

```

NORMS(23) = (LOG(NORMS(23)/AREA)**(1/4.))*128
NORMS(24) = (LOG(NORMS(24)/AREA)**.5)*128
NORMS(25) = (LOG(NORMS(25)/AREA)**.5)*128
      NORMS(26) = NORMS(26)/AREA
NORMS(27) = NORMS(27)/AREA
NORMS(28) = NORMS(28)/AREA
NORMS(29) = NORMS(29)/AREA
NORMS(30) = NORMS(30)/AREA
DO 5222 L = NFACT-9,NFACT-5
      NORMS(L) = EXP(NORMS(L))
5222 END DO
NORMS(28) = NORMS(28)/4
NORMS(29) = NORMS(29)/2
NORMS(30) = NORMS(30)/2
DO 5221 L = NFACT-14,NFACT
      NORMS(L) = FUDGE(L)*NORMS(L)*VMT
5221 END DO
SU = FUDGE(0)*SQRT(AREA)*VMT
NORMS(0) = SU
PRINT*,NORMS
      WRITE(20,*) ,(IFIX(NORMS(L)),L=0,NFACT)
PRINT*, ' COUNT',LAST, ' SUM MU_M',AREA
RETURN
END

SUBROUTINE COPY(A,B,N)
INTEGER A(N),B(N),N,I
DO 10 I = 1,N
      10 B(I) = A(I)
RETURN
END

      SUBROUTINE ZEROIT(A,N)
IMPLICIT NONE
      INTEGER N,A(N),I
      DO 10 I = 1,N
            A(I) = 0
10      CONTINUE
      RETURN
      END

SUBROUTINE MAKEIT(FLAG)
IMPLICIT NONE
LOGICAL FLAG
INTEGER IS,I,J,K
CHARACTER*40 FILENAME
      INTEGER NBITS,NROW,NROWB,NCOL,NPLA,EDGES,IMAGE,MOREE
      PARAMETER (NBITS=32,NROW=128,NROWB=NROW/NBITS)
      PARAMETER (NCOL=128,NPLA=128)
COMMON/ARRAY/ EDGES(0:NROWB-1,NCOL,NPLA),
+ IMAGE(0:NROWB-1,NCOL,NPLA),MOREE(0:NROWB-1,NCOL,NPLA)
C INTEGER IIS(NROWB)
IF (FLAG) THEN
C DATA IIS/'7FFFFFFFF'X,'FFFFFFFF'X/
DO 10 I = 1,NROWB
      DO 10 J = 2,NCOL-1
DO 10 K = 2,NPLA-1
      IF (IABS(J-NCOL/2).LE.1.AND.K.LE.NPLA/2) GOTO 10
      IF (IABS(K-NPLA/3).LE.1) GOTO 10
      IF ((J-NCOL/2)**2+(K-NPLA/2)**2.GE.NCOL**2/4) GOTO 10
10      CONTINUE
ELSE
      PRINT*, ' FILENAME CONTAINING DATA'
      READ1,FILENAME
      WRITE(20,11) FILENAME
      WRITE(6,11) FILENAME
1      FORMAT(A)
11      FORMAT(5X,A)
      OPEN(11,FILE=FILENAME,STATUS='OLD',FORM='UNFORMATTED')
      CALL READDATA(11,IMAGE)
END IF
PRINT*, ' DATA SET UP'
RETURN
END
SUBROUTINE LSFIT(XX,YY,ZZ,N,LAST)
IMPLICIT NONE
INTEGER XX(*),YY(*),ZZ(*),N,LAST

```

```

INTEGER MAXSIZE
PARAMETER (MAXSIZE = 80000)
INTEGER X(MAXSIZE)
REAL ALPHA, BETA, GAMMA, MU
COMMON/TANGWT/ALPHA(MAXSIZE), BETA(MAXSIZE), GAMMA(MAXSIZE),
+ MU(MAXSIZE)
INTEGER A, B, C, D, E, F, I, CA, CB, CC, IFF
REAL C1, C2, C3, AALPHA, BBETA, GGAMMA, RMMU
SAVE IFF
A = 0
B = 0
C = 0
D = 0
E = 0
F = 0
DO 10 I = 1, N
  A = A+XX(I)**2
  B = B+YY(I)**2
  C = C+ZZ(I)**2
  D = D+XX(I)*YY(I)
  E = E+XX(I)*ZZ(I)
  F = F+YY(I)*ZZ(I)
10 CONTINUE
CA = B*C-F**2
CB = A*C-E**2
CC = A*B-D**2
  IF (CA.GT.CB) THEN
    IF (CA.GT.CC) GO TO 20
  ELSE
    IF (CB.GT.CC) GO TO 40
  END IF

C Here CC is the largest...case 4.C of the paper---check for 0 first
IF (CC.EQ.0) GO TO 100
C3 = 1./FLOAT(CC)
C1 = C3*(E*B-F*D)
C2 = C3*(A*F-D*E)
RMMU = SQRT(1.+C1**2+C2**2)
GGAMMA = -1./RMMU
AALPHA = -C1*GGAMMA
BBETA = -C2*GGAMMA
GO TO 50

C Here CA is biggest...case 4.a of the paper (it can't be 0)
20 CONTINUE
  C1 = 1./FLOAT(CA)
C2 = C1*(D*C-F*E)
C3 = C1*(B*E-D*F)
RMMU = SQRT(1.+C2**2+C3**2)
AALPHA = -1./RMMU
BBETA = -C2*AALPHA
GGAMMA = -C3*AALPHA
GO TO 50

C Here CB is biggest...case 4.b (it can't be 0)
40 CONTINUE
C2 = 1./FLOAT(CB)
C1 = C2*(D*C-E*F)
C3 = C2*(A*F-E*D)
RMMU = SQRT(1.+C1**2+C3**2)
BBETA = -1./RMMU
AALPHA = -C1*BBETA
GGAMMA = -C3*BBETA
50 CONTINUE
IF (RMMU.GT.2.) THEN
  IFF = IFF+1
  PRINT*, ' FUNNY ONE', RMMU, IFF
  RMMU = 2.
END IF
ALPHA(LAST) = AALPHA
BETA(LAST) = BBETA
GAMMA(LAST) = GGAMMA
MU(LAST) = RMMU
GO TO 200

C This is the impossible case of course!
100 MU(LAST) = 0.
PRINT*, ' ZERO', N, LAST, CA, CB, CC
200 RETURN
END

```



```
SUBROUTINE READDATA (UNITNO,IMAGE)
```

C Reads the data in its new compressed form

```
IMPLICIT NONE
```

```
INTEGER NROWB,NCOL,NBITS  
PARAMETER (NROWB = 4)  
PARAMETER (NCOL = 128)  
PARAMETER (NBITS = 32)
```

```
INTEGER UNITNO  
INTEGER IMAGE(NROWB,NCOL,NCOL)  
INTEGER I,J,K,L  
INTEGER COUNT1  
INTEGER MAP(NROWB*NCOL*NCOL/NBITS)  
INTEGER NONZERO(NROWB*NCOL*NCOL)  
INTEGER NONZEROCOUNT  
CHARACTER*20 FILENAME
```

```
READ (UNITNO) COUNT1
```

```
WRITE (*,*) 'READING ZERO MAP...'  
READ (UNITNO) (MAP(I),I=1,NCOL*NCOL*NROWB/NBITS)
```

```
WRITE (*,*) 'READING NONZERO ENTRIES...'  
READ (UNITNO) (NONZERO(I),I=1,COUNT1)
```

```
WRITE (*,*) 'REBUILDING THE DATA...'  
NONZEROCOUNT = 0  
DO 70 L = 1, NCOL*NCOL*NROWB  
  IF (BTEST(MAP((L-1)/NBITS+1),MOD((L-1),NBITS))) THEN  
    I = (L-1)/(NCOL*NCOL)  
    J = (L-I*NCOL*NCOL-1)/NCOL  
    K = L-I*NCOL*NCOL-J*NCOL  
    I = I + 1  
    J = J + 1  
    NONZEROCOUNT = NONZEROCOUNT + 1  
    IMAGE(I,J,K) = NONZERO(NONZEROCOUNT)
```

70 CONTINUE

```
WRITE (*,*) 'DECOMPRESSION COMPLETE.'
```

```
RETURN  
END
```

```
SUBROUTINE WRITEDATA (UNITNO,IMAGE)
```

C Writes the data in its new compressed form

```
IMPLICIT NONE
```

```
INTEGER NROWB,NCOL,NBITS  
PARAMETER (NROWB = 4)  
PARAMETER (NCOL = 128)  
PARAMETER (NBITS = 32)
```

```
INTEGER UNITNO  
INTEGER IMAGE(0:NROWB-1,NCOL,NCOL)  
INTEGER I,J,K  
INTEGER COUNT1  
INTEGER MAP(NROWB*NCOL*NCOL/NBITS)  
INTEGER NONZERO(NROWB*NCOL*NCOL)  
CHARACTER*20 FILENAME
```

```
COUNT1 = 0
```

```
WRITE (*,*) 'COMPUTING ZERO MAP...'  
DO 50 I = 0,NROWB-1  
  DO 50 J = 1,NCOL  
    DO 50 K = 1,NCOL  
      IF (IMAGE(I,J,K) .NE. 0) THEN  
        A non-zero entry  
        Count it  
        COUNT1 = COUNT1 + 1
```

C
C

```

C          Set map entry
          MAP(I*NCOL*NCOL/NBITS+(J-1)*NCOL/NBITS
&          +INT((K-1)/32)+1)
&          = IBSET(MAP(I*NCOL*NCOL/NBITS+(J-1)*NCOL/NBITS
&          +INT((K-1)/NBITS)+1),MOD(NCOL*(J-1)+K-1,NBITS))
          NONZERO(COUNT1) = IMAGE(I,J,K)
        ENDIF
50 CONTINUE

        WRITE (*,*) 'WRITING ZERO MAP...'
        WRITE (UNITNO) COUNT1
        WRITE (UNITNO)(MAP(I),I=1,NCOL*NCOL*NROWB/NBITS)
60 CONTINUE

        WRITE (*,*) 'WRITING NONZERO DATA...'
        WRITE (UNITNO) (NONZERO(I),I=1,COUNT1)

        WRITE (*,*) 'COMPRESSION COMPLETE.'

        RETURN
        END

```

PROGRAM MAKES

```

INTEGER NHDR ! Number of header records if not IIS files
INTEGER NS ! Number of samples
INTEGER NL ! Number of lines
INTEGER NB ! Number of bands
INTEGER I ! DO loop index
CHARACTER*40 IMAGNAME(48) ! Allow 48 bands if type 1
CHARACTER*40 OUTNAME ! Output VMS file name for cluster map
CHARACTER*16 IISNAME ! IIS name of cluster map
CHARACTER*40 OUT3BND ! Output VMS file name for 3 band output
CHARACTER*16 IIS3BND ! IIS name of 3 band image
INTEGER SWITCH ! Used to tell whether to reduce
INTEGER IV,IR ! Band index of VIS & IR data
LOGICAL MASK ! if T, data in band 1 =0 means mask of image
INTEGER CLASMETH ! classification method
INTEGER WSRCDC ! Error type I weight
INTEGER WDRCS ! Error type II weight
INTEGER WITER ! Number of iterations
INTEGER MAXCLUS ! Maximum number of clusters
INTEGER NALIKE ! How many must match if AMOEBA classification
INTEGER PCTWFLD ! Percent estimated to be in fields
BYTE BLINE(512),BUFFER(128)
INTEGER*2 LLINE(512)
PRINT*, ' OUTPUT FILE NAME?'
  READ1,OUTNAME
  1 FORMAT(A)
DO I = 1,16
  4 PRINT*, ' INPUT FILE NAME',I
  READ1,IMAGNAME(I)
  OPEN(I+10,FILE=IMAGNAME(I),ERR=2,STATUS='OLD')
  CLOSE(I+10)
  GO TO 3
  2 PRINT*, ' SOMETHING WRONG!'
  GO TO 4
  3 END DO
NS = 512
NL = 512
DO 10 K = 1,512
10 LLINE(K) = 0
LLINE(1) = NS ! Number samples request
LLINE(2) = NL ! Number lines request
LLINE(3) = 1 ! Number of bands request
LLINE(4) = 1 ! SS
LLINE(5) = 1 ! SL
LLINE(6) = NS ! NS
LLINE(7) = NL ! NL
LLINE(33) = NS ! Total NS
LLINE(34) = NL ! NL
LLINE(35) = 1 ! NB
LLINE(8) = 1 ! SINC
LLINE(9) = 1 ! LINC
LLINE(16) = 1 ! Band list length
LLINE(17) = 1 ! Band list
LLINE(80) = 512 ! Block size
LLINE(74) = 1 ! Data type
LLINE(77) = 1 ! Open flag (=OPEN)
LLINE(78) = 1 ! I/O Flag (=INPUT)
LLINE(84) = NL*((NS-1)/512+1) ! Total number of records
LLINE(83) = (NS-1)/512+1 ! Number of records per line
LLINE(81) = 511/NS+1 ! Number of header records
LLINE(85) = NS
IF (NS.GT.512) LLINE(85) = 512 ! Samples per record
LLINE(86) = NS
IF (NS.GT.512) LLINE(86) = MOD(NS,512)
IF (LLINE(86).EQ.0) LLINE(86) = 512
BLINE(175) = '0'
BLINE(176) = 'K'
NSIZE = MINO(NS,512)
OPEN(UNIT=99,FILE=OUTNAME,STATUS='NEW',BLOCKSIZE=512,
+ RECL=NSIZE,RECORDTYPE='FIXED',FORM='FORMATTED')
DO I = 1,16
BLINE(112+I) = ICHAR(OUTNAME(I:I))
END DO
NH = 0
DO I = 1,512,NSIZE
JJ = MINO(512,I+NSIZE-1)

```

```

NH = NH+1
WRITE(99,199)(BLINE(J),J=I,JJ)
199 FORMAT(512A1)
END DO
IF (NH.GT.1) THEN
TYPE*,NH,' header records written.'
ELSE
TYPE*,'      One header record written.'
END IF
C
C----- Now take care of all the others -----
C
DO I = 10+1,10+16
  OPEN(I,STATUS='OLD',FORM='UNFORMATTED',
    +      FILE=IMAGNAME(I-10),READONLY)
END DO
DO I = 0,3
  DO J = 1,127
    DO L = 0,3
      READ(11+4*I+L,END=6) BUFFER
      DO JJ = L*128+1,L*128+128
        BLINE(JJ) = BUFFER(JJ-128*L)
      END DO
    END DO
      WRITE(99,199)(BLINE(JJ),JJ=1,512)
  END DO
  6 DO M = J,128
    WRITE(99,199)(BLINE(JJ),JJ=1,512)
  END DO
END DO
STOP '--- Normal Termination ---'
END

```