SLAC-192
UC-32

INTERACTIVE 3D MOTION GRAPHICS WITH LARGE DATA BASES*

STEPHEN ROBERT LEVINE
STANFORD LINEAR ACCELERATOR CENTER
STANFORD UNIVERTISY
Stanford, California 94305

PREPARED FOR THE ENERGY RESEARCH AND DEVELOPMENT ADMINISTRATION
UNDER CONTRACT NO. E(04-3)-515

Manuscript Completed June 1975

176P

Printed in the United States of America. Available from National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161. Price: Printed Copy $7.50; Microfiche $2.25.

*Ph.D. dissertation.

K- computer Group
Sci- misc.

ABSTRACT

A graphic data organization was designed that allows for fast access to an arbitrarily large data base. The graphic data is organized by clustering into successively larger spheres. The derived tree structure provides for multilevel descriptions of objects that allow for display of only the amount of detail that can be resolved on the display, in addition to rapid determination of data in the field of view.

An algorithm was developed to search for the minimum volume sphere enclosing a set of points in three dimensions. A heuristic was used to speed the search for the minimum volume sphere gaining several orders of magnitude improvement over a non-heuristic search.

In support of the minimum volume sphere heuristic, an algorithm to calculate the convex hull of a set of points in more than two dimensions was constructed. For the three-dimensional data sets tested, an order of magnitude improvement over an existing algorithm was found.

To facilitate display of a simulation of a user moving through space containing a large number of objects, a fast hidden-line algorithm was developed. It allows for display of linearly separable three-dimensional convex solids, two-dimensional concave planes, and three-dimensional wire frame objects. A complex picture consisting of 16 objects (192 edges) requires only 130 milliseconds on a third generation large scale computer.

Extensive tests of the algorithms were made using a variety of data sets. The results are summarized in a series of graphical presentations.

ACKNOWLEDGMENTS

The author gratefully acknowledges the careful guidance, direction, and utmost patience of Professor William F. Miller, without whose insight into the overall scope of the research, this thesis would not have been possible.

Grateful acknowledgment is also made to Mary Anne Fisherkeller, Robert Beach, Charles Zahn, John Tukey, David Thompson and Forest Baskett for many helpful discussions.

In addition, much appreciation goes to the Stanford Linear Accelerator Center Computation Research Group and the Lawrence Livermore Laboratory Computation Department for the many hours of computer time they provided.

Finally, the author is indebted to Mrs. Harriet Canfield for here superb preparation of this manuscript, and to my wife, Janet, for her sacrifice and support during this work.

TABLE OF CONTENTS

CHAPTER I

A Brief Overview

This thesis is concerned with the interactive motion display of three-dimensional objects on a two-dimensional screen. The system described is one in which the user can "fly" smoothly through a three-dimensional space filled with a large number of objects. The view presented to the user, on the display, is a perspective projection of the data from the current position in space.

Perspective display of 3D data is, of course, not a new idea. The governing mathematics have been understood for hundreds of years. With a line drawing display such as the IDIIOM,[4] the display of the data involves only simple calculations.

If the time required to generate a picture were not a consideration, then our goal could be reached by using well-known techniques. For each position along the user path for which a display is required, each of the objects in the data base must be tested to determine if they are either in or intersected by the pyramid of vision. This is the first of the problem areas we encounter.

If the data base is very large, this initial testing becomes the most time consuming calculation, particularly when the data set resides partially on disk. It becomes clear that some structuring of the data base is required to accelerate this process. The structuring becomes important when the data "visible" to the user is only a fraction of the data base. We need to minimize the number of tests made on data outside the pyramid of vision. We shall refer to this process as the data retrieval problem and the required structuring as the data organization.

A great deal of work has been done on "graphic data structures".[11] These data structures are designed to facilitate picture construction, manipulation and recognition. We found no reported work dealing with a data structure designed to facilitate graphic data retrieval. In Chapter II, we report on a new type of structuring designed to solve this problem. It encloses the objects into nested spheres. An algorithm to generate the nested spheres is presented in Chapters IV and V.

Another problem area is in the choice of a hidden line algorithm. The fastest reported times are not adequate for our needs. Chapter VI presents a new algorithm that gives the fastest times for the type of objects in our data base.

CHAPTER II

Data Organization

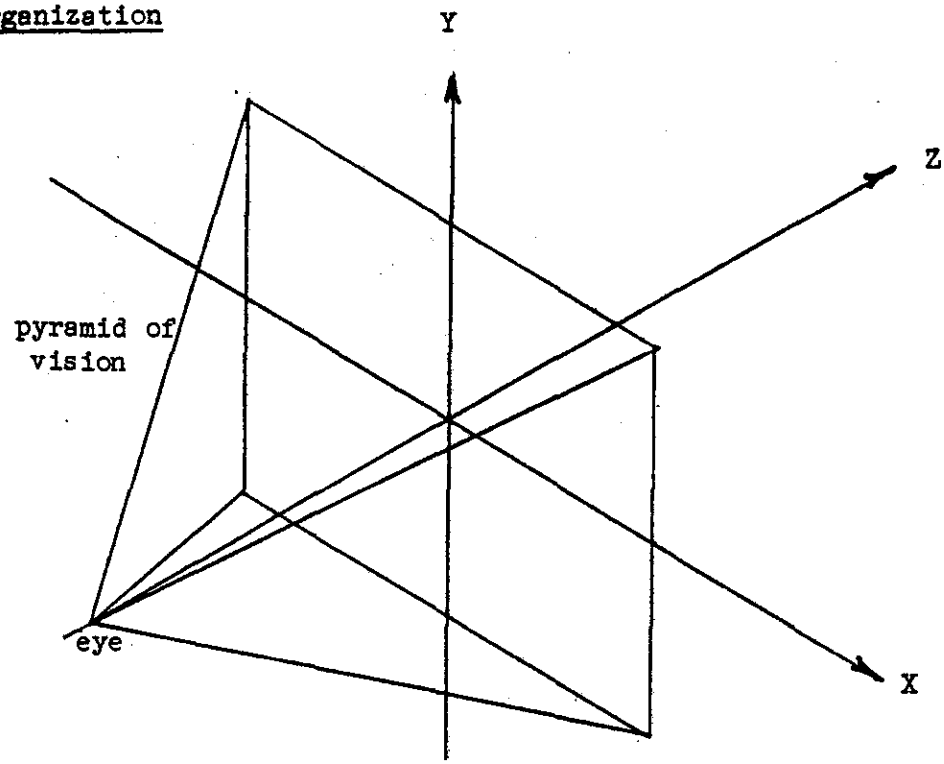


FIGURE 2.1

The objects in the data base are defined on a three-dimensional Cartesian coordinate system with center (0,0,0) and no limit on X, Y, or Z, as shown in Figure 2.1. There is no restriction on size or location of the objects. Further, the user can position him(her)self at any point in space and view in any direction. The only restriction is embodied in the addressability of the display itself. Figure 2.1 shows the special case where the viewing position is on the Z-axis.

All digital displays have finite addressability that typically range from 512 x 512 to 16384 x 16384 raster units. The IDIIOM display we use has an addressability of 1024 x 1024 raster units. The actual resolution is somewhat less in that a square drawn, connecting four adjacent addressable positions, appears as a single bright point.

It is because of this finite addressability that certain simplifications can be made. We recognize that objects far away from the viewing position (as well as small objects) will appear small in the plane of projection. This is due to the fact that in perspective projection the X and Y dimensions are inversely proportional to the distance from the eye to the object. For example, if it can be determined that a complex object in the field of view will occupy an area of one square raster unit, then it is a waste of time to both retrieve this object, as well as perform the time consuming hidden line calculations.

A. Design Goals

The data organization has two design goals. First, it must provide a fast test that can give an upper bound on the projected size of the object. Secondly, a fast test should be available to eliminate large amounts of data outside the pyramid of vision without resorting to examination of each object. The solution to the first problem requires some information to be stored with each object which can be used as a measure of its size. The second problem indicates that some sort of partitioning of the space is required. At this point, we will discuss several approaches used in two dimensions and show how they can be extended to a three-dimensional data base.

B. Two-Dimensional Data Selection

The problems encountered in the display of two-dimensional data are essentially the same as for the three-dimensional case. Rather than fly through a three-dimensional world, we move our "window" (as opposed to a pyramid of vision) across the two-dimensional field. By changing the size of this window, we can, in effect, zoom closer for a more detailed view or move back for a broader view. When we are close in, most

of the data will be outside of the window. We need a scheme for partitioning the space to quickly eliminate large portions of the data. Conversely, when our window is large, relative to the entire two-dimensional field, much of the data will fall below resolvability and should not be retrieved for display.

The first approach we consider is to divide the region into successively smaller regions in a manner similar to that used by Warnock.[10] The entire two-dimensional field is extended (if necessary) to form a square. This square is divided into four equal squares. Each of the squares containing any data can be subdivided further (see Figure 2.2). The large squares are numbered as shown. The four smaller squares within are numbered in a similar manner. For example, the square containing Object B would be 3.1 while the square containing Objects E would be labeled 3.3.3.1.

Data is assigned to the smallest square(s) region(s) that contain it. Thus, Object B belongs to 3.1 while Object D belongs to 3.1, 3.2.3, 3.4, 3.3.2. Regions are subdivided only when necessary.

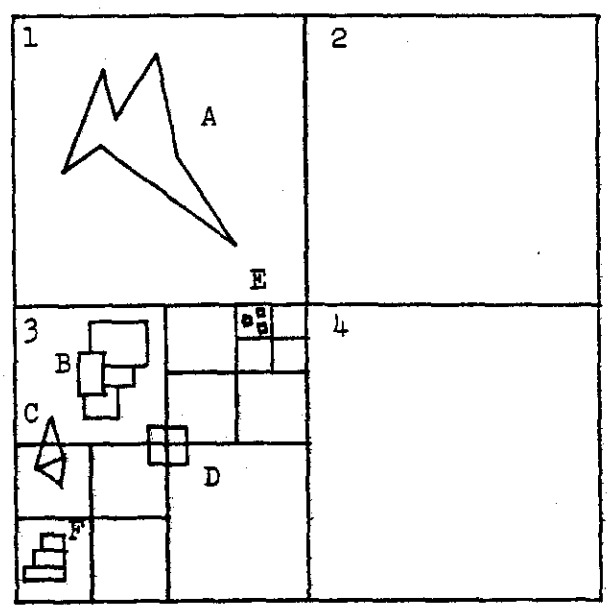


FIGURE 2.2

This subdivision can be implemented as a tree with four-way branching at each node. A tree for the example in Figure 2.2 is shown in Figure 2.3.

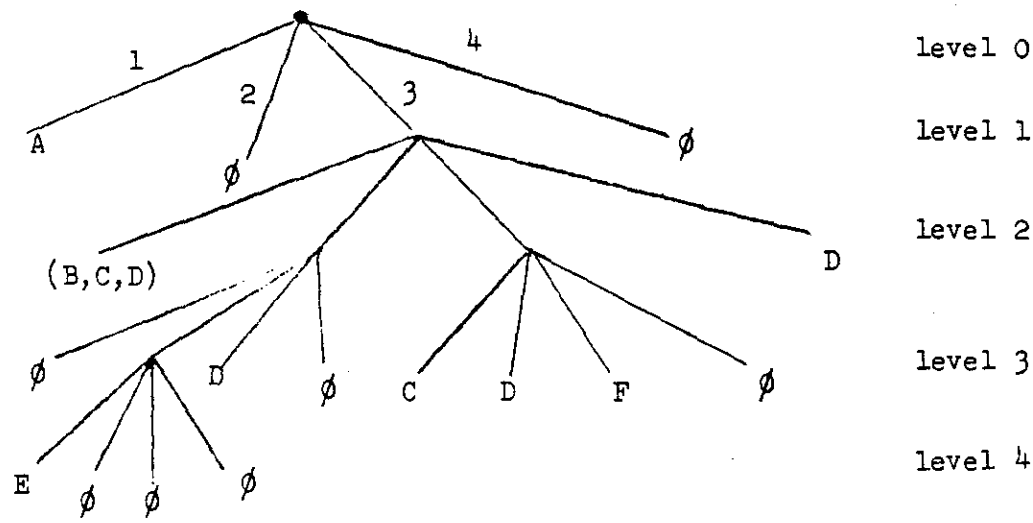


FIGURE 2.3

The letters of the nodes represent pointers to the actual object data where ∅ represents an empty region. At each level, the area of a region is one-fourth the area of the region that contains it. The length of an edge of a region at level n is 2^{-n} times the length of the edge of the region at level 0. For example, let the window contain the entire two-dimensional field. Let the addressability be 1024 raster units. Then a square at level 10 would be one raster unit on a side, and no further levels need be considered. This follows from the observation that an edge of region i is twice the length of an edge at level i + 1.

The display of data is accomplished by first determing the data in the display window and then retrieving that data from the data base and displaying it. For example, suppose the window were entirely within

region 1. It would then not be necessary to examine the data in regions 2, 3, or 4. This allows quick elimination of potentially large amounts of data.

Small non-resolvable objects can be eliminated by determining the size of the enclosing square relative to the size of the window. Given a window size, it is easy to determine the smallest size square that need be considered.

With all of its advantages, this scheme does have several drawbacks. Consider Object D. With the particular subdivision shown, it belongs to four regions at two different levels. This complicates the problem of updating the data base. Suppose we move Object D so that it is now contained entirely in region 3.1. The three remaining pointers from the structure to the data will have to be deleted. To do this requires either the use of back pointers from the data or a search through the entire data base for all references to the object in question. Neither of these approaches is desirable.

From Figure 2.2 we notice that Object D is small enough to be contained in a square the size of the one containing Object E. However, since the D belongs to larger squares, it will be considered for display even when the window size indicates it would be below resolvability. The only solution would be to subdivide its enclosing regions which adds unnecessary levels to the tree structure in the data base.

The second approach is to associate the regions directly with the data. For this approach, a region is a rectangle that encloses an object or group of objects.

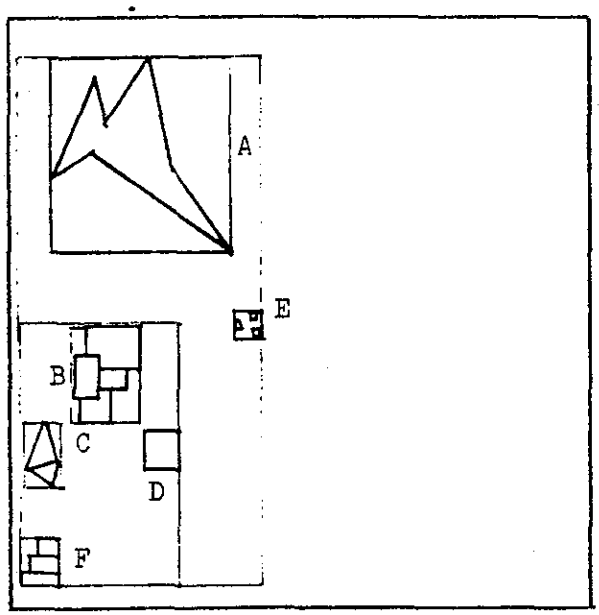


FIGURE 2.4

Figure 2.4 shows the same data only using enclosing rectangles. Note the rectangle enclosing Objects B, C, D, F. By clustering groups of objects together, we can form the necessary hierarchical structure. Without this clustering of objects, a test must be performed for each object to determine its position relative to the window. The clustering serves the same function as the subdivision into successively smaller squares.

For the simple example in Figure 2.4, the derived tree takes on the structure as shown in Figure 2.5.

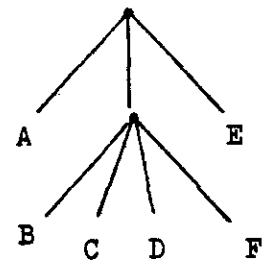


FIGURE 2.5

The choice of how to cluster the data is, of course, somewhat arbitrary. In Figure 2.5, a particularly simple choice was made. What is important to note is that each branch is associated directly with data and not with space. With the exception of a cleverly designed pathological case, the tree structure used here will always be simpler than the one used with the subdivided squares.

Only when the window intersects the rectangle associated with the root of the tree, does the data retrieval take place. In a similar manner, only when the window intersects the rectangle associated with objects (B, C, D, F), does the system even "know" that these objects exist. Clearly, the efficiency of the data retrieval is a direct function of the nesting structure of the rectangles. Strategies for optimum structures for a given data base fall outside the scope of this research. The clustering is performed by the user when the data base is set up.

The elimination of non-resolvable data works in the same manner as described previously. In this case, the ratio of the rectangle to the window is examined to determine if it is small enough to be considered non-resolvable. Here we cannot establish a cutoff based upon the depth of the tree, but must search each branch of the tree until the limit of

resolvability for the rectangles encountered is reached. We examine the ratio of the longest edge of the rectangle to an edge of the window.

This scheme has some advantages over the subdivided squares. We first notice that the size of the rectangle more truly represents the size of the enclosed data than does the square. This is important since the subsequent examination of the data depends on the relative size of the region that encloses it. A second advantage is that updating of the data base is simplified. Data and its regions can be moved without affecting the remaining structure. Also, the associated tree is generally smaller, but not as regular.

C. A Three-Dimensional Organization

In our examination of alternative organizations, extensions of the two schemes described above were tried. They are described in the following two sections.

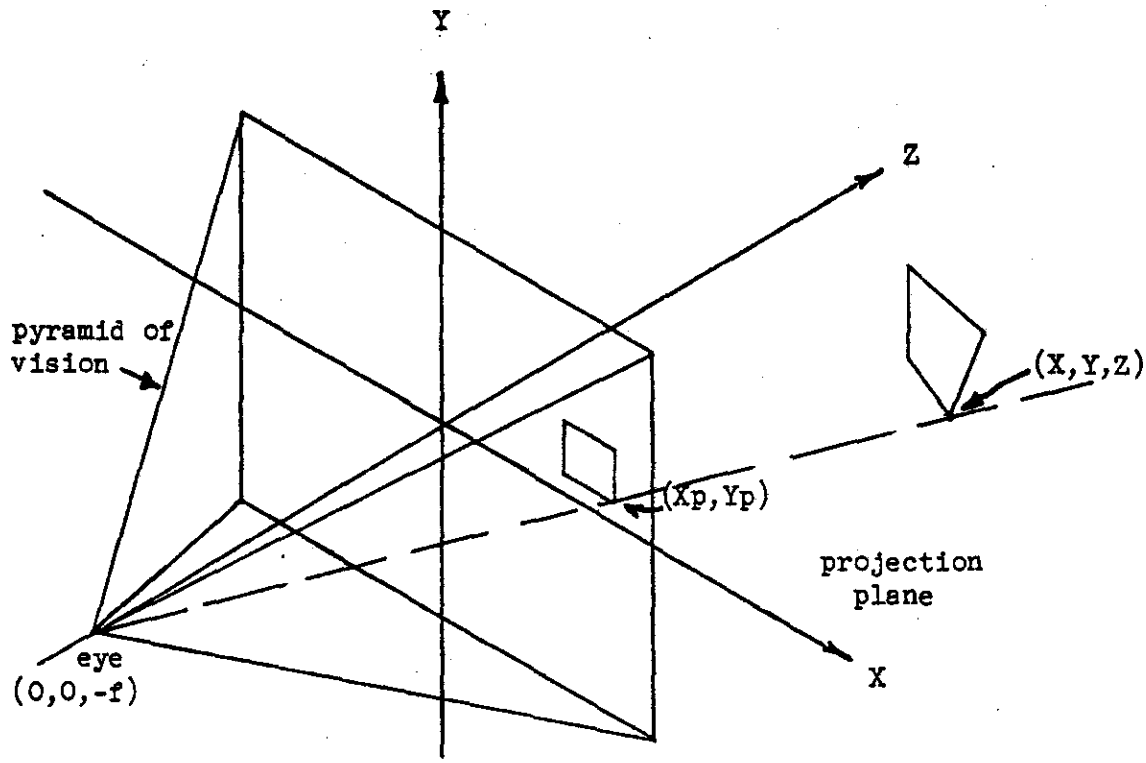
D. Physical Partitioning of the Space

The successive subdivision of the square becomes the successive subdivision of a cube. At each subdivision, eight new cubes are formed. The tree that is formed will take the same shape as the one shown in Figure 2.3, except there are eight branches at each node. The principle for retrieval and display of data are essentially the same with two complications due to the three-dimensional data base.

With two-dimensional data, determination of resolvability of the data within a region amounts to comparing the ratio of the edge of the region in question to an edge of the viewing window. This gives a measure of the size of the region and, hence, an upper bound on the size of the data contained within the region.

In three dimensions, the effective size of the region depends not

only on the actual size of the region, but on the distance from the viewing position to the region. This is due to the perspective projection used. The apparent size of a region is inversely proportional to the distance from the viewing position to the cube (see Figure 2.6).



$Xp = \frac{f \cdot x}{f - z}$ FIGURE 2.6 $Xp = \frac{f \cdot y}{f-z}$ $(f < 0)$

The example in Figure 2.6 shows the viewing position lined up along the Z axis with the plane of projection in the X - Y plane.

The second complication is the determination of position of the cube relative to the pyramid of vision. Since the user is not restricted to position and orientation, the cube can likewise assume any orientation relative to the planes bounding the pyramid of vision. This requires that the intersection of the cube with the pyramid of vision on all four bounding planes be computed. This requires that each of the eight vertices of the cube be tested against each plane that form the pyramid of vision to determine on what side of the plane each point lies.

Only when all eight points lie on the outside of any of the four planes, can the objects contained inside the cube not be considered for display. This amounts to 32 calculations of the form $y = ax + by + cz + d$. If the projected cube covers an area large enough to be considered resolvable, then all data connected with that cube will be processed for display. Note that objects can be attached directly to a region that is to be further subdivided. These directly attached objects are processed in addition to subdivided regions (if any). Since a region may intersect with the pyramid of vision, each of the potential eight subregions must be considered to determine their potential visibility. However, if a region is found to be totally within the pyramid of vision, then so will its subregions and, of course, they need not be tested for visibility. These subregions must be checked for resolvability.

This scheme has essentially the same drawbacks as its two-dimensional equivalent, only compounded by extra calculation due to the third dimension.

E. Logical Partitioning of the Data

The extension of the similar two-dimensional data organization requires that each object (or group of objects) be enclosed in a rectangular prism with sides equal to the maximum in x, y, z of the objects(s). The tree generated by the clustering does not become more complicated due to the extension to three dimensions. This is because the branching is based on the isolation of data in the space rather than on an isolation of volumes of space containing data.

This scheme has a number of advantages over physical partitioning. First, there are more likely to be fewer regions that need be tested against the pyramid of vision for visibility. Secondly, since the

region is more likely to give a better estimate of the size of the object(s), a more accurate estimate of resolvability can be made. In addition, updating of the tree is easier, as has been described for the two-dimensional case.

However, there is still the problem of testing for the visibility of a region. Each region still requires a test of eight vertices against each of the bounding planes of the pyramid of vision. If the number of regions is large, the cost of the tests may be prohibitive. This is particularly frustrating when, for an "average" picture, most of the data (and hence the regions) will be within the pyramid of vision. The results of virtually all the tests will be that the object is entirely in or outside of the pyramid of vision. It is a fairly unusual picture where most of the objects intersect the pyramid of vision.

There is also the determination of the area of the display screen occupied by the region. Both the cube and the rectangular prism have eight points on a two-dimensional screen. Figure 2.7 shows a typical case. The area covered by the region is determined

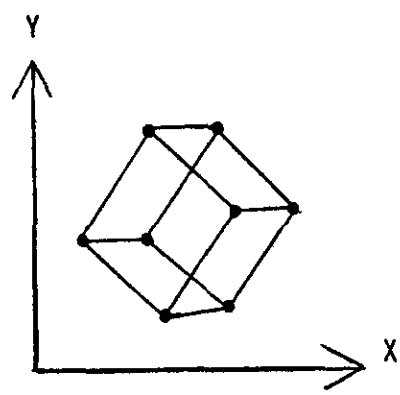


Figure 2.7

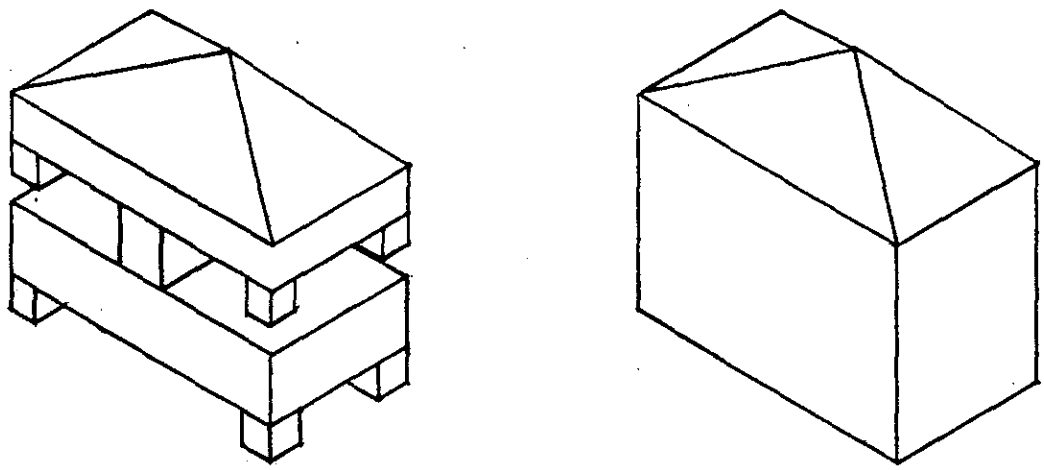
by first computing the minima and maxima of each of the eight projected points. The square generated from these minima and maxima gives a measure of the projected size of the data contained in the region. This calculation must be performed for every region in or intersecting the pyramid of vision.

In the next section, we will describe a modification of the Logical Partitioning Scheme that addresses these problems.

F. Partitioning with Spheres

Based upon our studies of the previously described data organizations, we set ourselves a number of goals that this new organization should attain. The first goal is that the testing of a region for visibility should be very fast. A single test per region per plane is desirable. The attaching of the region to the object(s) should be the best solution to the problem of updating of the data base. In addition, this gives a better estimate of object size to be used in the test for resolvability. We feel that the binary decision of processing or not processing of the data, depending upon the size of the region on the projection plane, is too severe a test.

Consider the complex object in Figure 2.8a. It is composed of 11 simple objects with a reasonable amount of detail. If the region containing this complex object occupied a total area on the display of less than two square raster units, then only a single point need be considered for display since no detail could be shown. On the other hand, if this complex object occupied 25% of the display, then the entire data should be processed and displayed. It is clear that the more accurately we know the size of the object, the more likely it will be that we will not process an object that cannot be resolved anyway.



11 objects
136 edges
59 faces
(a)

1 object
9 faces
(b)

FIGURE 2.8

There is a problem when the complex object occupies an area too small for all the detail but too large for a single dot. This will occur when the user is moving through a space containing these complex objects. When the objects are far away, they appear as points. As the user's position changes, the objects grow larger slowly. It seems reasonable to add detail only as needed. The object in Figure 2.8b could be used as a representation of the complex object 2.8a until the detail in the complex object could be resolved. This would save a great deal of display processing time.

Our goal for this requirement is multilevel descriptions of an object corresponding to the amount of detail that can be resolved. It should be possible to have an arbitrary number of descriptions of the same object available for display. The proper one is to be selected according to a user set threshold based on the area of the screen occupied by the object.

G. A New Organization

The regions that enclose the data are spheres. A sphere has some properties that move us close to the goals we have set. The test for visibility is very fast. We substitute the center of the sphere (X,Y,Z) into each of the bounding planes of the pyramid of vision $(ax + by + cz + d)$. The normal distance from (X, Y, Z,) to the plane is

$$D = \pm \frac{ax_1 + by_1 + cz_1 + d}{(a^2 + b^2 + c^2)^{1/2}} .$$

Let the sphere have radius R. We set up the parameters of the plane equations so that D is positive for a point on the inside of the pyramid while a negative distance is outside (see Figure 2.9). From this

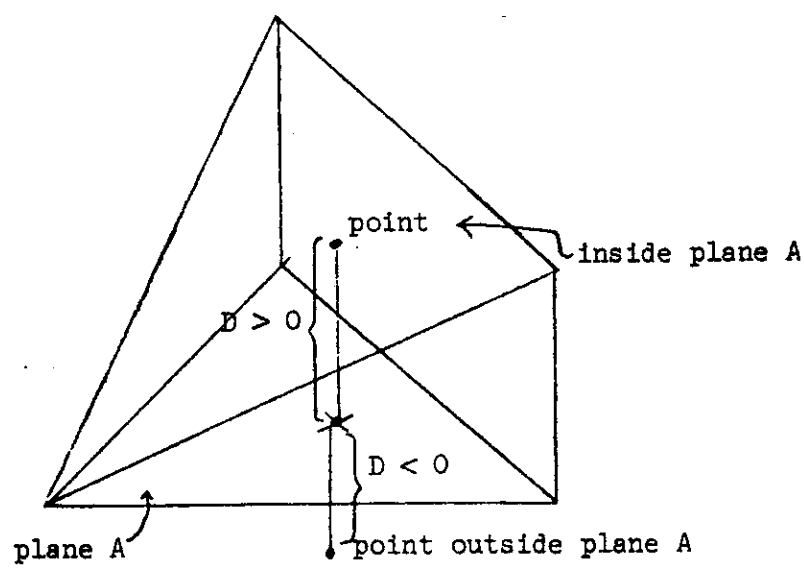


FIGURE 2.9

single test (as opposed to 8 for the rectangular prism), the following three conditions can arise:

1. $D + R < 0$ for <u>any</u> plane. The sphere is outside of the pyramid of vision and need not be considered for display.
2. $D - R \geq 0$ for <u>all</u> <u>four</u> planes. The sphere is completely inside the pyramid. There are no edges from any of the objects contained in the sphere that intersect the pyramid of vision.
3. The sphere intersects the pyramid. This means that it is possible that only a portion of an object may be visible. If this is the case, then the object must be clipped; i.e., the determination of intersection of edges with the screen (see Figure 2.10).

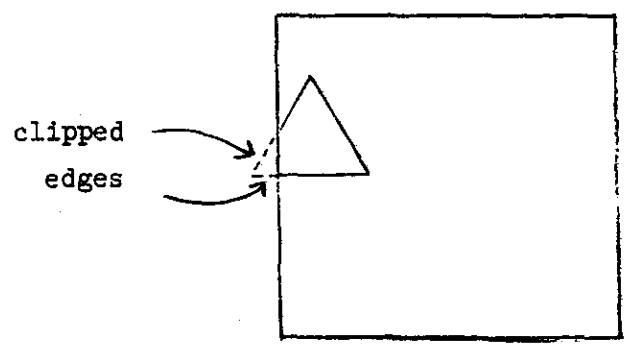


FIGURE 2.10

The clipping of the edges is time consuming and done only for those spheres that fall into case 3 above. The clipping can be performed in three-dimensions prior to projection, or after projection in two-dimensions. Clipping in two dimensions involves less calculations than when performed in three dimensions. There is only one case where clipping must be performed in three dimensions. This case occurs when an edge has one endpoint in front of the viewing position and one endpoint behind (Figure 2.11).

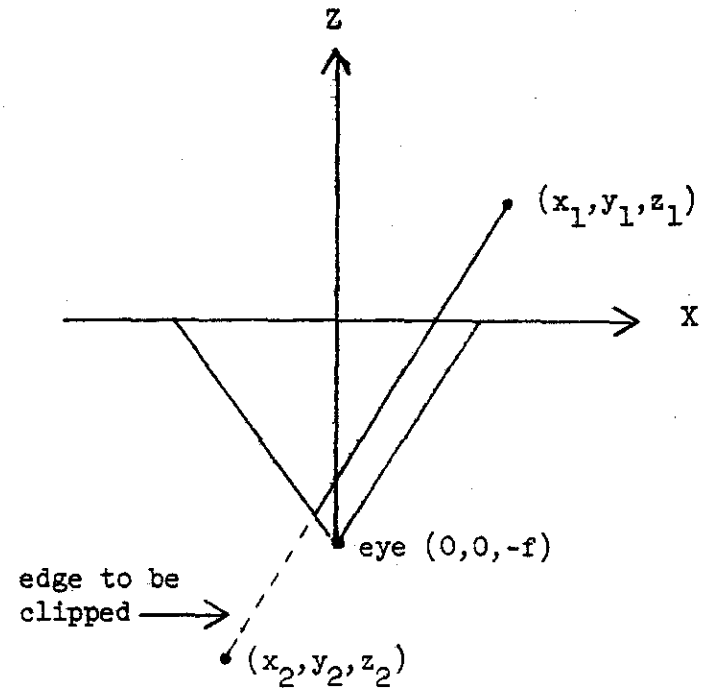


FIGURE 2.11

In addition, care must be taken in the projection of points with Z_2 near -f. This is because X_p and Y_p (Figure 2.6) are inversely proportional to Z-f.

If we know that the object is entirely in front of the eye position, then two-dimensional clipping can be used. This can be determined by calculating the distance from the center of each sphere to the eye position. If this distance is greater than the radius of the sphere, then two-dimensional clipping can be used.

The spheres also give us a fast measure of size. This measure is used to decide when detail is to be added. These are the cases when the sphere's projected size changes from a point through one or more small sizes according to the detail present. The data structure must contain user set thresholds that indicate at what projected size more detail is to be added.

One advantage of using spherical regions is that their projection is the same regardless of orientation. The projected area is directly proportional to the radius and inversely proportional to the distance from the center of the sphere to the viewing position.

From the equations in Figure 2.6, we note that size of the projected radius R is $R_p = \frac{f \cdot r}{f - z}$ and the projected area is πR_p^2. Rather than actually compute the projected area, we can set our thresholds in terms of the projected radius instead. This clearly gives the same result. Thus, the single calculation of R_p can be used as an indication of when a new description of the object in question is required.

An alternate scheme is to use the distance to the sphere as an indicator. This is an equivalent test since the size of the sphere is inversely proportional to Z. We can associate a critical distance with each sphere. When this distance is reached, more (or less) detail is added (deleted).

To accomplish all of this detail addition, we propose a structure we call the Choose One Consider All or COCA tree (Figure 2.12). This differs from the other tree structures described earlier in that alternate levels have different meanings.

The data definition nodes are located at a "consider all" level and are held on linked lists. They contain pointers to real data or a pointer to a data description node. Each of the nodes is considered independently for display. For nodes that point directly to data, their enclosing spheres are clipped to determine visibility conditions. If the data is in the field of view, then it is retrieved and displayed.

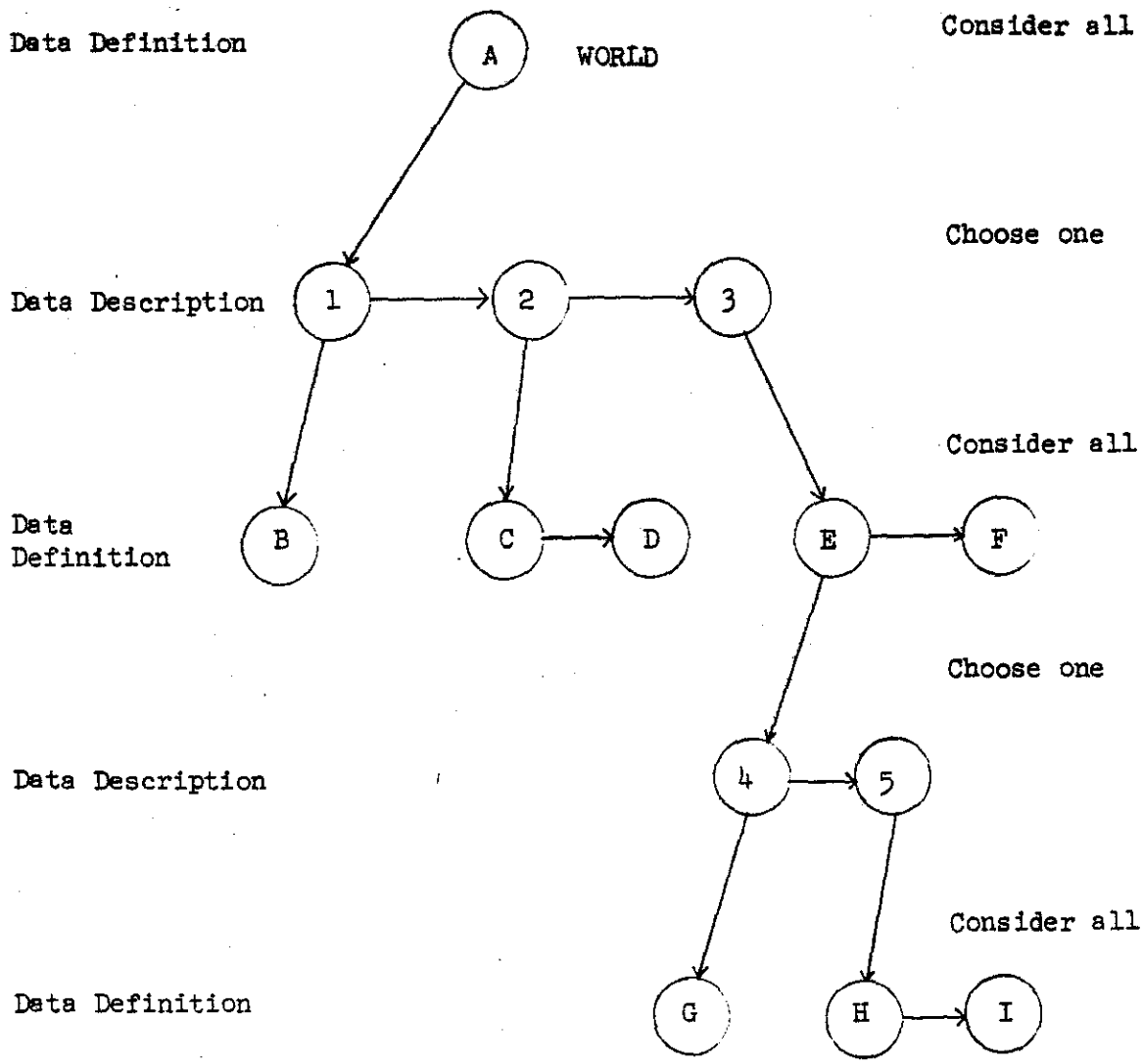


FIGURE 2.12

Those nodes not pointing directly to real data, point to a data description node. These nodes lie on a "choose one" level, and also are held on linked lists. Each list pointed to by a data definition node contains different descriptions of the same object. These descriptions are ordered with the least detailed description first and the most detailed last. It is the size of the projected radius of the sphere con-

taining the data that determines which of the data descriptions is used. Let us consider the tree structure in Figure 2.12.

Node A is the top of the tree and by definition is the description of the world. We have three possible descriptions. The first and least detailed is a single object B. The second detailed description consists of two objects C and D, both of which are to be displayed if the pre-set threshold so indicates. The most detailed description consists of two objects, E and F. If description three is chosen, then object F will be tested for visibility and displayed. Since object E points to a data description node, the sphere containing object E will be used to choose either object 4 or 5. Depending upon this test, either object G, or objects H and I, will be displayed.

It is important to remember that these thresholds associated with the data description nodes are set by the user. These values are determined by looking at the object on the display and deciding at what projected size more detail is necessary.

H. Summary

A new method of organizing data has been described. It has the advantage of allowing multiple descriptions of objects in order to retrieve only resolvable data. It provides for a fast test to determine both visibility and level of detail required.

This organization also provides the ability to "fine tune" the display by adjustment of display thresholds. It provides, in a single structure, a method for viewing a data base containing a large number of objects (e.g., a city from far away) as well as a detailed picture of a single entity.

CHAPTER III

On Calculation of Spheres

In the previous chapter, we have seen the need for enclosing objects in spheres. The solution to this problem proved to be one of the most interesting aspects of our research. The initial approach taken was to find an approximate sphere by using the centroid of the vertices of the data to be enclosed, and generating the sphere whose radius is the distance from the centroid to the point farthest away. An argument can be made for the case that the mininum sphere is not necessary, but that an approximate sphere would suffice. This proved satisfactory only when the points were uniformly distributed in S, Y, and Z, which is not typical for the types of objects likely to be encountered. In fact, most of the schemes we examined that attempted to use the points to find a center and construct a sphere generated spheres too large for some fairly typical data.

The most obvious approach is some sort of iterative scheme that starts with a large enclosing sphere, eventually reaching the minimum sphere. An algorithm was reported that claimed to have solved this problem.[7,3] It was stated that given n points, the minimum volume sphere would be found prior to generating a maximum of $\binom{n}{4} \cdot \binom{n}{2}$ spheres. In fact, it was claimed that far fewer spheres would be generated, but offered only a single two-dimensional example to back up this assertion.

The algorithm was not implemented in three dimensions.[8] The algorithm is quite complex (20 special cases) with no indication of speed of convergence as a function of the input data. For these reasons, no comparison will be made of this algorithm with the algorithm we develop.

The algorithm described here is conceptually very simple. Take the points 2, 3, and 4 at a time and construct spheres, testing each one to see if it is the smallest one. For n points, this amounts to $\binom{n}{4} + \binom{n}{3} + \binom{n}{2}$ possible spheres which is not practical for even moderate n.

It occurred to us that a possible method for determing what points of the data set are most likely to be on the minimum volume sphere would be to first find the convex hull of the data set. Faces are generated from three points, which gives only triangular faces. If more than three points lie in a plane, then several faces will be generated that lie in the same plane (see Figure 3.1).

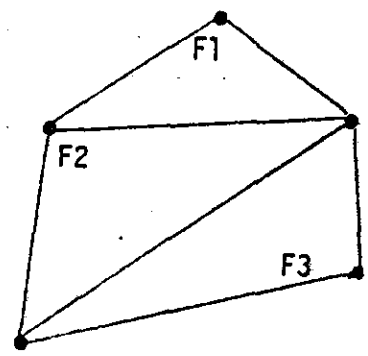


FIGURE 3.1

Only those points <u>on</u> the convex hull could possibly be on the minimum volume sphere. To reduce the number of spheres tested, the points are sorted according to how "sharp a point" they are. As an approximation to "sharpness", we choose to calculate the solid angle each point on the convex hull subtends and sort the points on ascending solid angle. These points are then examined in order by the minimum sphere algorithm. These two algorithms will be described in the next two chapters.

CHAPTER IV

Convex Hull Algorithm

A search of the literature provides only a single algorithm on calculation of a convex hull in more than two dimensions.[2] Since a personal communication with one of the authors failed to provide any information on performance of the algorithm (company private), a detailed comparison of the timing of the two algorithms is not possible. However, an analysis of the calculations required versus the size of the data set will show that our new algorithm requires nearly an order of magnitude less calculations than the Chand-Kapur algorithm. Both of these algorithms will operate on n-dimensional data. The discussion here will concern data in three dimensions only.

A. Chand-Kapur Convex Hull Algorithm

The basic principle of this procedure is that given one edge and one of the faces containing that edge, a second face can be found by a process which is equivalent to the rotation of a face about an edge. A calculation is performed using the normal of the known face and each of the points with each of the edges on the convex hull. This calculation is of the form

$$\lambda = \frac{\bar{a} \cdot \bar{b}}{\bar{c} \cdot \bar{d}} \quad \text{(a,b,c,d are vectors)}$$

We will see that for large data sets, this calculation is the major computational effort in this algorithm.

For our purposes, we can say that this algorithm is initialized by first finding one face on the convex hull. This gives rise to three edges. For each edge and every point, an expression of the form λ is

calculated which is a measure of the angle the plane forms with that edge and each point makes with the known plane. The new plane, whose angle is maximum with respect to the given plane, is also on the convex hull.

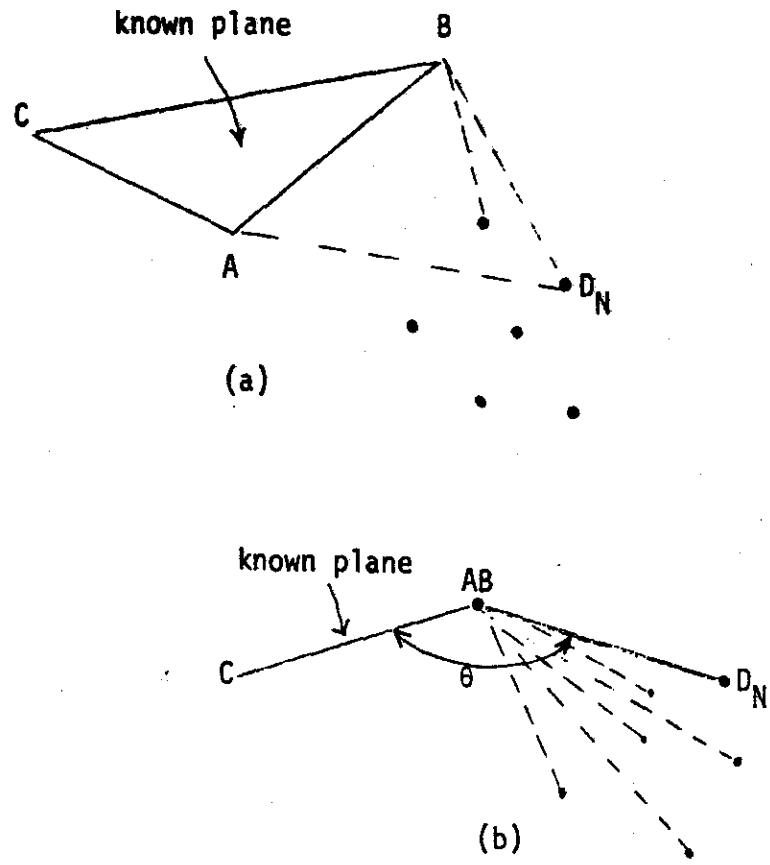


FIGURE 4.1

Figure 4.1 shows that the plane formed with edge A,B and point D_N would be selected since θ is a maximum for that plane. Edges AD_N and BD_N are then stored for later testing. Since each edge is contained in exactly two planes, only those edges not previously stored will be tested. This process is continued until no new edges are generated.

For purposes of later comparison, we state that a dot product is to be a single unit of calculation. This unit of calculation will be referred to as DP. We will estimate the calculation effort of the algorithm in units of DP.

let: NPH = number of planes on the convex hull

NPTS = number of points in the data set

Each plane contains three edges. Since each edge is shared by exactly two planes, then there are $\frac{3}{2}$ * NPH edges to be tested with <u>all</u> of the points in the data set. Each test of a point requires 2DP. This is the calculation of λ requiring two dot products and a divide. Thus, a measure of the amount of calculation required to compute a convex hull is

$$N_{CH} = \frac{3}{2} \text{ NPH} * \text{NPTS} * 2 * \text{DP} = 3 * \text{NPH} * \text{NPTS} * \text{DP}$$

We will ignore the divide used in calculation of λ at this time.

B. <u>New Algorithm</u>

This algorithm consists of an initialization phase and refinement steps.

During initialization, a number of extreme points are isolated and a tetrahedron is formed. The refinement phase takes the points outside of the tetrahedron and forms convex caps, replacing some of the planes but maintaining convexity. This continues until there are no more points outside the convex polyhedron.

C. <u>Initialization</u>

<u>Step 1</u>: Find at least three non-colinear points that lie on the convex hull.

Let {S} be a set of points in three dimensions. Let P be a plane of the form ax + by + cz + d = 0 lying anywhere in this space.

The normal distance from point (x_1,y_1,z_1) to this plane is $D = (ax_1 + by_1 + cz_1 + d)/(a^2 + b^2 + c^2)^{1/2}$. The distances to all points on one side of the plane are positive, while the distances are negative on the opposite side (Figure 4.2). The choice of sign is arbitrary. Let $D_S = ax_1 + by_1 + Cz_1 + d$ be defined as the signed distance from (x_1,y_1,z_1) to the plane P. D_S is proportional to the normal distance.

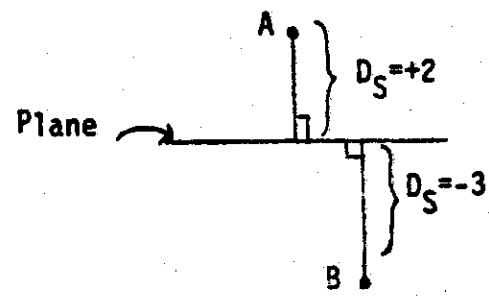


FIGURE 4.2

Using the signed distance, we see that the set of points $\{S_1\}$ whose signed distances are maximum and minimum with respect to any plane, are on the convex hull. For this not to be true would imply that the maximum (minimum) point lies inside some plane, Pm, on the convex hull, and that implies that Pm contains a point at a distance greater (less) than D_S.

Since any plane can serve to find points on the convex hull, we can minimize the amount of calculation required by choosing planes whose coefficients are simple. The signed distances we use are as follows:

$$D_{S1} = X$$

$$D_{S2} = Y$$

$$D_{S3} = Z$$

$$D_{S4} = X + Y + Z$$

$$D_{S5} = -X + Y + Z$$

$$D_{S6} = X + Y - Z$$

$$D_{S7} = -X + Y - Z$$

The minima and maxima of the above 7 signed distances give rise to <u>up to</u> 14 points on the convex hull. For example, consider the two-dimensional set of points in Figure 4.3. Note that in two dimensions, the planes becomes lines. The set of signed distances for two dimensions are:

$$D_{S1} = X$$

$$D_{S2} = Y$$

$$D_{S3} = X + Y$$

$$D_{S4} = -X + Y$$

The points giving the maximum and minimum signed distances are:

	Max.	Min.
D_{S1}	A	E
D_{S2}	A	D
D_{S3}	A	D
D_{S4}	F	C

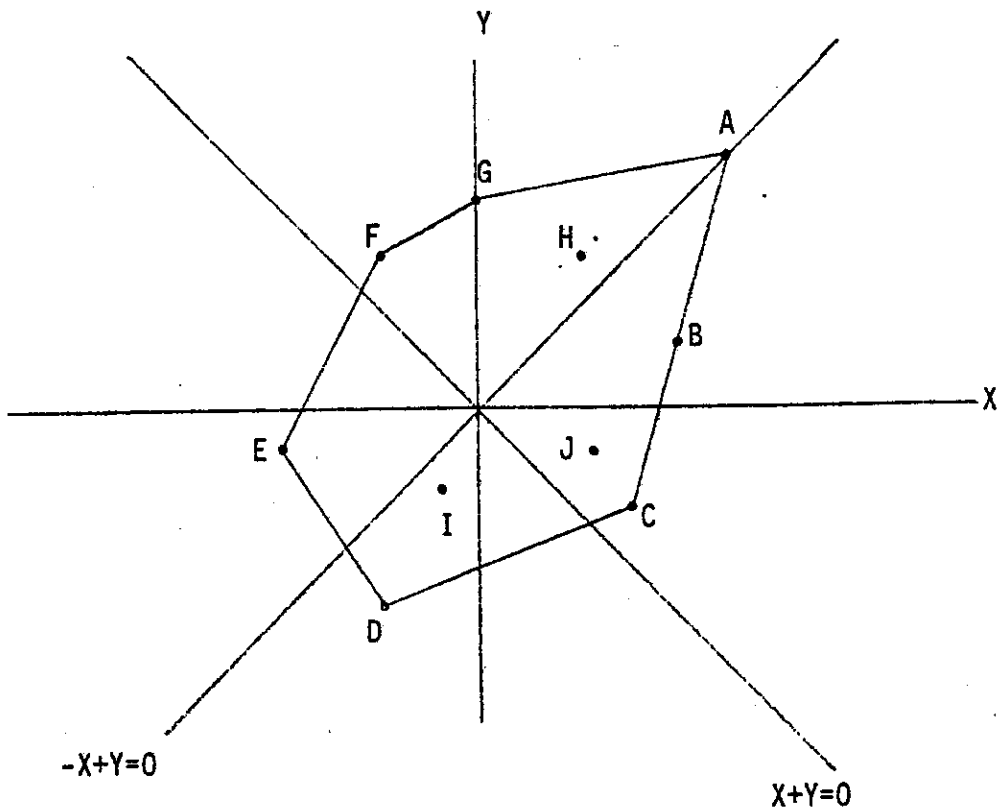


FIGURE 4.3

Of a possible eight points, only five different points on the convex hull were found using the simple planes. Points B and G, which are actually on the convex hull, were not found at this stage. After this calculation, we have a list of points that are known to be on the convex hull.

Returning to three dimensions, if at this point there do not exist three non-colinear points, then the set {S} lies in the lower dimensional space. The algorithm can then find a minimum volume circle enclosing the set {S}. Assume that three points (A,B,C) have been found to lie on the convex hull.

<u>Step 2:</u> Form the triangle passing through the three points (A,B,C) from Step 1. Compute the equation of the plane containing the triangle.

Step 3: Find the point TP farthest from the plane (A,B,C). From Step 1 we see that TP is also on the convex hull. Construct a tetrahedron with TP and (A,B,C). This tetrahedron is a polyhedron with all its vertices residing on the convex hull. Note that three new triangles have been calculated.

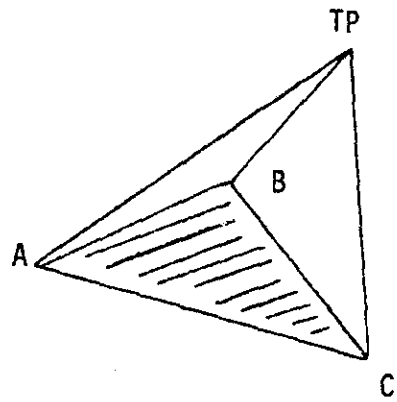


FIGURE 4.4

These triangles are all potential faces on the convex hull. As these faces are found, the algorithm orients the points so that the signed distance (D_S) from a point outside the convex polyhedron to its nearest face is always positive (see Figure 4.5).

Face	D_S(TP)
1	+
2	+
3	-
4	-
5	-

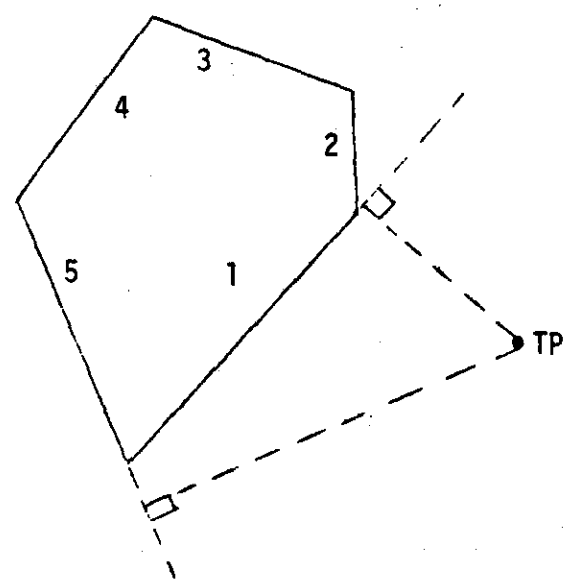


FIGURE 4.5

If the distance is positive to any face on the convex polyhedron, then Tp is outside the polyhedron. Conversely, if $D_s(TP)$ is negative for <u>all</u> faces, then TP is inside the convex polyhedron.

D. <u>Refinement</u>

At this point, we are ready for the refinement phase of the algorithm. The purpose of refinement is to change the convex polyhedron to the convex hull by incorporating those points that lie outside the convex polyhedron. The input to refinement is a list (FL) of faces forming a convex polyhedron, all of whose vertices are on the convex hull, and a list of points on the convex hull not on the polyhedron (TL). The polyhedron is the tetrahedron formed in Step 3, and the list is formed from those points calculated in Step 1 that were <u>not</u> used on the tetrahedron.

<u>Step 4:</u> The following procedure is followed for each of the points on TL.

Let the point under consideration be referred to as TP. Let $D_S(a)$ be the signed distance from a plane to point a.

<u>Step 4a:</u> Find a face on FL for which $D_S(TP)$ is positive. Let this face be denoted as PL.

<u>Step 4b:</u> Starting with PL, find the rest of the planes whose signed distances to TP are also positive and move them from list FL to a list of planes to be deleted (TBD).

The notion of positive signed distance to TP to a face is equivalent to stating that the face under consideration is "visible" to a person situated at TP, looking at that face. Visibility can also be defined by using the dot product of a vector from the face

to TP and the normal to the face (see Figure 4.6). We will show that this is identical to the signed distance.

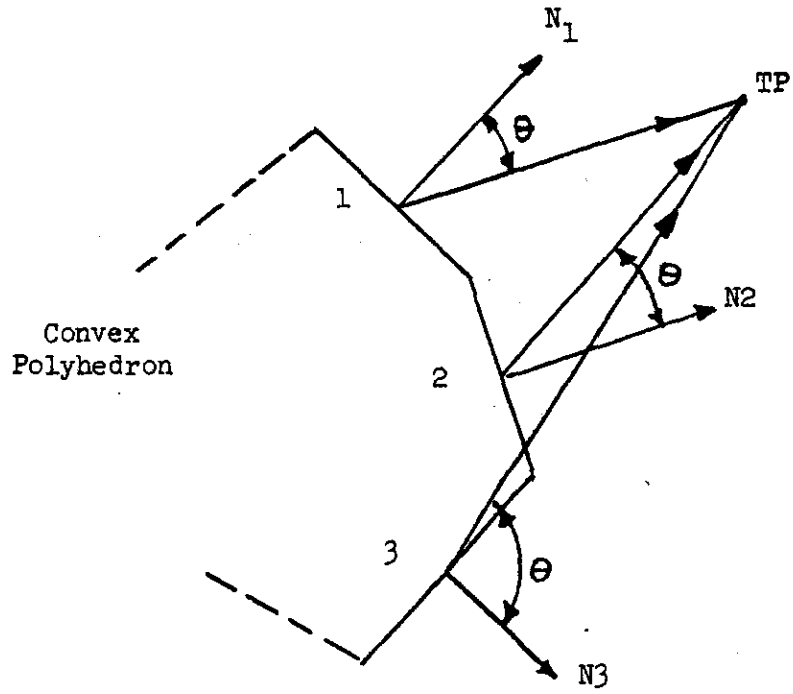


FIGURE 4.6

Let $ax + by + cz + d = 0$ be the equation of plane P.

Let $N = (a,b,c)$ be a normal to P. With no loss of generality, assume this is an outward facing normal.

Let $\bar{V} = (XP - X,\ YP - Y,\ ZP - Z)$ be a vector from P to TP.

$\therefore\ \bar{V}\cdot\bar{N} = aXP + bYP + cZP - (aX + bY + cZ) = |V|\ |N| \cos(\theta)$ where θ is the angle between the vectors.

Note that the sign of $\bar{V}\cdot\bar{N}$ is positive for $-90^\circ < \theta < 90^\circ$. Thus, the sign of $\bar{V}\cdot\bar{N}$ indicates if a face is visible or invisible from any point. This is identical to testing the signed distance

$$D_S(TP) = aXP + bYP + cZP + d$$

from P we get

$$d = -(aX + bY + cZ)$$

$$\therefore \quad D_S(TP) = aXP + bYP + cZP - (aX + bY + cZ) = \bar{V}\cdot\bar{N}$$

Given a single visible face, the process of locating the remainder of the visible faces does not involve testing all of the faces on FL. The data structure in which the convex polyhedron is embedded contains links which connect adjacent faces to one another. Only the faces adjacent to visible faces will be tested. Remember that a face on a convex polyhedron is either entirely visible or invisible.

<u>Step 4c:</u> Construction of a convex cap. From Step 4b, we can find the edges that are shared by visible and invisible faces. The extreme edges are those edges shared by a visible and invisible face. Again, if a person were situated at TP, then the extreme edges appear as the outline of the convex polyhedron, as viewed from TP.

New faces (triangles) are now constructed using TP and each of the extreme edges. The visible faces on TBD are now deleted and the new faces added to the polyhedron. By construction, the polyhedron remains convex. Consider the example in Figure 4.7.

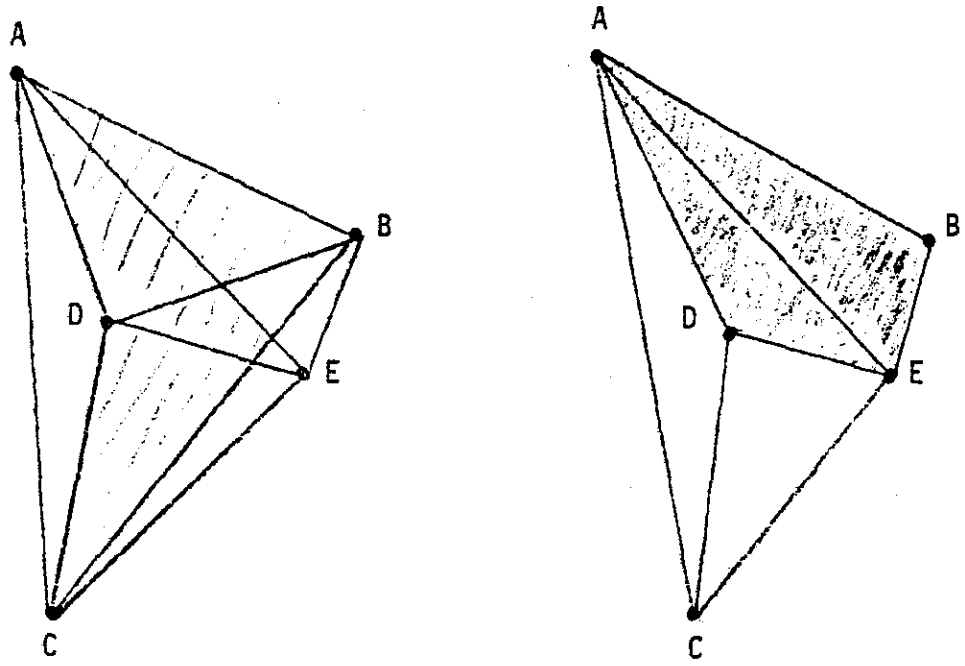


FIGURE 4.7

FL - list of faces on convex polyhedron (initially)

- ADC, ABD, ACB, BCD

TL - list of points on convex polyhedron not used in FL

- E - this is point TP

TBD- list of faces to be deleted

- ABD, BCD

FL - list of faces on convex polyhedron (final)

- ADC, ACB, AED, AED, CED, CEB

The polyhedron generated during Step 3 is the tetrahedron ABCD. It is defined by list FL. There is a single point (E) on TL, the list of points on the convex hull to be added to the convex polyhedron. Using E, two faces are found to be visible and are deleted, and four new faces added. The convex polyhedron now has six faces. If no more points are found outside the convex polyhedron,

then it becomes the convex hull. If there are points remaining outside, then the process is continued as described below.

As was stated earlier, Step 4 (a-c) is repeated for all the points on TL. For the previous example, only one point (E) was on that list. In most cases, many convex hull points would be on this list.

Step 5: All of the points not on the convex hull are now tested against all of the faces on FL, the current convex polyhedron.

These tests give the following data:

1) All points whose signed distance to all of the faces on FL is negative are inside the polyhedron. These points are deleted from the data set.

2) Those faces on FL for which no points have a positive signed distance are on the convex hull and are not considered in any further calculation.

3) If there are no points with a positive signed distance to any face, then the convex polyhedron is the convex hull and the algorithm terminates.

4) The points that have a maximum signed distance from each of the remaining faces on FL are transferred to list TL. This is the new set of points that are on the convex hull but not yet incorporated into the convex polyhedron.

Step 6: Go to Step 4.

This construction guarantees a unique relationship between the number of planes and the number of points on the convex hull. At each stage we add but a single point. A convex cap that replaces but a

single plane adds three new planes for a net gain of two planes. For every additional plane covered by the convex cap, two new planes are generated since each and every adjacent pair of planes share a common edge. The net gain of planes is zero since the plane sharing the common edge is deleted. Thus, each new point added to the convex hull increases the number of planes on the hull by exactly two. From the initial condition that the number of planes on the hull (NPH=4) is equal to the number of points on the hull (NPTH=4), we get

$$NPH = 2*NPTH-4.$$

This algorithm generates only triangular faces. We could reduce the number of planes without affecting the number of points on the hull if we combined adjacent faces that lie in the same plane. This would be highly undesirable during the construction because it would greatly complicate the data structure, and hence, add computation time to the algorithm. A test was made on the final convex hulls to determine if any planes could be combined. For the data sets we used, the number of planes that could be combined was found to be insignificant.

E. Computational Effort

From the description of the convex hull algorithm, it is clear that the major effort is the calculation of the signed distance, D_S. It will be shown experimentally that the number of calculations of D_S is far greater than the number of faces that are calculated. We will use this calculation as a means to compare this new algorithm with the Chand-Kapur algorithm.

The calculation of the signed distance involves one more addition than that of a dot product. We state, therefore, that a fair comparison

is that calculation of two signed distances is equal to two dot products and a divide. That is to say, the calculation of two dot products and a divide for the Chand-Kapur algorithm equals calculation of two signed distances. The measure we will use to compare will be calculation of signed distances versus dot products. The details of the performance of the algorithm are given in Chapter VII and Appendix II.

CHAPTER V

A. Minimum Volume Sphere

We should like to begin this chapter with a statement about why we take the approach we do. We do this because in the countless discussions with our colleagues, the question was continually raised, "but why not an iterative approach?". Our answer follows.

In the first place, we set as a goal that we want to calculate the exact minimum volume sphere. An approximation is not acceptable. We felt that any approximation would cause us to consider data in the sphere sooner than is actually necessary.

Secondly, the only algorithm we found that claimed to have solved this problem was incredibly complex and had not even been implemented. There was no indication of speed of convergence. It was totally unclear from the description how well it would work for all cases.

Thirdly, our algorithm works quite well. For some cases, the expected number of spheres created was four orders of magnitude greater than the actual number created. In addition, we found the approach of using a heuristic to speed the calculation most rewarding.

Finally, we could not derive an iterative scheme that worked as well on the large data sets we tried.

B. Definitions

A sphere is uniquely determined by four non-coplanar points. The equation of the sphere is calculated by solving the following four equations for a,b,c,d.

$$\{x_i^2 + y_i^2 + z_i^2 + ax_i + by_i + cz_i + d = 0\}_{i=1,4}$$

As long as the four points are not co-planer, a unique sphere will be generated. If we add constraints, other unique spheres can be generated.

4 Point Sphere: A sphere passing through four non-coplaner points and is unique by construction

3 Point Sphere: A unique sphere passing through three non-colinear points that lie on a plane passing through the center of the sphere.

2 Point Sphere: A unique sphere passing through two distinct points that lie on the opposite extremities of a diameter.

These spheres are not necessarily the smallest spheres that contain the points generating them. The sphere in Figure 5.1a is a three-point sphere but not the smallest sphere that contains those three points.

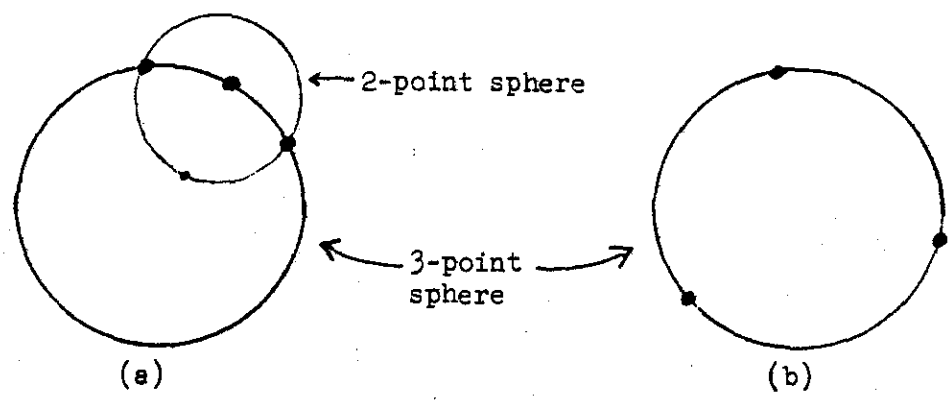


FIGURE 5.1

Clearly, there is a smaller two-point sphere that contains the same three points. The three-point sphere in Figure 5.1b is the smallest three-point sphere that contains these three points.

Minimum volume four-point sphere: A four-point sphere that is also the smallest sphere containing those four points

Minimum volume three-point sphere: A three-point sphere that is also the smallest sphere containing those three points.

A two-point sphere, as defined above, is also the minimum volume two-point sphere.

C. Conditions for the Minimum Volume Sphere

In this section, we shall show that the center of the minimum volume three (four) sphere must lie in or on the triangle (tetrahedron) formed by the three (four) points. We shall do this by showing that if the center does not lie in the triangle (tetrahedron), then a smaller sphere can always be constructed that will contain those points.

Figure 5.2a shows a circle of radius R passing through three points whose center lies outside the triangle formed by those points. Let the center of the circle be (0,0).

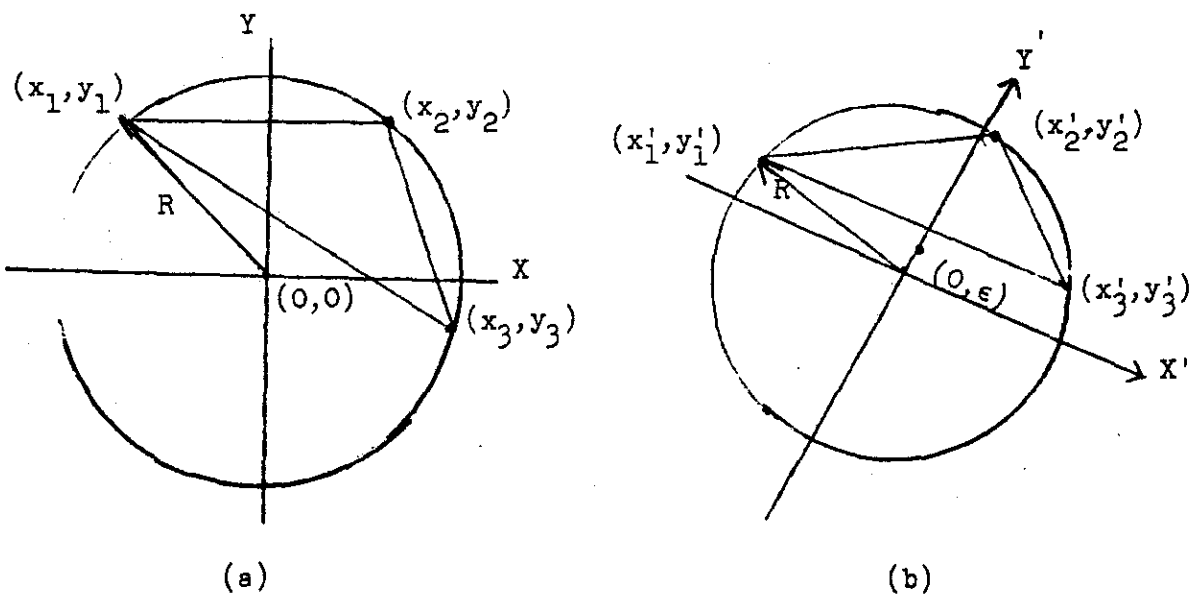


FIGURE 5.2

Rotate the coordinates about the center so that the x' axis does not intersect the triangle. In this new coordinate system (Figure 5.2b)

$$R^2 = x_1'^2 + y_1'^2$$

$$R^2 = x_2'^2 + y_2'^2$$

$$R^2 = x_3'^2 + y_3'^2 \quad .$$

Note that all of the y' have the same sign. We assume with no loss of generality that all the y' are positive. Consider the point $(0,\epsilon)$ where ϵ is an arbitrarily small but positive number. The distance from $(0,\epsilon)$ to each of the points is

$$R_1^2 = x_1'^2 + (y_1' - \epsilon)^2 < R^2$$

$$R_2^2 = x_2'^2 + (y_2' - \epsilon)^2 < R^2$$

$$R_3^2 = x_3'^2 + (y_3' - \epsilon)^2 < R^2$$

Thus, if we construct a circle of radius R with center $(0,\epsilon)$ (in the x', y' coordination system), all 3 points will lie inside the circle. Hence, the circle of radius R is not the smallest circle that contains the 3 points. Therefore, the smallest circle containing 3 points that passes through those points has its center inside the triangle formed by those 3 points. If the center lies on one of the triangle edges, then we have a 2 point circle.

In three dimensions, the triangle becomes a tetrahedron and the minimum volume sphere passes through four-points if the center lies inside that tetrahedron.

This condition that the minimum volume sphere must satisfy gives us a method for determining if a particular sphere we have calculated is the minimum volume one. Let {P} be a set of points in three dimensions. Let S be a four-point sphere. S is the minimum volume sphere containing {P} if

1) all points in {P} are either on or in S;

2) the center of S is in or on the tetrahedron formed by the four points that generated the sphere.

If S is a three-point sphere, the same conditions apply except that the center must lie in or on the triangle formed by the three points that generated the sphere.

For a two point sphere, the test is even simpler since the only pair of points that need be considered for sphere generation are those whose distance between them is a maximum.

D. Minimum Sphere Algorithm

Calculation of the minimum volume sphere consists of examining for minimum volume all possible three and four-point spheres, along with the two-point sphere generated from the two points farthest apart. This amounts to $\binom{n}{4} + \binom{n}{3} + 1$ spheres where n is the number of points in {P}. For n = 100, this amounts to a maximum of 4,082,924 spheres.

To speed the search for the minimum volume sphere, a heuristic has been used. As was stated earlier, the first step of the calculation is to calculate the convex hull of {P}. It is clear that only the points

on the convex hull of $\{P\}$ can be on the minimum volume sphere. The search of even this reduced point set was computationally not feasible since, for the data sets we examined, the number of points on the hull ranged upwards to nearly 200. We denote this reduced point set as $\{P_{ch}\}$.

The minimum sphere is determined by a maximum of four points. In the previous section, a condition was described that allowed us to test any sphere for minimum volume. Our goal is to generate the spheres in a particular order, such that the minimum volume sphere will be generated early in the sequence. This amounts to sorting the points in $\{P_{ch}\}$ in order of their likeliness to appear on the minimum volume sphere. We feel that, given a convex hull, the points that "stick out" the farthest are more likely to be on the minimum volume sphere. Our measure of the degree to which a point sticks out is to calculate the solid angle subtended by the point. The smaller the angle, the "sharper" the point. Note that the "sharpest" point is <u>not</u> necessarily on the minimum volume sphere.

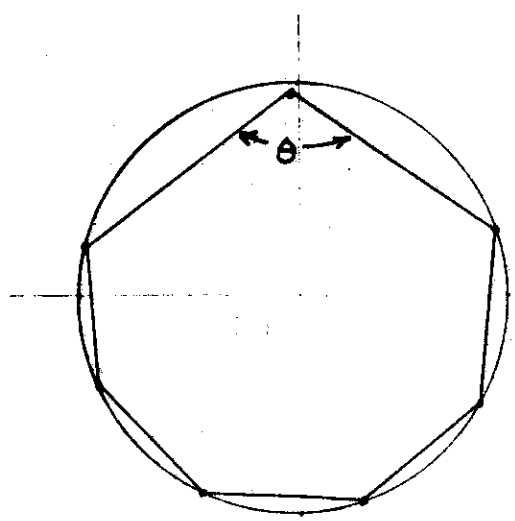


FIGURE 5.3

Figure 5.3 shows a two-dimensional case where the point on the convex hull with the smallest angle is the only one <u>not</u> on the minimum circle.

<u>Step 1a</u>: Calculate the solid angle for each point P_i on the convex hull. Consider a portion of the convex hull as shown in Figure 5.4.

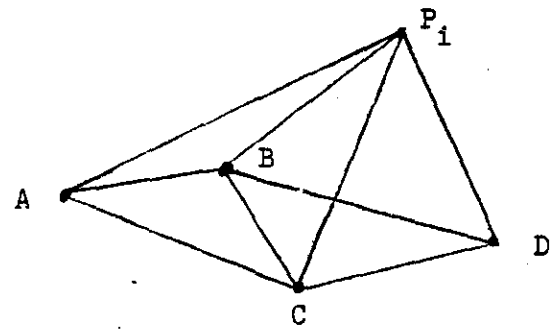


FIGURE 5.4

Let point P_i be the center of a sphere with a radius of unity. Let the triangles AP_iB, AP_iC, BP_iC, CP_iD, DP_iB be extended until they intersect the sphere as shown in Figure 5.5.

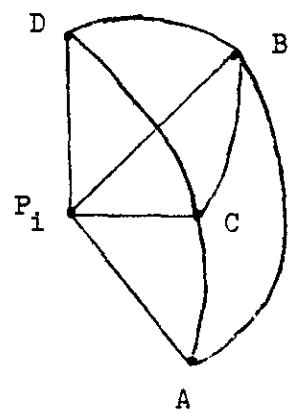


FIGURE 5.5

The solid angle is equal to the area of the portion of the surface of a sphere of unit radius, center at P_i, which is cut by the polar triangles with vertex at P_i. For this example, let us consider the single spherical triangle ABC. Let a denote the length of side BC, b denote the length of AC, and c denote the length of side AB. Since the radius of the sphere is unity, the lengths of the sides of the spherical triangle are as follows:

$$a = \angle BP_iC$$

$$b = \angle AP_iC$$

$$c = \angle AP_iB$$

The area of a spherical triangle with unit radius is

$$\text{Area} = \frac{\pi E}{180}$$

where $E = 4 \cdot TAN^{-1} \left(\tan(\frac{S}{2}) \cdot \tan(\frac{S-a}{2}) \cdot \tan(\frac{S-b}{2}) \cdot \tan(\frac{S-C}{2})\right)^{1/2}$

$$S = (a+b+c)/2$$

$$0 < a + b + c < 360^\circ$$

<u>Step 1b</u>: Sort the points in $\{P_{ch}\}$ according to ascending solid angle. This puts the "sharpest" points first.

<u>Step 2a</u>: Find the maximum square of the distance between each pair of points in $\{P_{ch}\}$. Generate a two-point sphere S_2 using this maximum.

<u>Step 2b</u>: Test each of the points in $\{P_{ch}\}$ to determine if they are inside S_2. If all of the points in $\{P_{ch}\}$ are inside S_2, then S_2 is the minimum volume sphere.

<u>Step 3</u>: Generate all possible three and four-point spheres from the sorted points in $\{P_{ch}\}$. The method of generation is given by

the incomplete PL/1-like program below. NPTS is the maximum number of points in $\{P_{ch}\}$.

```
DO L=4 TO NPTS
   DO I = 1 TO L-3
     DO J = I+1 TO L-2
        DO K = J+1 TO L-1
        IF K = L-1 THEN
          DO;
          Call GENERATE_3_POINT_SPHERE (I,J,K);
          IF CENTER_IS_IN_TRIANGLE (I,J,K) THEN
             IF ALL_POINTS_ARE_IN_SPHERE (I,J,K) THEN
               GO TO FOUND_3_POINT_SPHERE;
          END;
        Call GENERATE_4_POINT_SPHERE (I,J,K,L);
        IF CENTER_IS_IN_TETRAHEDRON (I,J,K,L) THEN
             IF ALL_POINTS_ARE_IN_SPHERE (I,J,K,L) THEN
                GO TO FOUND_4_POINT_SPHERE;
        END;
     END;
   END;
END;
```

Note that the conditions for minimum volume are checked prior to testing the points. We do this since it is a relatively short test compared to testing if all of the points lie in a sphere. Remember that without this condition, it is easy to generate very large spheres that can easily contain all the points, and we want to minimize the number of tests we

must make (see Figure 5.6).

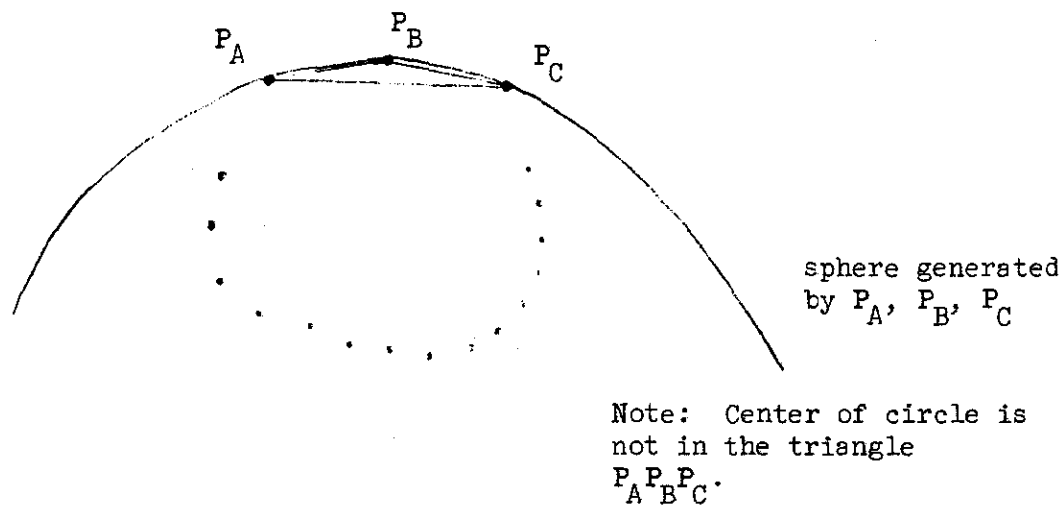


FIGURE 5.6

This procedure is continued until a minimum volume sphere is found.

E. Computational Effort

As was stated earlier, no existing algorithms were found to be compared with the algorithm presented here. We will, instead, compare the results with the algorithm without the use of heuristic to accelerate the search. We do this as a means to show the effectiveness of the heuristic.

If no heuristic is used, then any point is as likely as any other to lie on the minimum volume sphere. Thus, we can compute the expected number of spheres that would be created.

$$\text{Expected Spheres} = \left[\binom{n}{4} + \binom{n}{3} + 1\right] /2$$

If we do not first calculate the convex hull to reduce the point set, then the expected number of spheres becomes astronomical. For example, if n=400 points, then the expected number of spheres > 500,000,000.

The comparisons made using the expected spheres use $\{P_{ch}\}$ rather than $\{P\}$. For this reason, we do not include the calculation of the convex hull in our comparisons. It will be shown that the algorithm is highly sensitive to the input data. The data sets used are described in Chapter VII. A complete discussion of the results will be found in Appendix II.

CHAPTER VI

A. Hidden Line Algorithm

When this research was initiated, a major goal was to be able to display a simulation of one flying through a space containing a large number of objects. In addition, it was felt essential that the view presented not contain lines that should be hidden. Initially, we planned to use an existing algorithm since several "fast" hidden line algorithms had been published.[1,5,6]

To achieve some sense of motion, it was felt that a new picture needed to be displayed at least five times a second. Whereas the published algorithms would handle an arbitrary number of objects (none stated that they could not), a close examination indicsted that they were in some sense optimized for one complex object. This was due to the initialization required (details follow). No examples were found containing a large number of simple objects. Since the best example we found reported a time of five seconds for a 240 edge object on a CDC 6600,[6] we felt we needed to develop our own algorithm, optimized to fit our particular requirements. To facilitate design of a fast algorithm, the data base was defined to contain only objects which fall into three categories:

1) 3D convex bodies
2) 3D "wire frame" bodies
3) 2D planes (convex or concave)

In addition, every pair of objects was required to be linearly separable and non-intersecting. Non-convex objects can be constructed from convex ones. Clearly, every pair of non-intersecting convex objects can

be separated by a plane. Thus, these restrictions do not seriously limit the type of pictures that can be constructed.

It occurred to us that in a simulation of a city, as viewed from an airplane, most of the objects as presented on the two-dimensional screen would not occlude one another. This fact was not taken into account in other algorithms since they are concerned primarily with the hidden line removal for a single (or a few) complex object(s). As we will see, this observation enabled us to design an algorithm that works very fast for a data base such as we have described. It is difficult to make a fair comparison of our algorithm to other algorithms since our algorithm is designed to be highly sensitive to the relative positions of the objects in the two-dimensional projection.

B. Design Goals

The overriding concern was to be speed. All compromises that were made had that point in mind. Fundamentally, the algorithm proceeds by comparing every edge of every object to be displayed with each edge of every plane that can potentially hide the edge to be displayed. With n edges in the data picture, this gives a worst case of $\mathcal{O}(n^2)$ operations to be performed. The operation is a test for the intersection of two edges in three space as viewed from some fixed point. All hidden line algorithms have this in common.

This algorithm differs from its predecessors on one major point; the hidden line calculations are performed in the 2D projection plane rather than in 3D. This is made possible by the preprocessing of the depth information. The restriction of non-intersecting convex objects

allows depth information to be calculated on an object basis rathern than an edge or face basis, which amounts to a considerable saving in time. Timing studies showed the depth calculation for our algorithm to be insignificant.

C. Description of the Algorithm

The general philosophy was to use simple tests at the appropriate places to pare down the number of complex calculations. The tests must be carefully chosen. For example, suppose 100 tests were performed and 80 of these tests indicated that a calculation was to proceed. Unless the 100 tests involve less effort than the 20 calculations they save, it is far more expedient to simply perform the 100 calculations and forget the tests. We will show by experimental result that the tests we made offer a significant saving in calculation effort.

As was stated earlier, the major calculation performed is the intersection of two lines in a plane. In the worst case, $\mathcal{O}(n^2)$ of these calculations must be made where n is the number of edges. However, if the data base contains a large number of objects, most of the intersection calculations that we perform will not result in an intersection. They contribute no information to the display process (save the fact that a particular pair of lines does not intersect). We will reduce the number of intersection calculations that need be performed by a series of simple tests. By experiment, we will show that for the data bases we used, approximately 80% of the intersection calculations resulted in intersection.

D. Depth Calculation

Whereas the view of the objects continually changes, the relative position of the objects remain fixed. It is this static information

that we take advantage of to simplify the calculation of depth. The separating planes are used to form an anti-symmetric square matrix. Each of the n/2 unique elements of this matrix give the following information about each pair of objects (A,B):

1) A potentially hides B

2) B potentially hides A

3) A cannot hide B and B cannot hide A

The elements of this matrix are calculated once at the beginning of each frame and their calculation will be described below.

E. Separating Planes

Basic to this hidden line algorithm is the calculation of the intersection of the projection of two edges belonging to a pair of faces. It must then be determined which of the faces containing the edges are in front so the visible segment of the partially hidden edge can be displayed.

The separating planes are used to provide this depth information. This basic idea was used in the NASA raster display system built by General Electric and described in a report by Schumacher.[9] We used a modification of the separating plane technique to provide a fast calculation of depth. We felt that the restrictions imposed (linearly separable objects) were worth the benefit to be derived in speed.

The data base contains, in addition to the objects, equations of the planes that separate every pair of objects. In almost every case, the faces of the objects can serve as separating planes. The 2D example in Figure 6.1 will be used throughout this section.

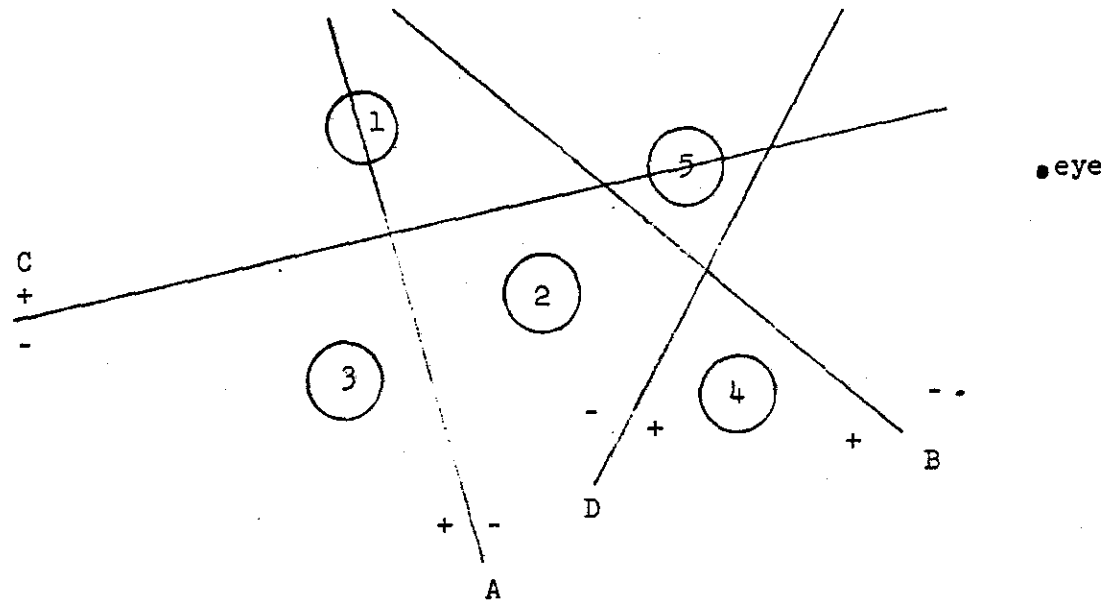


FIGURE 6.1

We can think of the depth information, as computed from the viewing position, as consisting of a static and a dynamic part. The static part consists of the relative positions of the objects {1,2,3,4,5} with respect to one another. The dynamic part is those relationships when observed from a particular point in space.

Each pair of objects is separated in space by a plane {A,B,C,D}. The equation of a plane is of the form

$$ax + by + cz + d = 0 \ .$$

The signed distance from any point (x_1,y_1,z_1) in space to that plane is

$$D = \pm \frac{ax_1 + by_1 + cz_1 + d}{\sqrt{a^2 + b^2 + c^2}}$$

The sign chosen is arbitrary and a matter of convention. Once the "sign" of the plane is noted, then the determination of which side of a

plane point (x_1, y_1, z_1) is situated on reduces to looking at sign (D_1) where

$$D_1 = ax_1 + by_1 + cz_1 + d$$

The separating plane matrix (SEPL) is a fixed part of the data base. It is created along with the data structure for the objects. This matrix and the equations of the separating plane coefficients are used to determine which objects can potentially hide one another.

i ↓ \ j →	A	B	C	D
1	∅	+	+	-
2	-	+	-	-
3	+	+	-	-
4	-	+	-	+
5	-	-	∅	-

SEPL - separating plane matrix

There are three possible entries to this matrix for object 1, plane j.

1. + object i is on + side of plane j
2. - object i is on - side of plane j
3. ∅ object i and plane j intersect

From the eye position shown in Figure 6.1, we determine on which sides of the planes this position is situated, and can "mark" the SEPL matrix accordingly.

	A	B	C	D
1	∅	0	0	0
2	1	0	1	0
3	0	0	1	0
4	1	0	1	1
5	1	1	∅	0

SEPL

The elements of column j (plane j) are marked 0 for minus and 1 for plus if the viewing position is on the plus side of plane j. The elements of column j are marked 1 for minus and 0 for plus if the viewing position is on the minus side of plane j. The rows can then be interpreted as follows:

1. 1 object i is on the same side of plane j as the viewing position
2. 0 object i is on the opposite side of plane j from the viewing position
3. ∅ object i and plane j intersect

We are now ready to compute the separating object matrix SO.

$\longrightarrow m$

$\ell \downarrow$

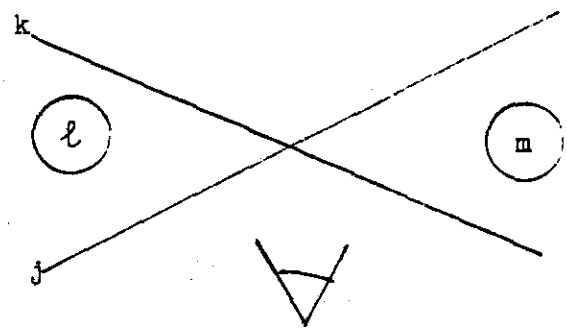
	1	2	3	4	5
1		-1	-1	-1	-1
2	1		1	-1	-1
3	1	-1		-1	-1
4	1	1	1		0
5	1	1	1	0	

SO Matrix

The matrix is anti-symmetric ($SO(\ell,m) = -SO(m,\ell)$). The entry $SO(\ell,m)$ is interpreted as follows:

1. -1 object ℓ is potentially hidden by object m
2. 1 object ℓ potentially hides object m
3. 0 objects ℓ and m cannot hide one another

The entry $SO(\ell,m)$ is computed from the marked SEPL matrix by comparing rows ℓ and m of SEPL. If for column j of those rows, $SEPL(\ell,j) = 1$ and $SEPL(m,j) = 0$, then object ℓ and the viewing position are on the same side of plane j and thus, potentially, hide object m. If $SEPL(\ell,j) = SEPL(m,j)$ or either of these entries $= \emptyset$, then that plane contributes no depth information for these objects. If there exist two planes for which $SEPL(\ell,j) = 1$, $SEPL(m,j) = 0$ and $SEPL(m,k) = 1$, $SEPL(\ell,k) = 0$, then a situation like Figure 6.2 exists and the objects cannot hide one another.



Viewing Position

FIGURE 6.2

For fast implementation purposes, SEPL is actually two single bit matrices, SEPL1 and SEPL2, with each object occupying a row and one bit allocated for each plane. SEPL1 is identical to SEPL except $\emptyset = 0$; thus, only one bit per entry. SEPL2 is the same size as SEPL1 with 1's everywhere except a 0 where SEPL contained a $\emptyset$.

For example compute SO(1,2)

$R_1 = 0000$

$R_2 = 1010$

$S_1 = 0111$

$S_2 = 1111$

$M = 0111$

$S = (1010) \cap (0111) = 0010$

$RR_1 = 0000$

$RR_2 = 0010$

$RR_2 > 0 \Rightarrow SO(2,1) = -SO(1,2) = 1$

This pairwise calculation is combinatorial, involving non-sequential logic, and could be implemented easily in hardware.

SEPL2

	A	B	C	D
1	0	1	1	1
2	1	1	1	1
3	1	1	1	1
4	1	1	1	1
5	1	1	0	1

SEPL1

	A	B	C	D
1	0	0	0	0
2	1	0	1	0
3	0	0	1	0
4	1	0	1	1
5	1	1	0	0

Let R_ℓ = row ℓ of SEPL1

R_m = row m of SEPL1

S_ℓ = row ℓ of SEPL2

S_m = row m of SEPL2

$M \leftarrow S_\ell \cap S_m$

$S \leftarrow (R_\ell \text{ XoR } R_m) \cap M$

$RR_\ell \leftarrow R_\ell \cap S$

$RR_m \leftarrow R_m \cap S$

If $RR_\ell > 0$ then $SO(\ell,m) = -SO(m,\ell) = 1$

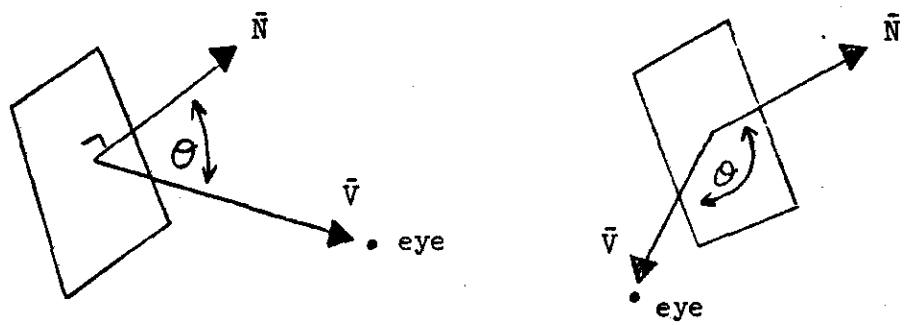
If $RR_m > 0$ then $SO(\ell,m) = -SO(m,\ell) = 1$

If $(RR_\ell > 0) \cap (RR_m > 0)$ then $SO(\ell,m) = -SO(m,\ell) = 0$

F. Description of the Algorithm

Step 1: Compute the SO matrix. This matrix provides the necessary depth information to allow the hidden line calculations to proceed in two dimensions. Its calculation was described in Section E.

Step 2: Classify the edges. A three-dimensional convex object has the property that each face is either visible or invisible. No part of the object can occlude itself. A face is said to be visible if the dot product of a vector ($\bar{V}$) from the face to the eye position, and an outward facing normal from the plane is positive (Figure 6.3).



Visible Face
$\bar{V} \cdot \bar{N} > 0$

Invisible Face
$\bar{V} \cdot \bar{N} \leq 0$

FIGURE 6.3

A face is invisible if $\bar{V} \cdot \bar{N} \leq 0$

A) BACK EDGE - a common edge between two invisible faces

B) FRONT EDGE - a common edge between two visible faces

C) EXTREME EDGE - a common edge between a visible and an invisible face.

Back edges of three-dimensional convex bodies do not enter into the display processing in any way.

Step 3: Projection of edges. All of the front and extreme edges are projected into two dimensions using perspective projection as described in Chapter II. From this point on, all calculations are performed in two dimensions.

Step 4: Object windows. For each object, the maximum and minimum in X and Y for the projected object is saved. These values define a region on the screen occupied by the object. Let W_i be the window for object O_i.

Step 5: Invisibility calculation. This step is performed once for each object O_i to be displayed. Let V_i be any vertex on O_i. The purpose is to calculate the number of objects that hide V_i. This quantity is called the invisibility I_i of V_i. Object O_j hides V_i if, in the two-dimensional plane, V_i lies inside the projection of O_j. The two-dimensional projection of a convex object is a closed convex polygon. Since we allowed concave planes as objects, this calculation must allow for the general case of covering a point by a closed concave figure. The calculation is to first construct a semi-infinite ray from V_i to some point off screen. Then, if the number of intersections that the semi-infinite ray makes with the extreme edges of O_j are even, O_j does not hide V_i. If the number is odd, then O_j hides V_i (see Figure 6.4).

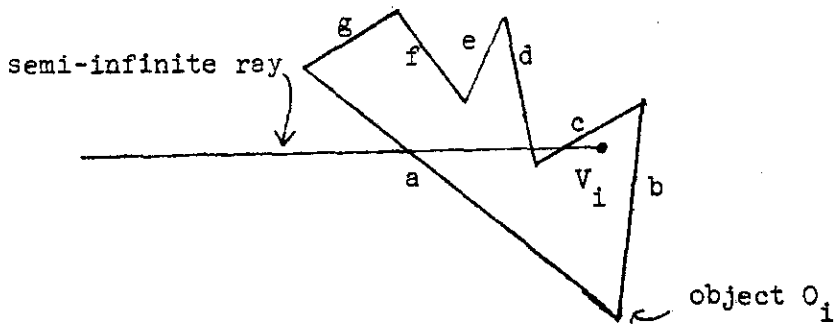


FIGURE 6.4

Several tests were made to speed up this calculation.

1. Only objects that potentially hide O_i are considered. We have this information from SO.

2. Only those objects whose window W_j covers V_i are considered.

3. Finally, only those extreme edges that have at least one vertex on or below the semi-infinite ray are considered for an intersection calculation. This reduces the number of intersection calculations that must be made. In Figure 6.4, only extreme edges (a,b,c,d) are tested for intersection.

Once the invisibility of V_i is known, O_i can be processed for display.

<u>Step 6:</u> Hidden line processing. This step is performed for each front and extreme edge for all objects to be displayed.

A) An edge E_i of O_i, for which the invisibility of vertex V_i is known, is selected for display. This edge is then tested for intersection with all objects

that can potentially hide it. In a manner similar to the invisibility calculation, several tests are made to pare down the number of intersection calculations that must be made.

1. Only objects that potentially hide O_i are considered. We have this information from SO.
2. Only those objects O_j whose window W_j intersects E_i are considered.
3. An intersection is computed for only those extreme edges E_j that lie in the rectangle R_i defined by E_i.

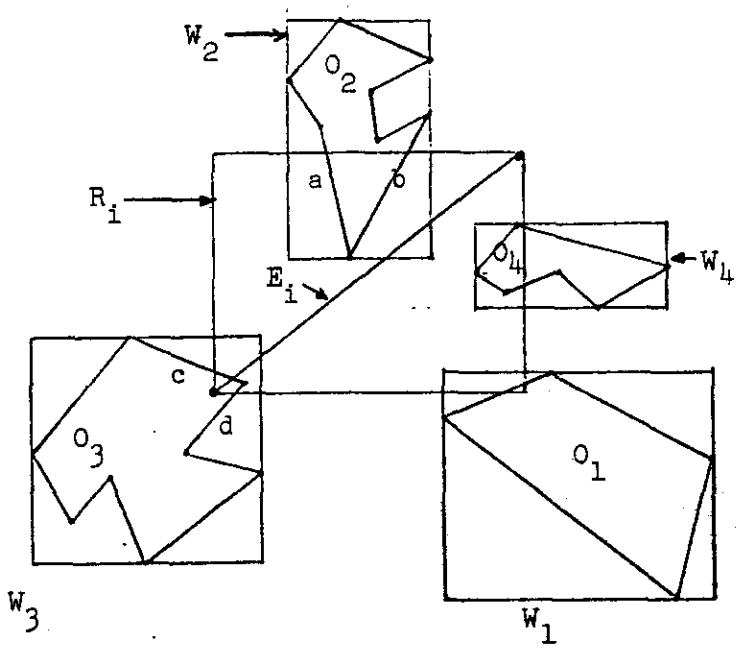


FIGURE 6.5

Consider the example in Figure 6.5. Assume O_1, O_2, O_3, O_4 potentially hide E_i. This is determined from SO. Only W_2 and W_3 intersect E_i. Thus, W_1 and W_4 are not considered further. Of all the extreme edges in O_2 and O_3, only edges E_a and E_b in O_2, and edges E_c and E_d in

O_3 intersect R_i. Thus, only four intersection calculations are made out of a possible 25.

B) All intersections are placed on an intersection list along with a flag to tell whether or not the segment (from initial vertex V_i to intersection) is going behind or coming out from the object under test. After testing against all candidate extreme edges, the intersection list is sorted. Since the number of surfaces hiding the initial vertex is known, each intersection can change the invisibility by, at most, 1. When the invisibility becomes 0 (no planes hiding), then that segment is displayed.

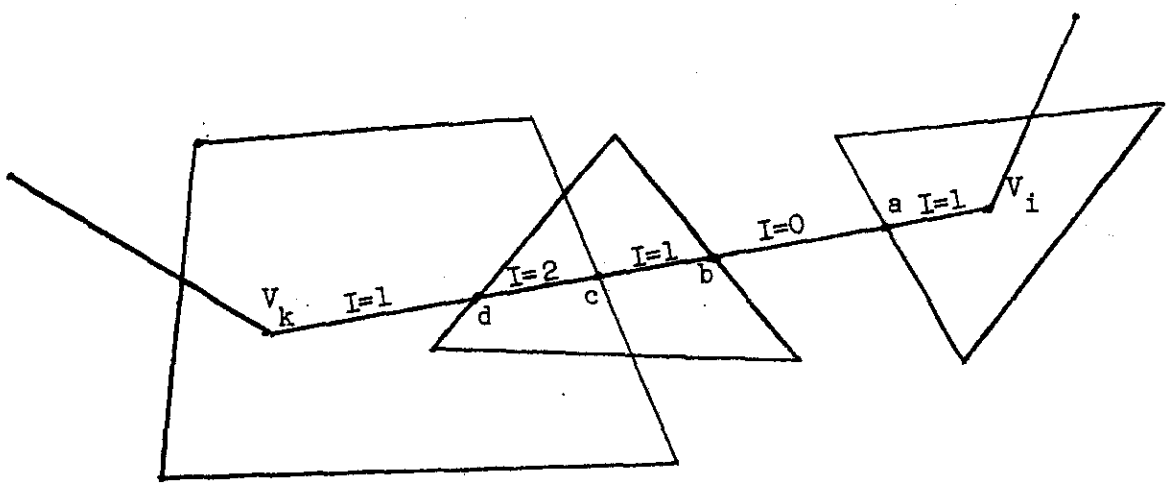


FIGURE 6.6

For example, assume the invisibility of $V_i = 1$. The intersection list would contain four intersections corresponding to a,b,c,d in Figure 6.6. The flag stored with each intersection is +1 if the segment from V_i to the intersection is going behind a face, -1 if it is coming out

from a face. In the example, a = -1, b,c = +1, d = -1. After the intersection list is sorted, each segment is examined for display. At point a, the invisibility is decreased by 1 to 0 so at that point the edge becomes visible. At the next intersection, b = 1, changing the invisibility to 1, and the edge again becomes invisible. Continuing on, the invisibility at c is increased to 2. At d, it is decreased to 1. Since there are no more intersections, the invisibility of V_k is set to 1. All edges with vertex V_k can now be processed.

G. Computational Effort

The most time consuming calculation performed in the inner loop is the intersection of two edges. We chose to do this using the parametric equations of a line. Consider the edge E_i with endpoints (x_1,y_1) and (x_2,y_2). Then we have

$$x = x_1 + (x_2-x_1)t$$

$$y = y_1 + (y_2-y_1)t$$

For $0 \leq t \leq 1$, x and y fall on the line segment (see Figure 6.7).

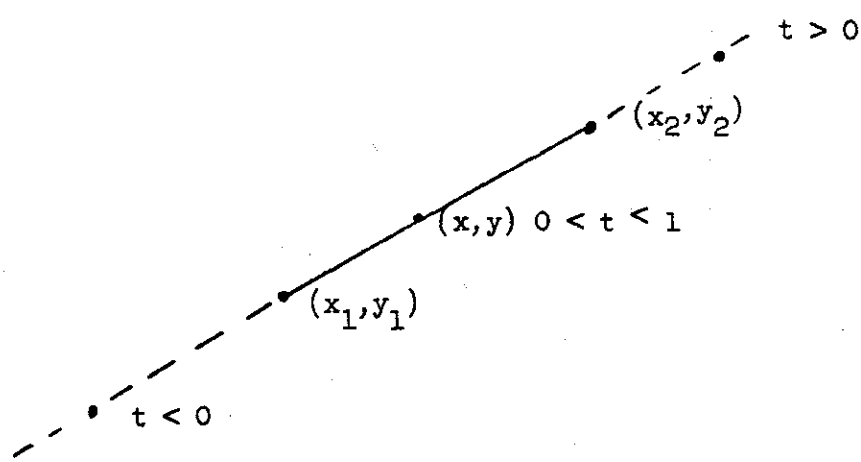


FIGURE 6.7

We compute the intersection of two lines by solving for t_1 and t_2 for each of the lines (see Figure 6.8).

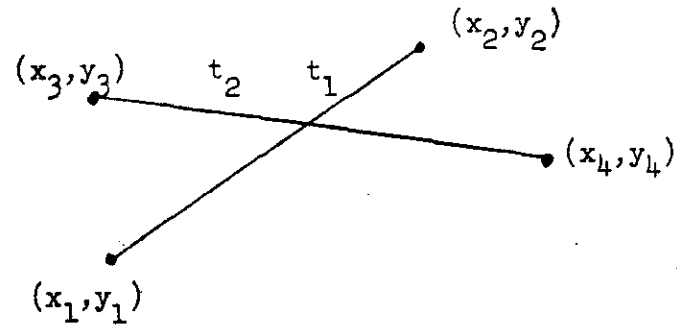


FIGURE 6.8

$$x_1 + (x_2 - x_1)t_1 = x_3 + (x_4 - x_3)t_2$$

$$y_1 + (y_2 - y_1)t_1 = y_3 + (y_4 - y_3)t_2$$

$$t_1 = \frac{(x_3 - x_1)(y_3 - y_4) - (x_3 - x_4)(y_3 - y_1)}{(x_2 - x_1)(y_3 - y_4) - (x_3 - x_4)(y_2 - y_1)}$$

$$t_2 = \frac{(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)}{(x_2 - x_1)(y_3 - y_4) - (x_3 - x_4)(y_2 - y_1)}$$

The two lines intersect only if $0 < t_1 < 1$ and $0 < t_2 < 1$. This calculation is easier than using equations of the form $y = mx+b$ because special cases of infinite slope do not have to be considered.

The test of intersection of an edge with a rectangular window consists of a maximum of 8 comparisons. The maximum is reached in the case where the edge intersects the rectangular window. In any case, this test is used to eliminate many potential intersection calculations.

H. Experimental Results

Our goal was to determine the speed of the algorithm as a function of the complexity of the data. We have an upper bound in that the number

of intersections to be computed is $\mathcal{O}(n^2)$ where n is the number of edges in the picture. Our attempt at increasing complexity was to generate a series of pictures containing unit cubes spaced at intervals of two units.

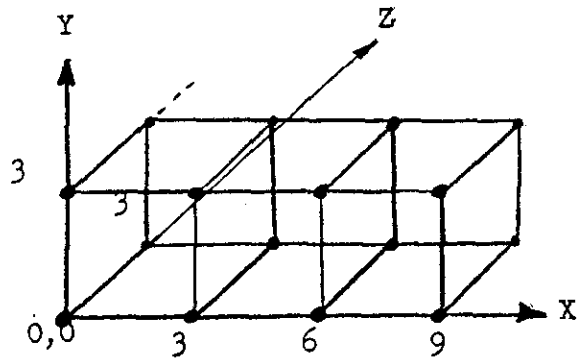


FIGURE 6.9

The data sets contain 4,8,16,24,32 and 48 cubes. These cubes are one unit on a side and centered on the grid as shown in Figure 6.9. They are two units apart in each dimension. The location of the cubes is as follows:

A) 4 cubes - (0,0,0), (3,0,0), (3,0,3), (0,0,3)

B) 8 cubes - same as four cubes with additional set at y = 3

C) 16 cubes - this is the configuration shown in Figure 6.9

D) 24 cubes - same as 16 cubes with 8 additional cubes at z = 6

E) 32 cubes - same as 24 cubes with 8 additional cubes at z = 9

D) 48 cubes - same as 24 cubes with an additional 24 cubes at y = 6.9. This is an array of cubes, four on a side.

In addition, two other data sets were tested. The first consisted of 16 rectangular prisms arranged in a "city-like" structure (see Figure 6.10, a-d). This was used as a representation of a "typical" picture. The second data set consisted of 32 non-intersecting cubes with random positions relative to the lattice. The experimental results are discussed in Appendix III.

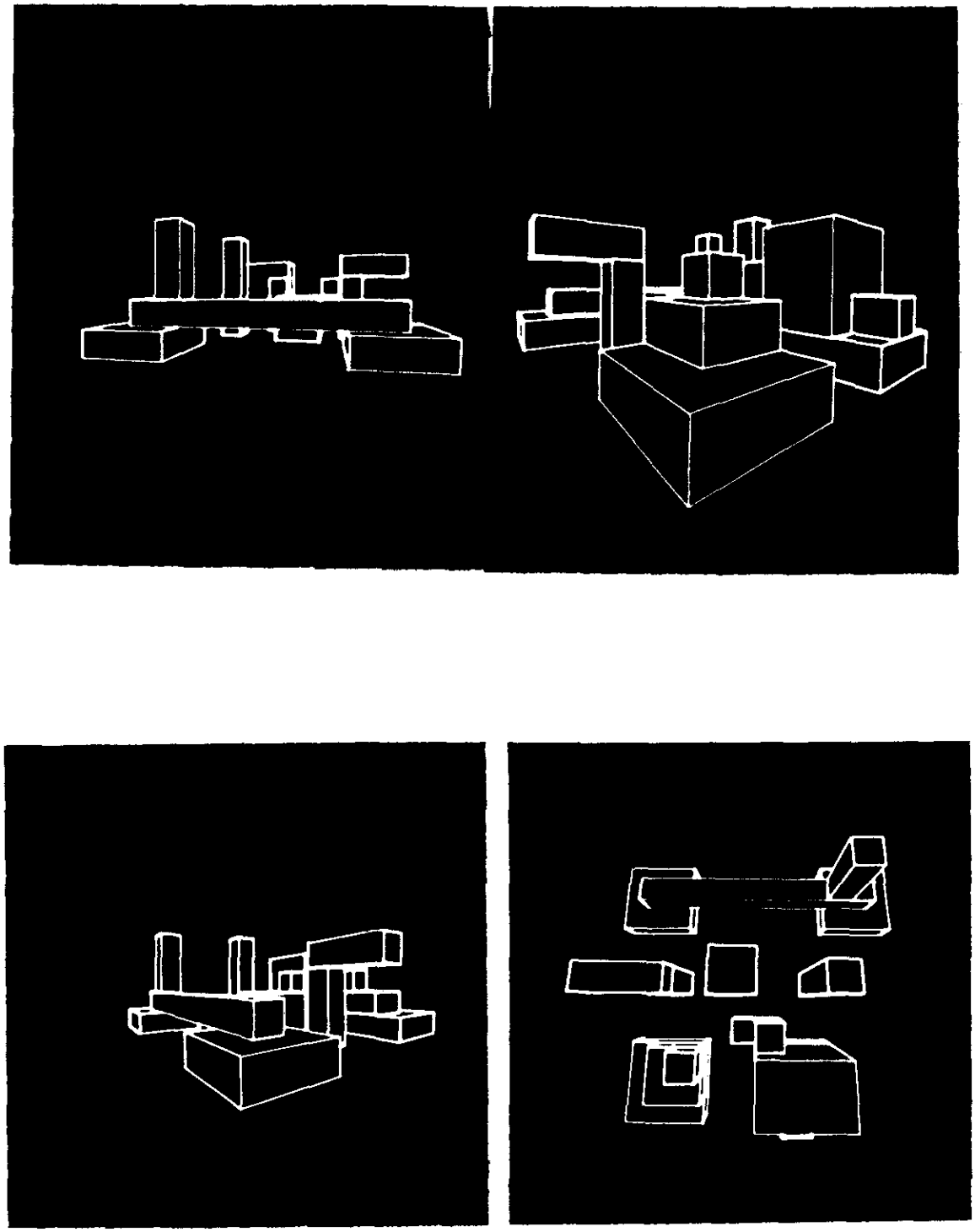


FIGURE 6.10, a-d

Four Views of a "City-like" Structure. Average calculation and display time = 130 milliseconds

CHAPTER VII

A. Convex Hull and Minimum Sphere Data

To facilitate a test of the algorithms, we chose eight types of data on which to test the algorithms. The data consisted of random numbers in uniform and normal distributions, as detailed below.

UNIFORM DISTRIBUTIONS - values generated from a uniform random number generator

A) Ellipse

Uniform random numbers are generated three at a time for x, y, and z. They are generated in the interval

$$-2 < x,y,z < 2$$

only those numbers satisfaying the following equation are used

$$\frac{X^2}{4} + Y^2 + Z^2 \leq 1$$

B) Polyhedron

Uniform random numbers are generated on the interval

$$0 < x,y,z < 2$$

Only those satisfying the following inequality are used

$$0 < Y < 2$$

$$0 < Y < 1$$

$$0 < Z < 1$$

NORMAL DISTRIBUTIONS - value for X,Y,Z generated from the following:

Let R_1 and R_2 be two random numbers $0 < r < 1$ from a uniform random number generator. Two normally distributed random numbers N_1 and N_2 are generated from R_1 and R_2 as follows

$$T_1 = (-2 \ln (R_1))^{1/2}$$

$$T_2 = 2\pi * R_2$$

$$N_1 = T_1 \text{ Sin } (T_2)$$

$$N_2 = T_1 \text{ Cos } (T_2)$$

Since each of the normal random numbers can range from $-\infty$ to $+\infty$, we accept only those within a limited range. Accepting only those points that lie on the interval $-n\sigma < r_N < n\sigma$ gives a set of points whose distribution is normal with a standard deviation of $n\sigma$. Let X_N, Y_N, Z_N be normal random numbers.

C) Normal Cube

$$-n\sigma < X_N < n\sigma \qquad -n\sigma < X_N < n\sigma$$

$$-n\sigma < Y_N < n\sigma \qquad -n\sigma < Y_N < n\sigma$$

$$-n\sigma < Z_N < n\sigma \qquad -n\sigma < Z_N < n\sigma$$

These data sets were generated for $n\sigma = 1,2,4$. They will be referred to as cube $\sigma = n$ $(n=1,2,4)$.

D) Normal Sphere

$$X_N^2 + Y_N^2 + Z_N^2 \leq (n\sigma)^2$$

These data were generated for $n\sigma = 1,2,4$. They will be referred to as sphere $\sigma = n$ $(n=1,2,4)$.

Using these distributions, a number of sets of data were prepared. 500 cases were run for each of the data sets. The number of points in each data set is given below.

	25	50	100	200	400	800	1600	3200
Cube $\sigma = 1$	x	x	x	x	x	x	x	x
Cube $\sigma = 2$	x	x	x	x	x	x	x	x
Cube $\sigma = 4$	x	x	x	x	x	x	x	x
Polyhedron	x	x	x	x	x	x	x	
Sphere $\sigma = 1$	x	x	x	x	x	x		
Sphere $\sigma = 2$	x	x	x	x	x	x		
Sphere $\sigma = 4$	x	x	x	x	x	x		
Ellipse	X	x	x	x	x	x	x	

Data sets were chosen to provide typical as well as worst case tests for the calculation of the minimum sphere. The distributions where $n\sigma = 4$ were felt to represent the points likely to be encountered for arbitrary groupings of objects.

The spherically distributed data is a worst case test for the minimum volume sphere algorithm because it tends to neutralize the benefit of the heuristic. The solid angles subtended by the points on the convex hull do not have the wide distribution enjoyed by the cubical or elliptical data.

The worst case test for the convex hull consists of a set of points, all of which lie on the convex hull. Since we can estimate the performance $\mathcal{O}(n^2)$ of the algorithm, no data sets were generated to test this case. In addition, this case is unlikely for the types of data this thesis is concerned with.

CHAPTER VIII

Conclusions

We developed a graphic data structure that allows for fast access to an arbitrarily large data base. The data is organized in two ways. First, the objects can be clustered in successively larger groups to facilitate fast retrieval or elimination from consideration. These groupings represent the "natural" clustering of the objects rather than an arbitrary partitioning of the space containing the objects. This structure allows for easy modification of objects in the data base.

Embodied in the same structure is a facility for multiple description of objects. This can be used as a means of retrieving only the correct amount of detail that can be resolved on the display, assuring minimum display processing. For example, if a distant view of a data base containing a large number of objects would appear as a point, then only a single point would be retrieved and displayed. The structure allows the system to only "know" about objects that are both in the field of view and the proper amount of detail for each object.

To support this data organization, an algorithm was designed to construct a minimum volume sphere around a set of points. The spheres were used to group and isolate objects. The algorithm proceeds by taking the points three and four at a time and generating spheres until the smallest is found. A heuristic is used to generate the minimum volume sphere early in the search. The result is that the minimum volume sphere is generated with up to seven orders of magnitude less calculations than would be expected without the use of the heuristic.

To implement the heuristic, an algorithm was designed to compute the convex hull of a set of points. For the data sets we tested, this algorithm requires about an order of magnitude less calculation than the only other algorithm we found. For the worst case (all points in the data set are on the convex hull), the algorithms are roughly equivalent.

Finally, a new hidden line algorithm was developed. By restricting the type of objects to linearly separable three-dimensional convex objects, two-dimensional concave planes and three-dimensional wire frame objects, a very fast algorithm was produced. It was written in PL/1 (non-optimizing compiler) and run on a 360/91 to which an IDIIOM interactive display was attached. A complex picture consisting of 16 objects (192 edges) took 130 milliseconds (see Figures 6.10 a-d).

Future Research

This research suggested a number of areas that should be investigated further. The first is the development of a procedure to generate groupings that are in some sense optimum. In concert with this procedure would be an algorithm to generate (without user specification) the required multilevel descriptions.

The convex hull algorithm described in Chapter IV was actually implemented in n-dimensions. Studies of the algorithm's behavior as a function of the dimensionality of the space would be most useful.

The minimum volume sphere algorithm needs much refinement. A full examination of an iterative approach would be valuable.

Finally and most important, a great deal of work needs to be done on the problem of displaying simple relationships among variables. It

is so easy to plot a graph that we quickly lose sight of the reason we are plotting the data to begin with, namely, to convey information. We spent an inordinate amount of time trying to develop a procedure that would examine data and automatically choose the proper type of scaling that would convey the greatest amount of information. Work toward this goal would do a great deal to enhance the stature of computer graphics.

BIBLIOGRAPHY

1. Appel, A., "The notion of quantitative invisibility and the machine rendering of solids," Proc. ACM National Conference (1967), pp 387-393.

2. Chand, D.R. and Kapur, S.S., "An algorithm for convex polytopes," Journal of the ACM, Vol. 17, No. 1 (1970).

3. Ecker, J.G., "Analytical derivation of statistical distributions of parameters of minimal sphere coverings," Memorandum Report No. 1672, Ballistic Research Laboratories, Aberdeen Proving Ground, Maryland (1965).

4. Fisherkeller, M.A., "The SLAC graphic Interpretation facility," Stanford Linear Accelerator Center, Computation Group Report CGTM No. 120 (March 1971).

5. Galimberti, R. and Montarari, V., "An algorithm for hidden-line elimination," Comm. ACM, 12,4, (April 1969), p. 206.

6. Loutrel, P., "A solution to the hidden-line problem for computer drawn polyhedra," IEEE Trans. Computers, C-19,3. (March 1970), p. 205.

7. Masaitis, C., "Minimal sphere covering," ARO-D Report 65-1, Proc. ARO Working Group on Computers (1964).

8. Masaitis, C., private communication.

9. Schumacker, R.A., Brand, B., Gilliland, M. and Sharp, W., "Study for applying computer-generated images to visual simulation," AFHRL-TR-69-14, U.S. Air Force Human Resources Laboratory, (Sept. 1969).

10. Warnock, J.E., "A hidden surface algorithm for computer-generated halftone pictures," Computer Science Dept., University of Michigan, TR 4-15, (June 1969).

11. Williams, R., "A survey of data structures for computer generated graphics systems," Computing Surveys, Vol. 3, No. 1, (March 1971).

APPENDIX I

Convex Hull Data

For the convex hull algorithm, the following quantities were recorded.

A. Number of original data points (NPTS). All graphs are plotted versus this quantity.

B. Number of points on the convex hull (NPTH). For a given number of points in the data set, this is the number of points on the convex hull.

C. Number of Tests (NT). A test is defined as the calculation of the signed distance $D_1 = ax + by + cz + d$. This is the major calculation performed by the convex hull algorithm.

D. Number of Equivalent Tests (NEQ). For the Chand-Kapur algorithm we showed that the major calculation is equivalent to two tests, as defined in C. From Chapter IV, we have that

$$NEQ = 3 \cdot NPTS * NPH$$

where NPH = number of planes on the convex hull. NPH was not plotted since it is proportional to NPTH. From the construction of the convex hull, $NPH = 2 \cdot NPTH - 4$.

The results are displayed in two groups. Group 1 contains cube $\sigma = 1$, cube $\sigma = 2$, cube $\sigma = 4$, and the polyhedron. Group 2 contains sphere $\sigma = 1$, sphere $\sigma = 2$, sphere $\sigma = 4$ and ellipse. The same data is plotted for each group. The data from group 1 is plotted followed by the data for group 2.

Summary Graphs

Each graph contains a plot of the average value of the variable versus the number of original data points for each of the four cases in

the group. Each curve is uniquely labeled. An attempt to provide an indication of how the data varied with the points in the data set is given in the upper left corner of each graph. A straight line least squares fit of the $\log_{10}$ of the last four points was made.

$$\text{Let } y' = mx' + b$$

$$\text{where } y' = \log_{10}(y)$$

$$x' = \log_{10}(x)$$

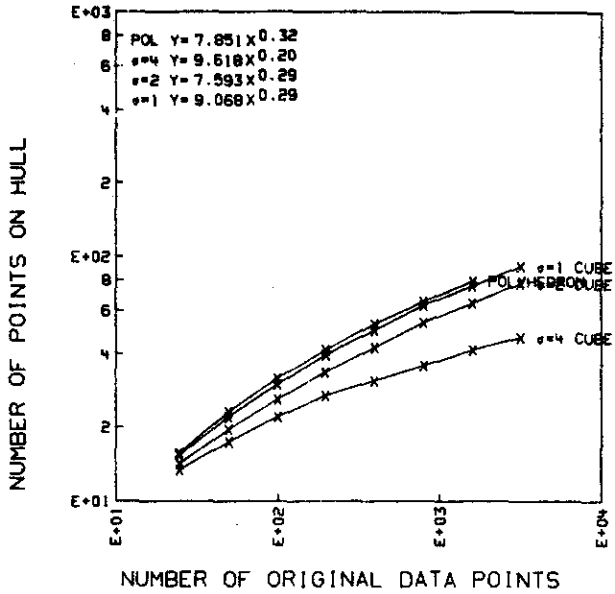
$$\text{then } y = 10^b x^m$$

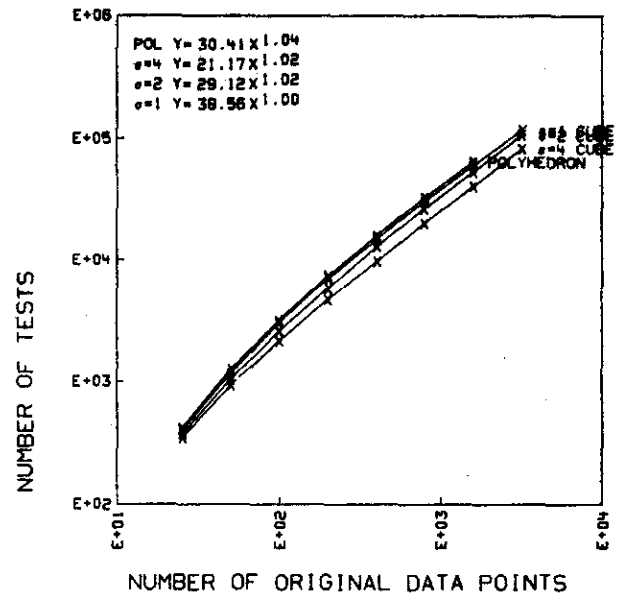
For example, consider the graph of NPTH for the group 1 data (cubes). The $\sigma = 1$ cube data varies like $(\text{NPTS})^{.29}$ while the $\sigma = 4$ cube data varies like $(\text{NPTS})^{.20}$. Note that this is the limiting trend because the fit is made with the last four points on the curve.

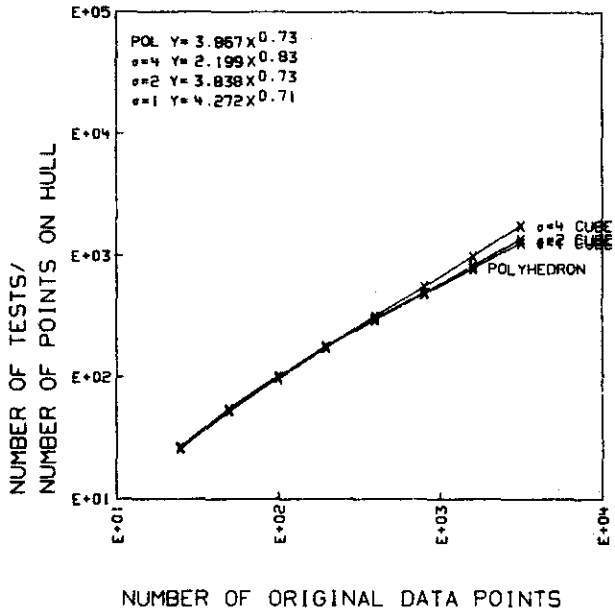
Since the points in the data set are generated from a random number generator, the number of points on the convex hull is not constant but normally distributed. It was decided that NT/NPTH and NEQ/NPTH would provide a better measure of the amount of work performed by the algorithm.

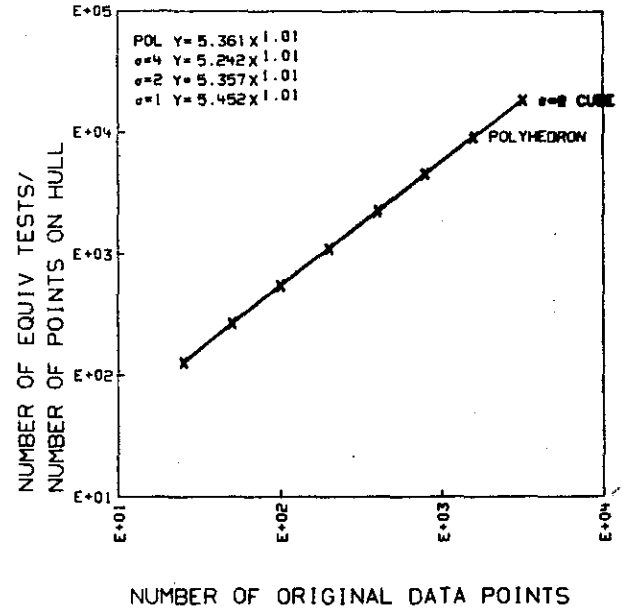
From these plots we note that on the data sets we tested, the new algorithm varies like $\approx(\text{NPTS})^{.75}$ while the Chand-Kapur algorithm varies like (NPTS). In addition, NT/NPTH is approximately an order of magnitude smaller than NEQ/NPTH. This is true for both groups of data.

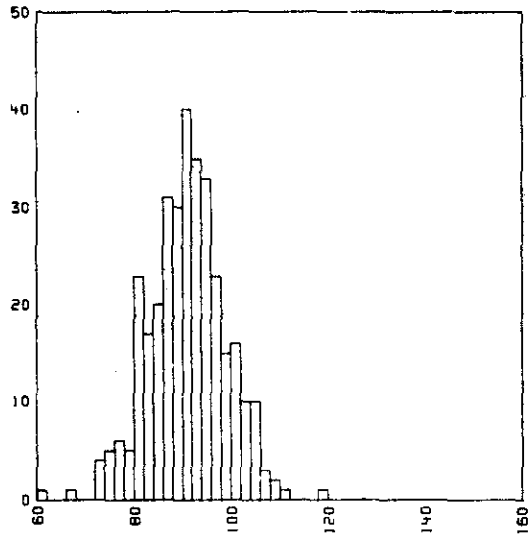
Following each of the summary sheets are histograms for the maximum number of points for each of the cases. This plot shows the normally distributed nature of the data as well as the range of each of the variables.


POL Y= 7.851 X^0.32
σ=4 Y= 9.618X^0.20
σ=2 Y= 7.593X^0.29
σ=1 Y= 9.068X^0.29
σ=1 CUBE
POLYHEDRON
σ=4 CUBE
NUMBER OF POINTS ON HULL
NUMBER OF ORIGINAL DATA POINTS

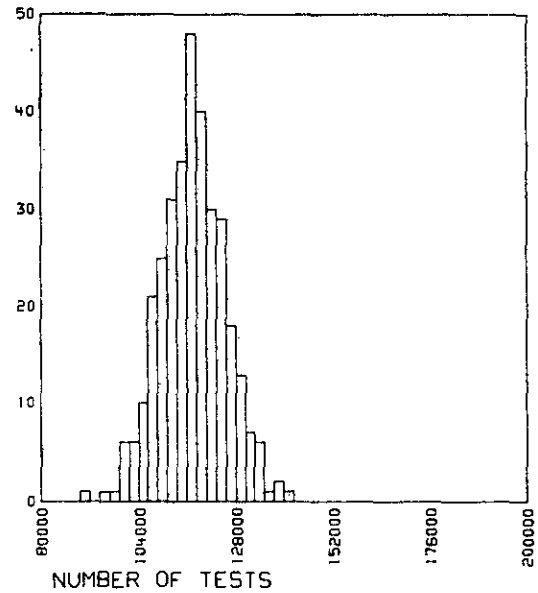

POL Y= 30.41 X^1.04
σ=4 Y= 21.17X^1.02
σ=2 Y= 29.12X^1.02
σ=1 Y= 38.56X^1.00
σ=4 CUBE
POLYHEDRON
NUMBER OF TESTS
NUMBER OF ORIGINAL DATA POINTS


POL Y= 3.867X^0.73
σ=4 Y= 2.199X^0.83
σ=2 Y= 3.838X^0.73
σ=1 Y= 4.272X^0.71
σ=4 CUBE
POLYHEDRON
NUMBER OF TESTS/
NUMBER OF POINTS ON HULL
NUMBER OF ORIGINAL DATA POINTS

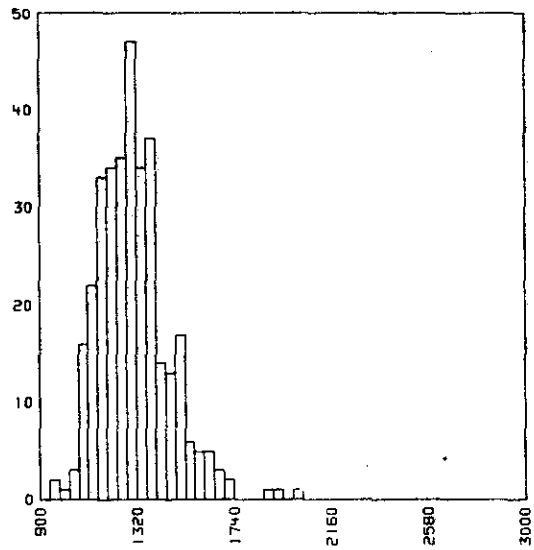

POL Y= 5.361 X^1.01
σ=4 Y= 5.242X^1.01
σ=2 Y= 5.357X^1.01
σ=1 Y= 5.452X^1.01
POLYHEDRON
NUMBER OF EQUIV TESTS/
NUMBER OF POINTS ON HULL
NUMBER OF ORIGINAL DATA POINTS



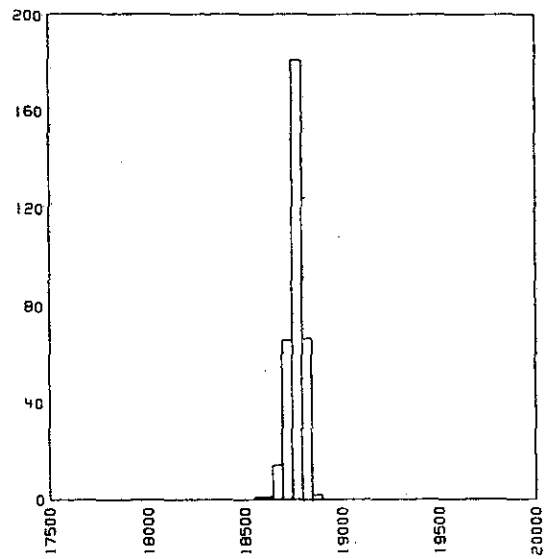
NUMBER OF POINTS ON HULL



NUMBER OF TESTS

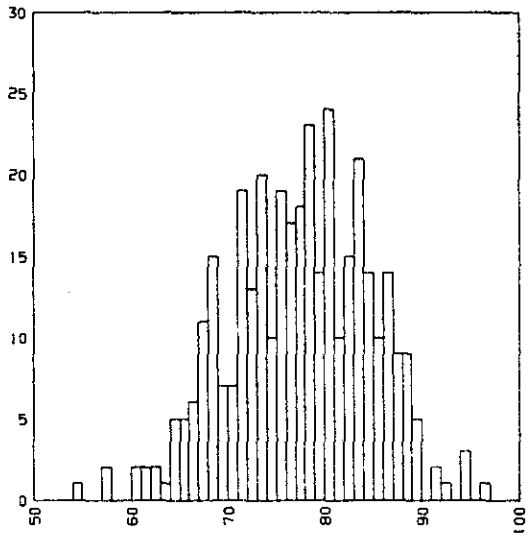


NUMBER OF TESTS/
NUMBER OF POINTS ON HULL

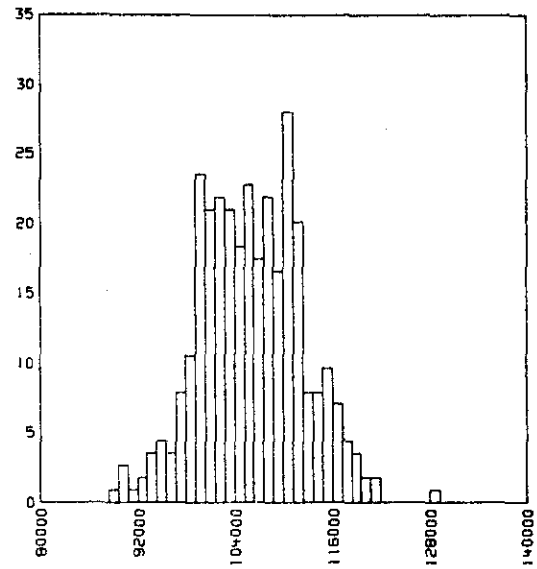


NUMBER OF EQUIV TESTS/
NUMBER OF POINTS ON HULL

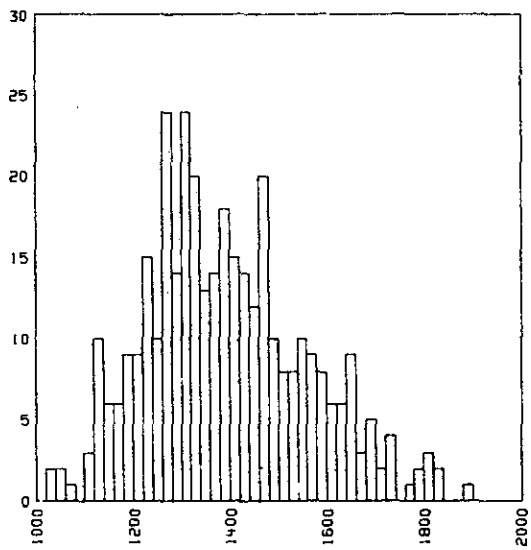
3200 POINTS CUBE $\sigma = 1$



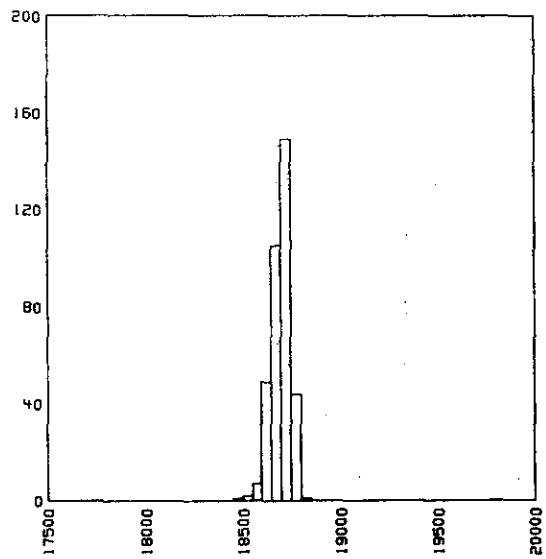
NUMBER OF POINTS ON HULL



NUMBER OF TESTS

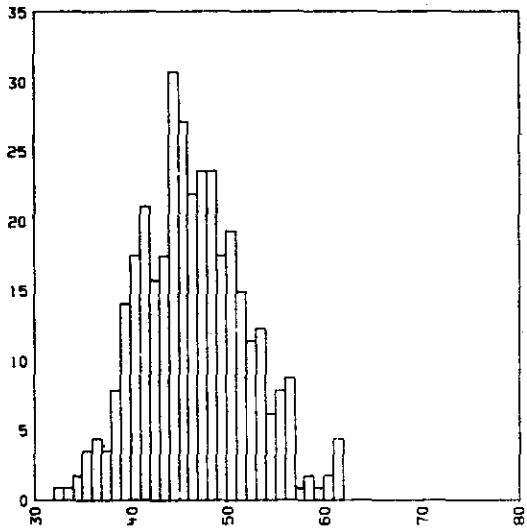


NUMBER OF TESTS/
NUMBER OF POINTS ON HULL

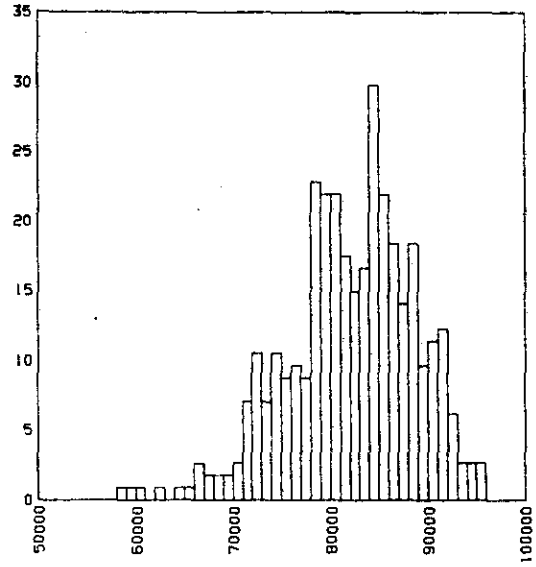


NUMBER OF EQUIV TESTS/
NUMBER OF POINTS ON HULL

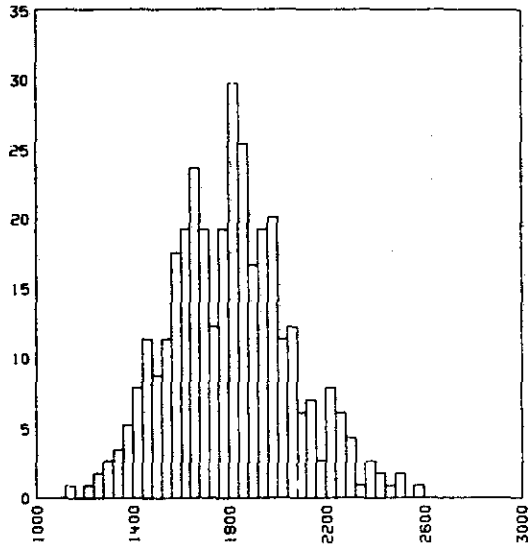
3200 POINTS CUBE $\sigma=2$



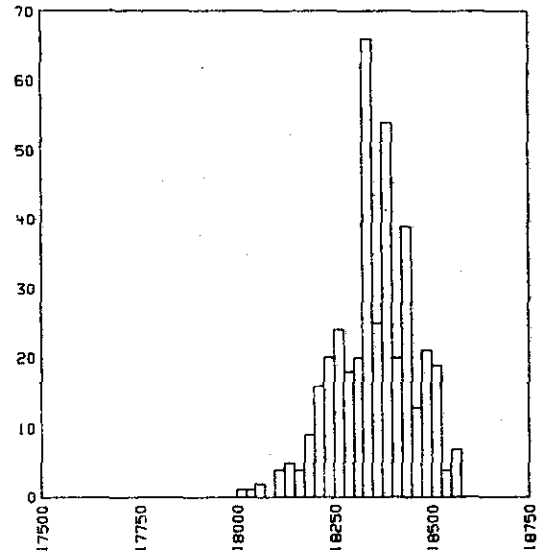
NUMBER OF POINTS ON HULL



NUMBER OF TESTS

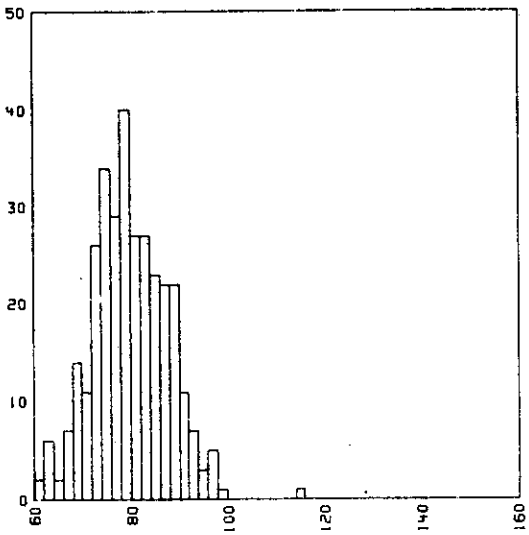


NUMBER OF TESTS/
NUMBER OF POINTS ON HULL

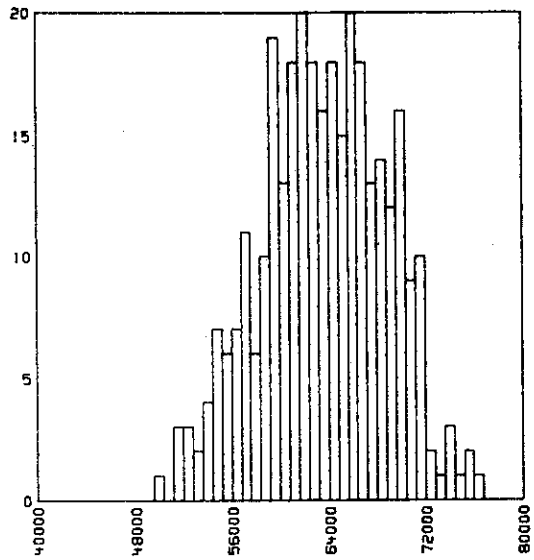


NUMBER OF EQUIV TESTS/
NUMBER OF POINTS ON HULL

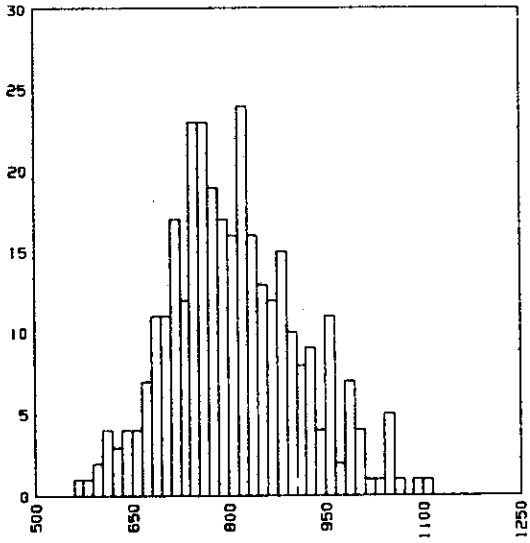
3200 POINTS CUBE $\sigma=4$



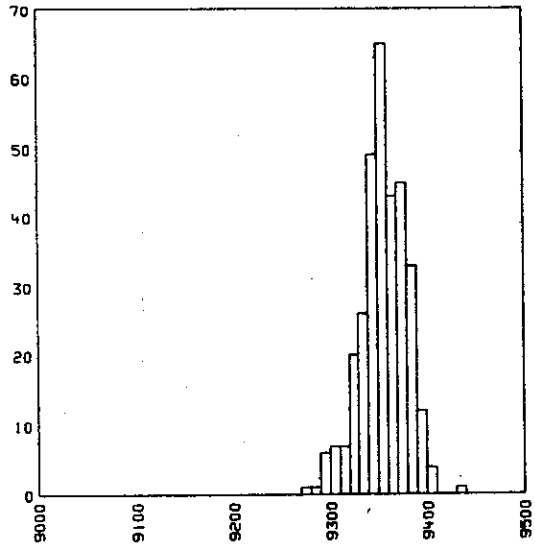
NUMBER OF POINTS ON HULL



NUMBER OF TESTS

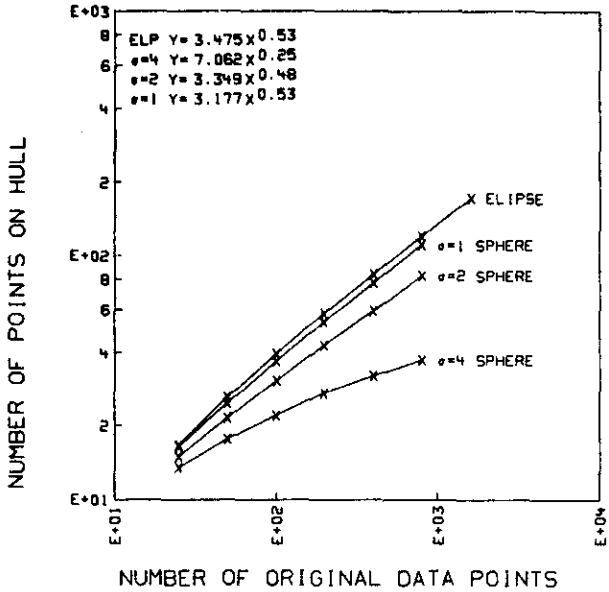


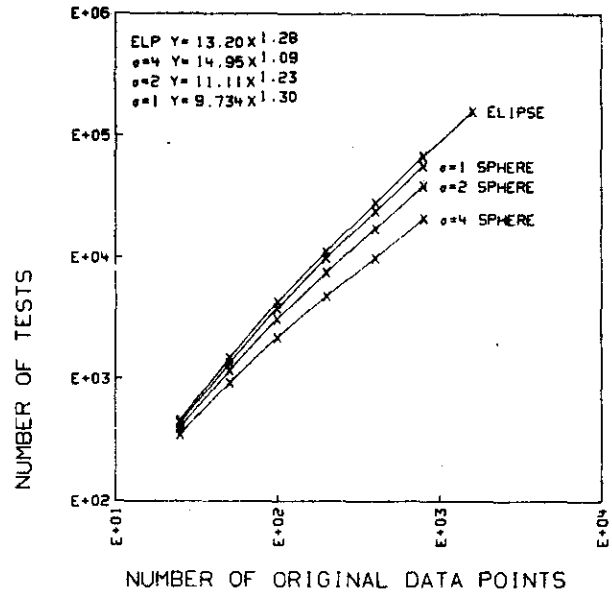
NUMBER OF TESTS/
NUMBER OF POINTS ON HULL

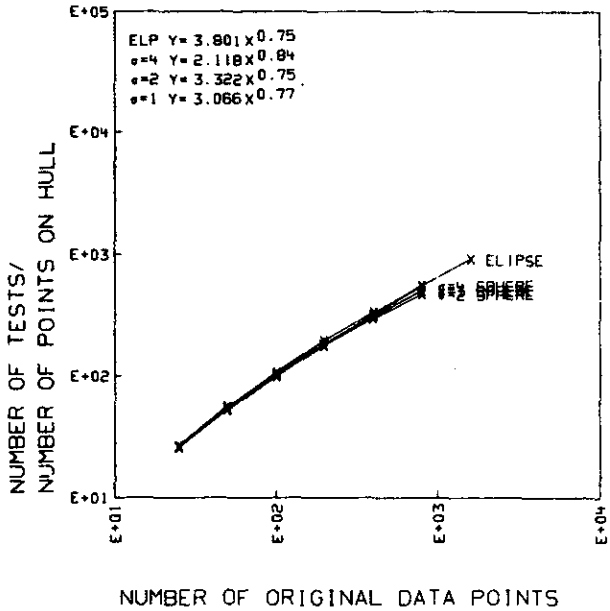


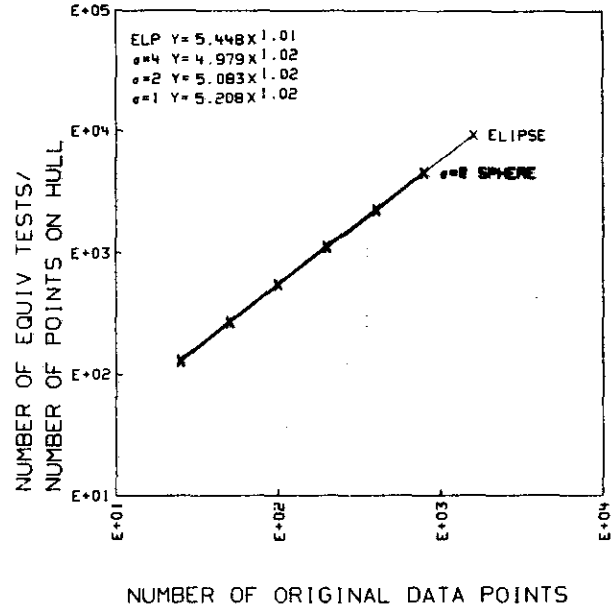
NUMBER OF EQUIV TESTS/
NUMBER OF POINTS ON HULL

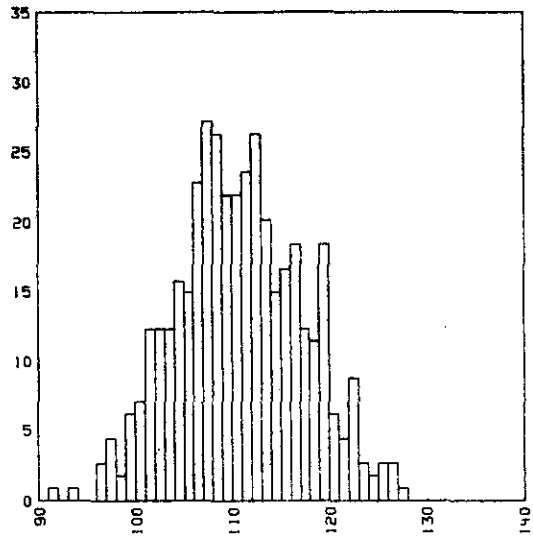
1600 POINTS POLYHEDRON


ELP Y= 3.475X^0.53
σ=4 Y= 7.062X^0.25
σ=2 Y= 3.349X^0.48
σ=1 Y= 3.177X^0.53
ELIPSE
σ=1 SPHERE
σ=2 SPHERE
σ=4 SPHERE
NUMBER OF POINTS ON HULL
NUMBER OF ORIGINAL DATA POINTS

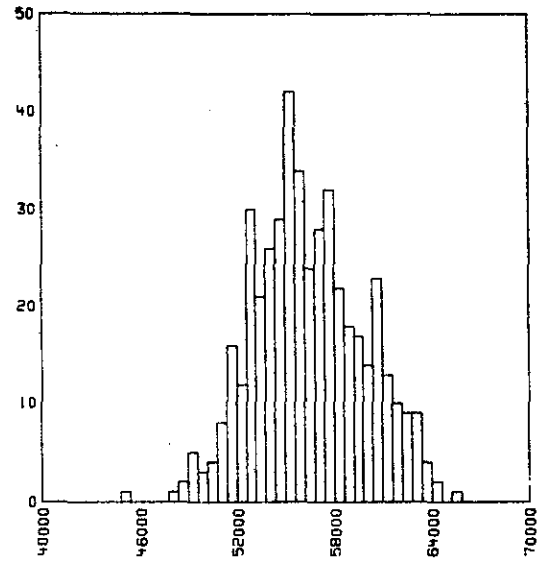

ELP Y= 13.20X^1.28
σ=4 Y= 14.95X^1.09
σ=2 Y= 11.11X^1.23
σ=1 Y= 9.734X^1.30
ELIPSE
σ=1 SPHERE
σ=2 SPHERE
σ=4 SPHERE
NUMBER OF TESTS
NUMBER OF ORIGINAL DATA POINTS


ELP Y= 3.801X^0.75
σ=4 Y= 2.118X^0.84
σ=2 Y= 3.322X^0.75
σ=1 Y= 3.066X^0.77
ELIPSE
NUMBER OF TESTS/
NUMBER OF POINTS ON HULL
NUMBER OF ORIGINAL DATA POINTS

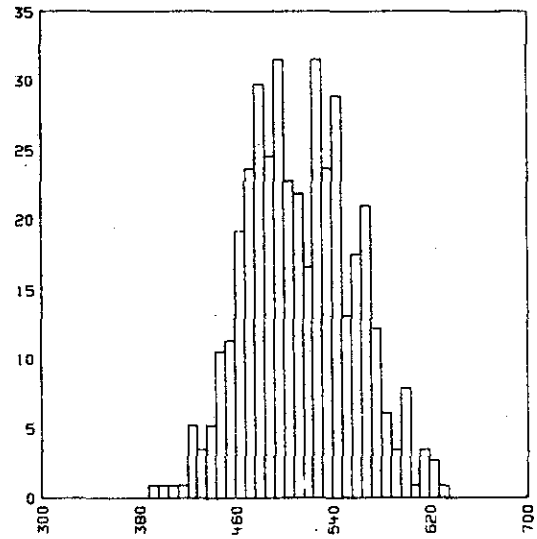

ELP Y= 5.448X^1.01
σ=4 Y= 4.979X^1.02
σ=2 Y= 5.083X^1.02
σ=1 Y= 5.208X^1.02
ELIPSE
NUMBER OF EQUIV TESTS/
NUMBER OF POINTS ON HULL
NUMBER OF ORIGINAL DATA POINTS



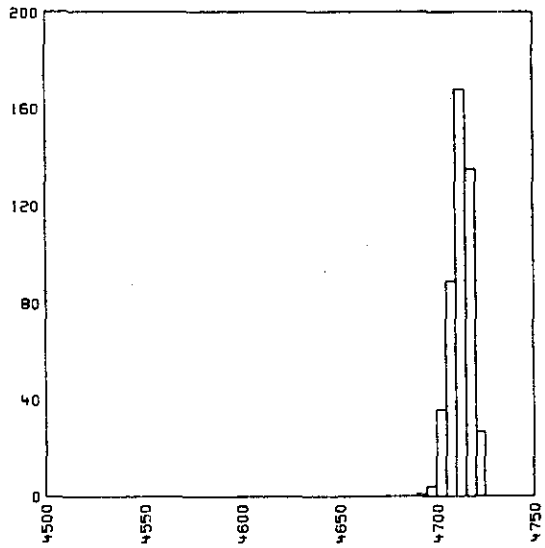
NUMBER OF POINTS ON HULL



NUMBER OF TESTS

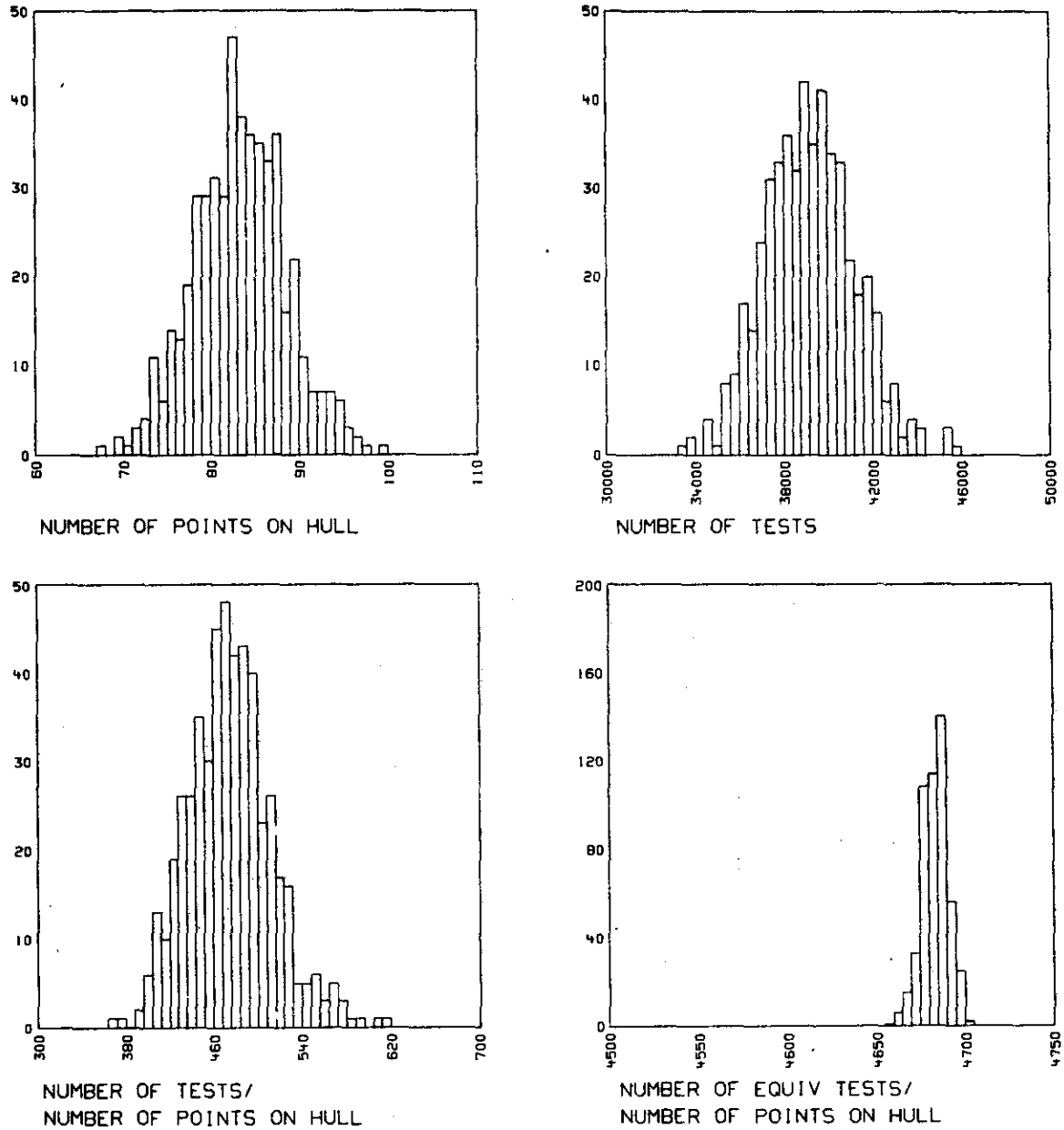


NUMBER OF TESTS/
NUMBER OF POINTS ON HULL

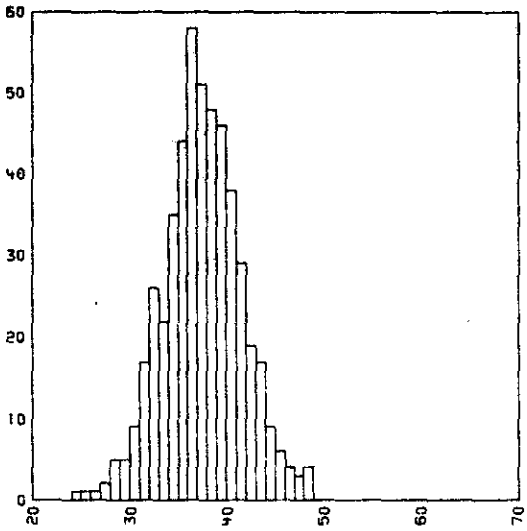


NUMBER OF EQUIV TESTS/
NUMBER OF POINTS ON HULL

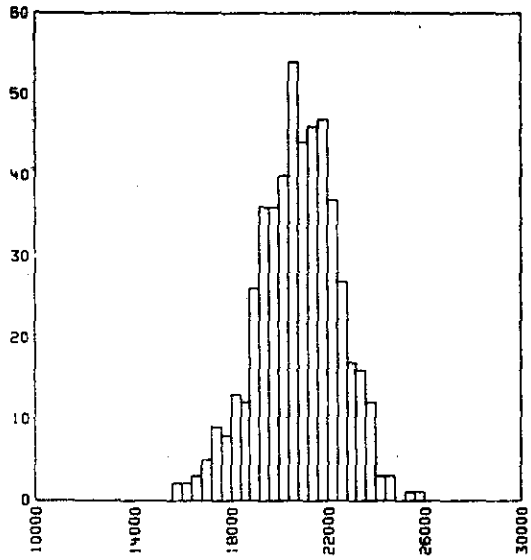
800 POINTS SPHERE $\sigma = 1$



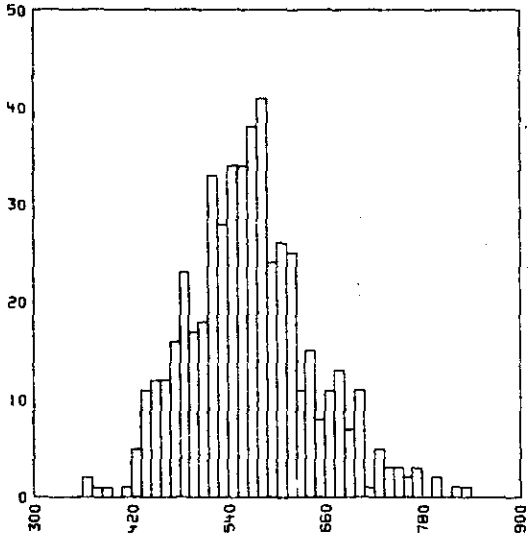
800 POINTS SPHERE $\sigma=2$



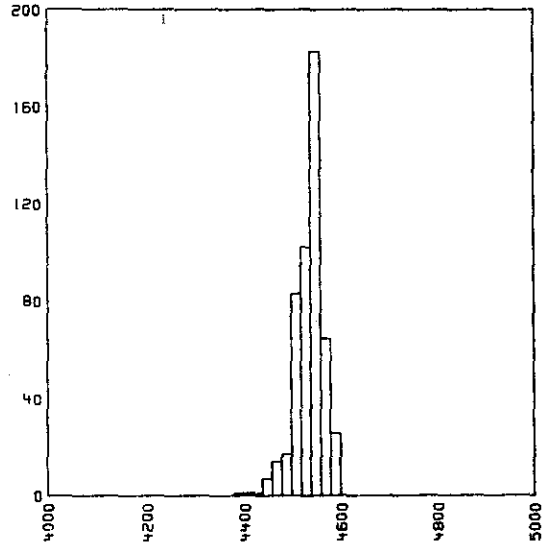
NUMBER OF POINTS ON HULL



NUMBER OF TESTS

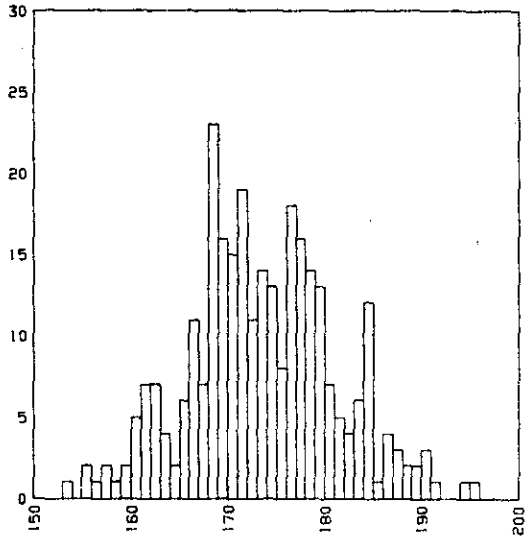


NUMBER OF TESTS/
NUMBER OF POINTS ON HULL

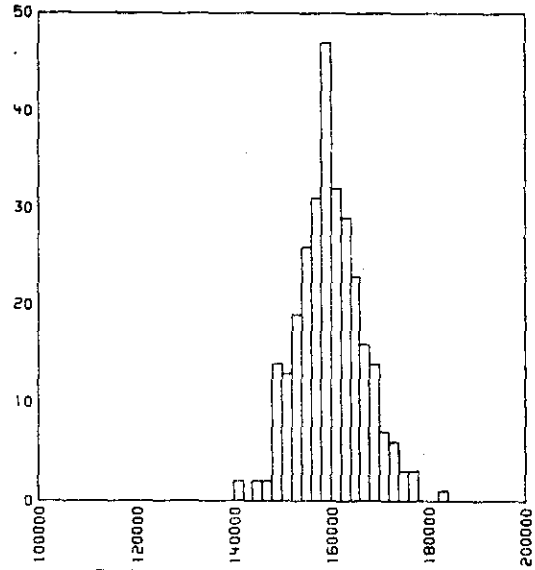


NUMBER OF EQUIV TESTS/
NUMBER OF POINTS ON HULL

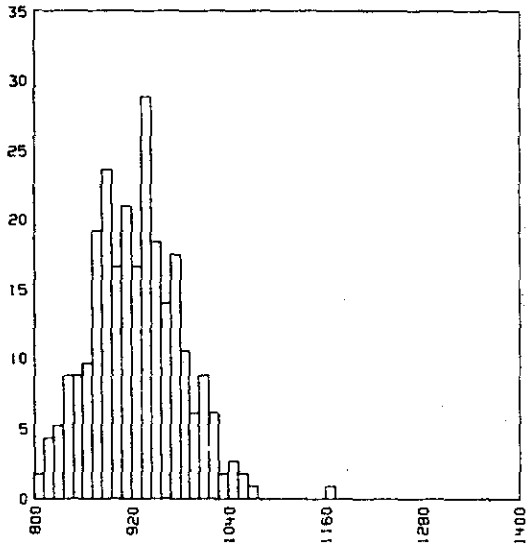
800 POINTS SPHERE $\sigma=4$



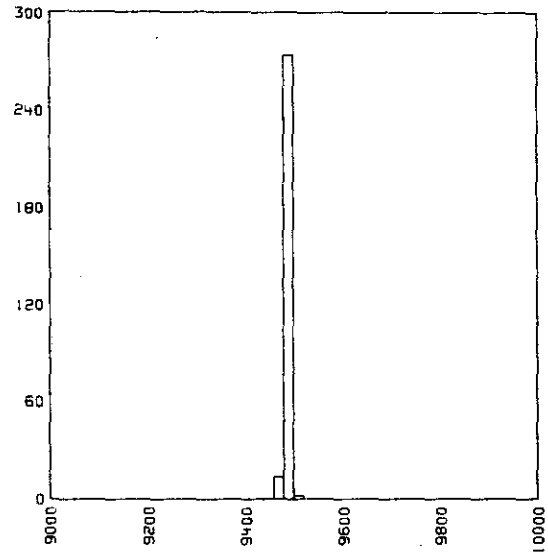
NUMBER OF POINTS ON HULL



NUMBER OF TESTS



NUMBER OF TESTS/
NUMBER OF POINTS ON HULL



NUMBER OF EQUIV TESTS/
NUMBER OF POINTS ON HULL

1600 POINTS ELIPSE

APPENDIX II

Minimum Volume Sphere Data

For the minimum volume sphere algorithm, the following quantities were recorded.

A. Number of Original Data Points (NPTS). All of the graphs are plotted versus this quantity. Note that the algorithm actually uses only the ordered subset of points on the convex hull.

B. Number of Tests (NT). A test is defined as substitution of a point (x_1,y_1,z_1) into the equation of a sphere of radius R. We calculate $(x_1 - x_c)^2 + (y_1 - y_c)^2 + (z_1 - z_c)^2 = R^2$, where (x_c,y_c,z_c) is the center of the sphere.

$R_1^2 < R^2$ (x_1,y_1,z_1) inside sphere

$R_1^2 = R^2$ (x_1,y_1,z_1) on the sphere

$R_1^2 > R^2$ (x_1,y_1,z_1) outside the sphere

C. Number of Spheres Created (NSC). This is the total number of spheres created during the search for the minimum volume sphere.

D. Number of Spheres Tested (NST). Only those spheres created, whose center passes the tests defined in Chapter 5, Part C, are tested to ascertain if all the points lie in or on the sphere.

E. Expected Spheres Created (using convex hull) (ESC). Using the number of points on the convex hull (NPTH), ESC = $\left[\binom{NPTH}{4} + \binom{NPTH}{3} + 1\right]/2$.

F. Selected Points (SP). A tabulation was made of the position in the ordered list of the points used to generate the minimum

volume sphere. SP shows the distribution of the four points. Remember that two points are always used. A third point is used for three or four-point spheres, while a fourth point is used only for four-point spheres.

G. Type of Sphere Created (TSC). A count was made of the total number of 2,3, and 4-point spheres.

The results are displayed in two groups. Group 1 contains cube $\sigma = 1$, cube $\sigma = 2$, cube $\sigma = 4$, and Polyhedron. Group 2 contains sphere $\sigma = 1$, sphere $\sigma = 2$, sphere $\sigma = 4$, and Ellipse. The same data is plotted for each group. The data from Group 1 is plotted, followed by the data for Group 2.

Summary Graphs

The results of the calculations over the 500 cases run for each data set are, for the most part, not normally distributed. They have an exponential type distribution. For this reason, the average value is not very meaningful. We chose, instead, to sort the data and look at four points for each distribution of values. They are as follows.

50%-maximum value for 50% of the data

60%-maximum value for 60% of the data

70%-maximum value for 70% of the data

80%-maximum value for 80% of the data

Each page contains four graphs, one for each of the four data sets in the group. With a few exceptions, each graph contains four curves, one each for 50%, 60%, 70%, and 80%. In the upper left corner of each graph an indication of the trend of the data is given. A straight line least squares fit of the $\log_{10}$ of the last four points was made.

$$\text{Let } y' = mx' + b$$

$$\text{where } y' = \log_{10}(y)$$

$$x' = \log_{10}(x)$$

$$\text{then} \quad y = 10^b x^m$$

The value of m serves as an indication of how the data varies with NPTS. The comparable graphs from the Group 1 and Group 2 data are plotted with the same scales to make comparison easy.

The first four graphs in each group show how the number of tests (NT) varies with NPTS. For the cube data, note that NT/NPTH is practically a constant for all cases. As we predicted, $\sigma = 1$ sphere is the worst case. NT/NPTH varies approximately linearly with NPTS. However, $\sigma = 4$ sphere and the ellipse are nearly constant. We note that $\sigma = 4$ sphere and $\sigma = 4$ cube give essentially the same result.

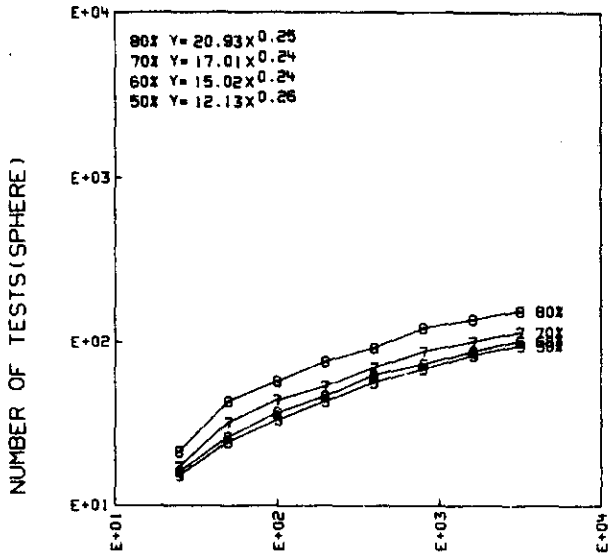
For the sphere $\sigma = 1$ and $\sigma = 2$ data, some data for NPTS = 800 is missing. If the minimum volume sphere was not found by point number 26, then the algorithm stopped and reported NO sphere. For example, consider the graph NT for $\sigma = 1$ sphere. NO sphere was found by point 26 for 40% of the cases tried.

The next five graphs in each group show how the number of spheres created varies with NPTS. The most interesting comparison is ESC with NSC. For example, ESC for ellipse 1600 points at 50% is 10^7 while NSC for the same graph is 1. This means for these particular cases, the minimun volume sphere was found 10,000,000 faster when using the heuristic.

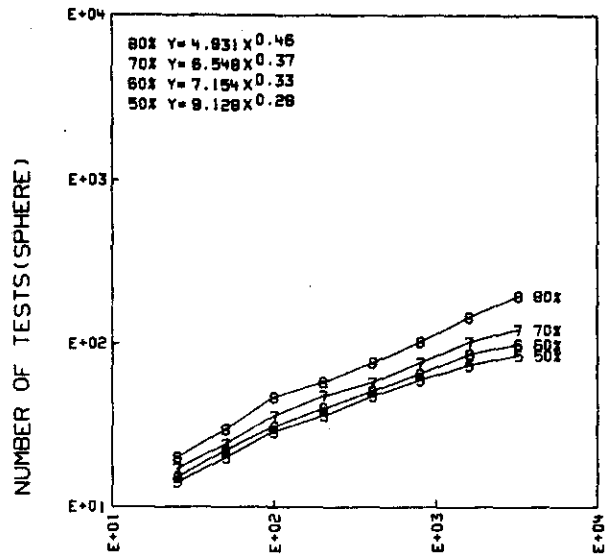
The next graphs show the distribution of the selected points at the 50% point. As expected, $\sigma = 1$ and $\sigma = 2$ sphere show the failure of the heuristic. All other cases show that the selected points are relatively

insensitive to NPTS. It is interesting to note that for $\sigma = 1$ cube, point 1 was on the minimum volume sphere for at least 50% of the cases over a range of input data spanning almost three decades.

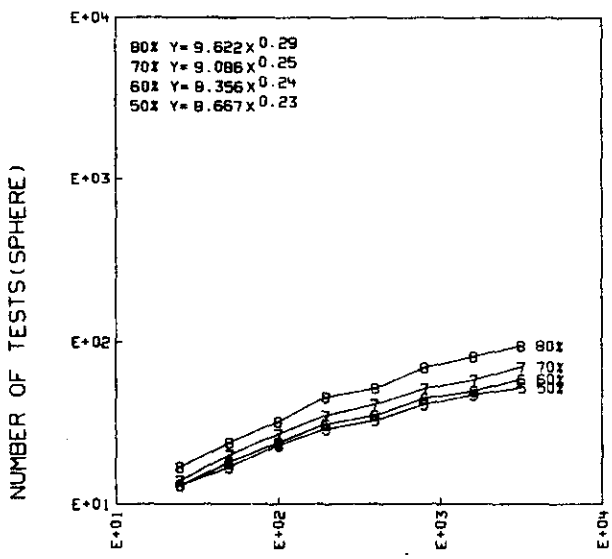
The histograms show some typical distributions for a few selected cases. In the main, they show the effectiveness of the heuristic. There are two histograms for each of the selected cases. Most interesting is the set of histograms that show the distribution of the four points. Without a heuristic, we would expect a uniform distribution. The exponential nature of the histogram shows how effective is the sorting on solid angle.



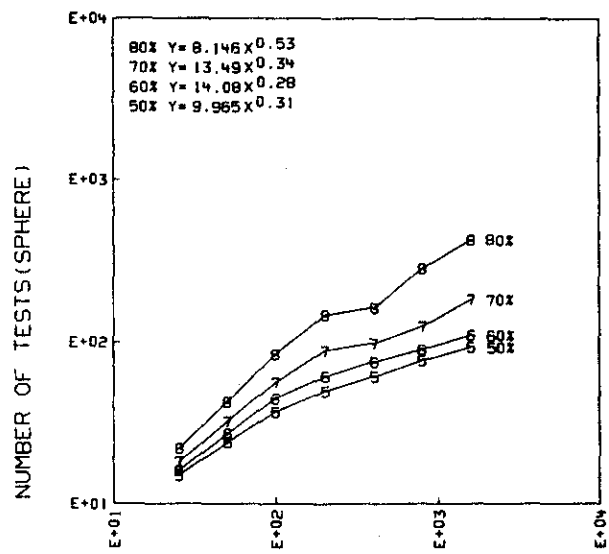
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma = 1$



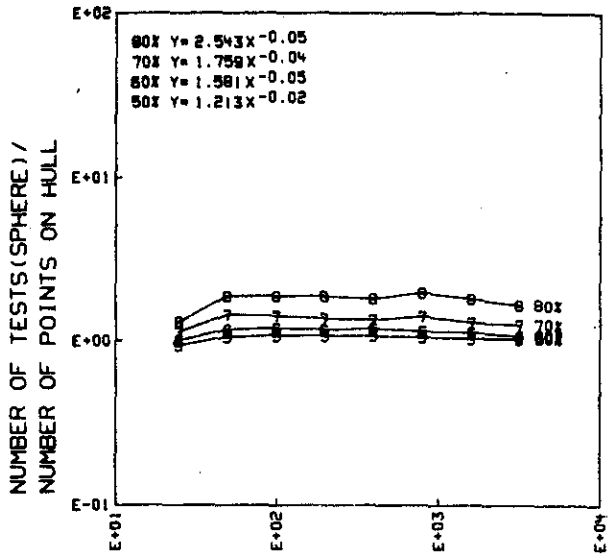
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma = 2$



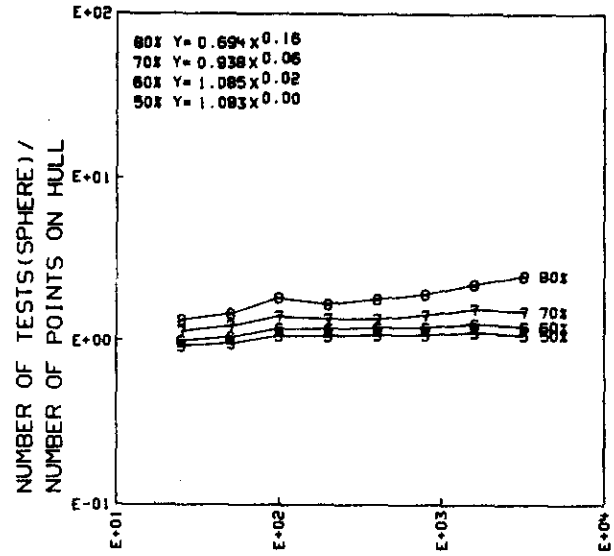
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma = 4$



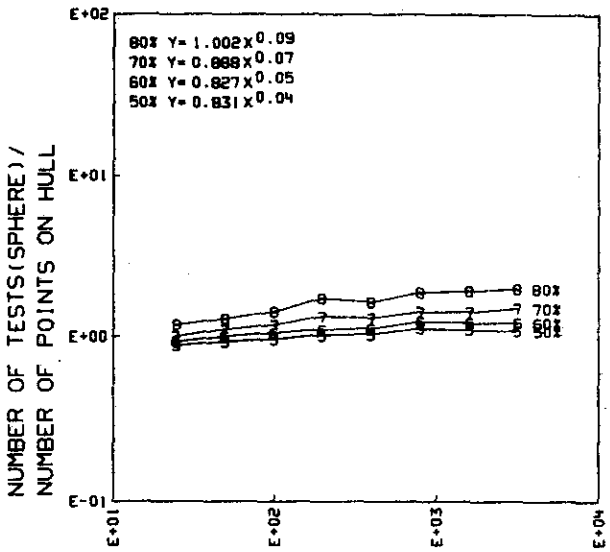
NUMBER OF ORIGINAL DATA POINTS
POLYHEDRON



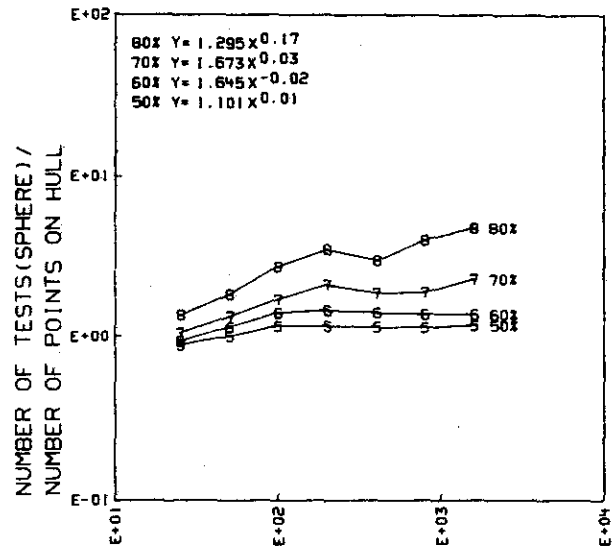
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma=1$



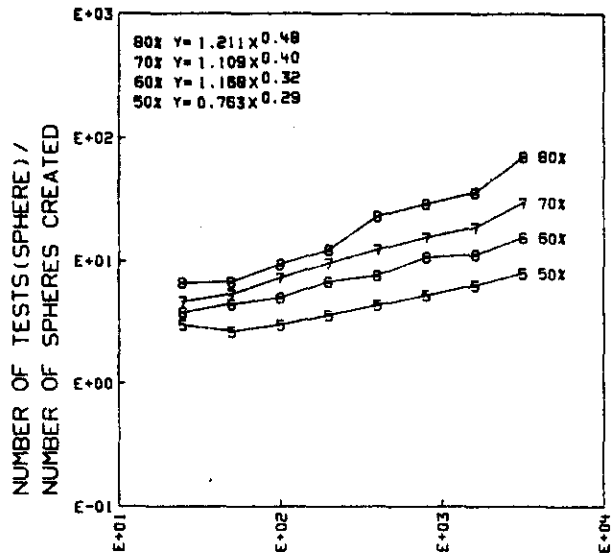
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma=2$



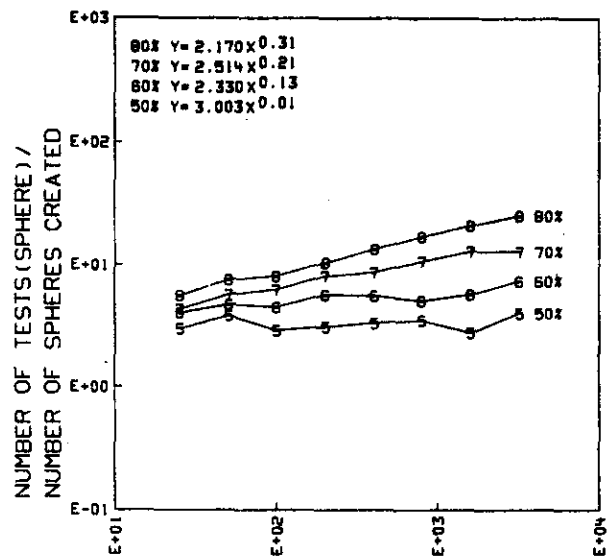
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma=4$



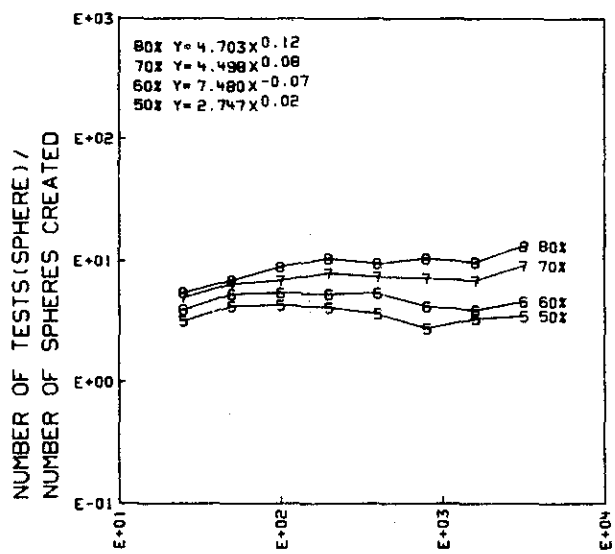
NUMBER OF ORIGINAL DATA POINTS
POLYHEDRON



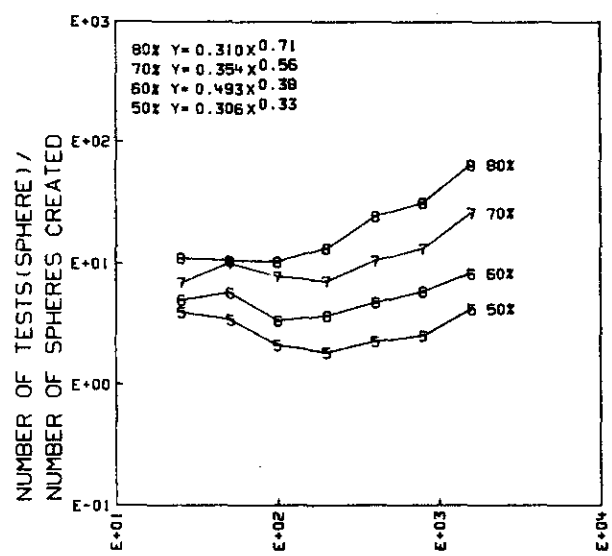
NUMBER OF ORIGINAL DATA POINTS
CUBE σ=1



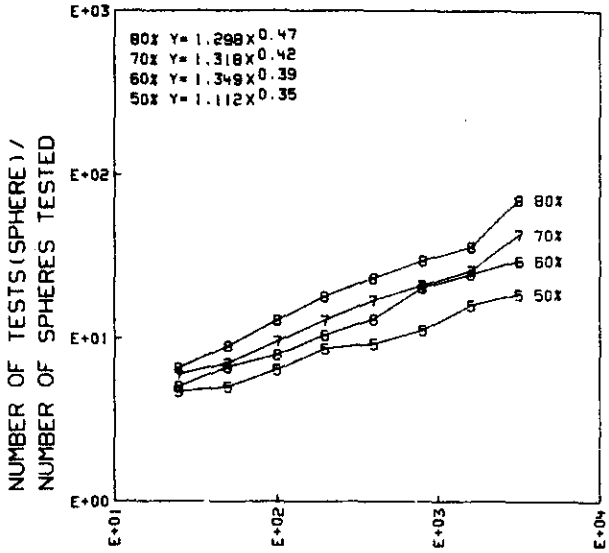
NUMBER OF ORIGINAL DATA POINTS
CUBE σ=2



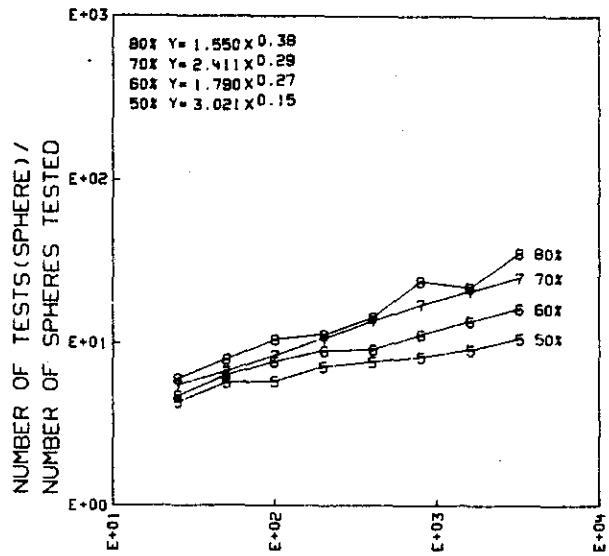
NUMBER OF ORIGINAL DATA POINTS
CUBE σ=4



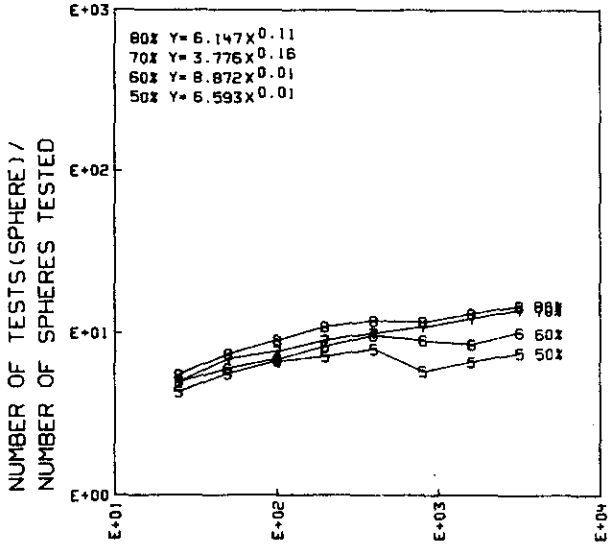
NUMBER OF ORIGINAL DATA POINTS
POLYHEDRON



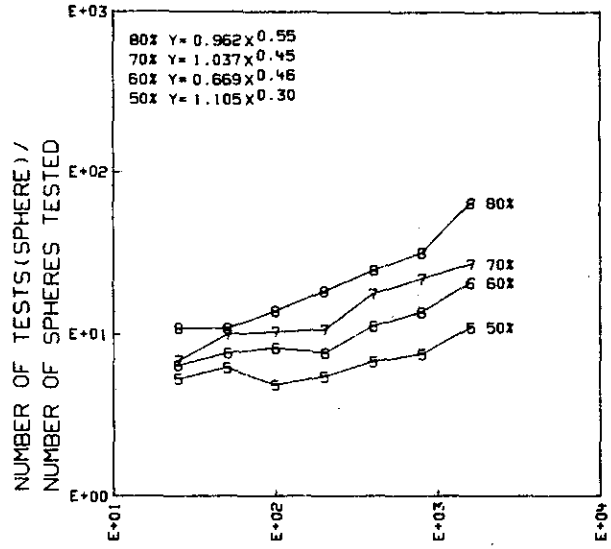
NUMBER OF ORIGINAL DATA POINTS
CUBE σ=1



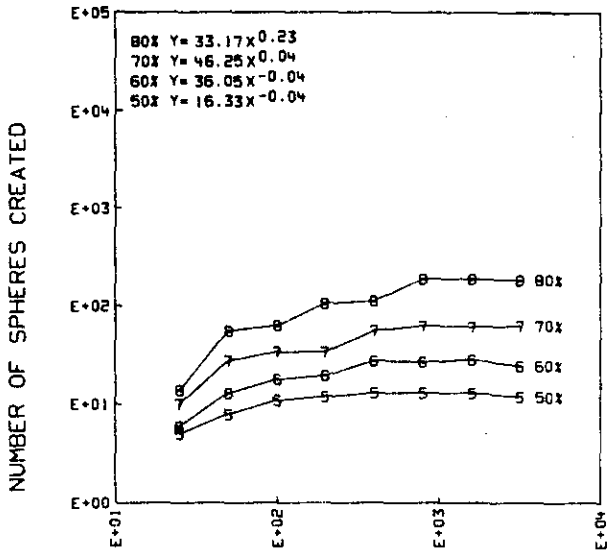
NUMBER OF ORIGINAL DATA POINTS
CUBE σ=2



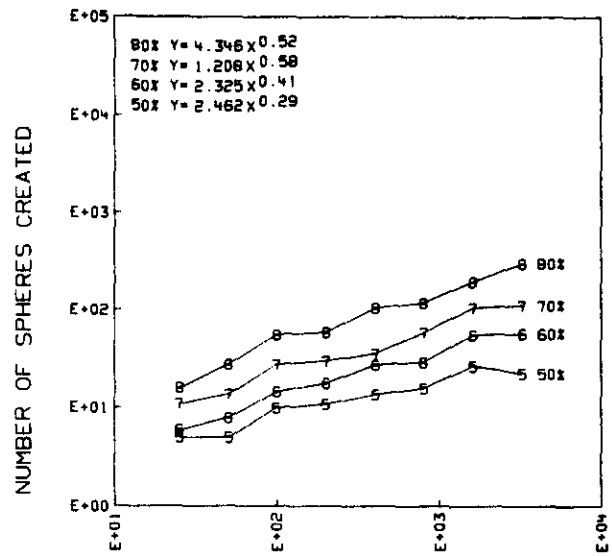
NUMBER OF ORIGINAL DATA POINTS
CUBE σ=4



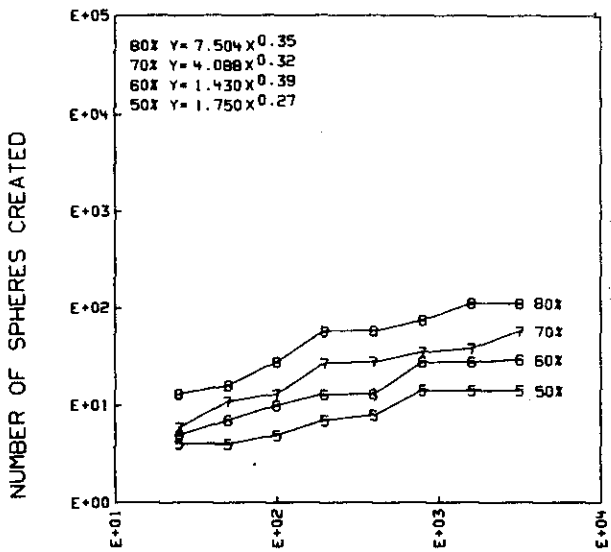
NUMBER OF ORIGINAL DATA POINTS
POLYHEDRON



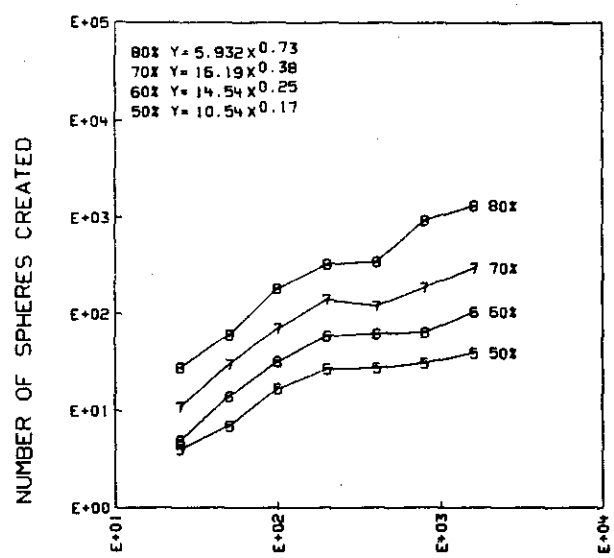
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma=1$



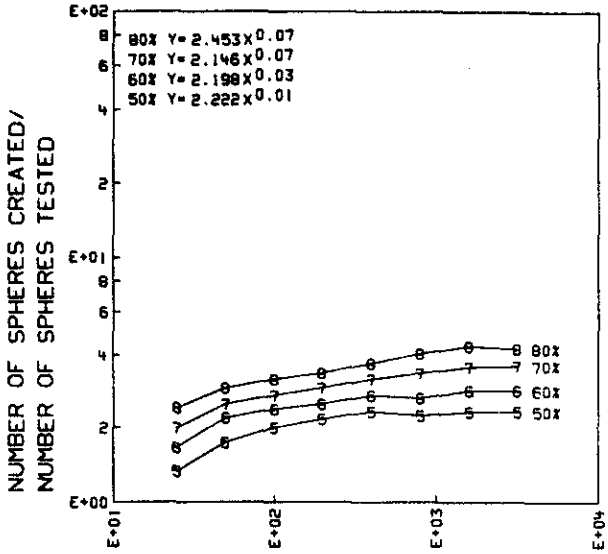
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma=2$



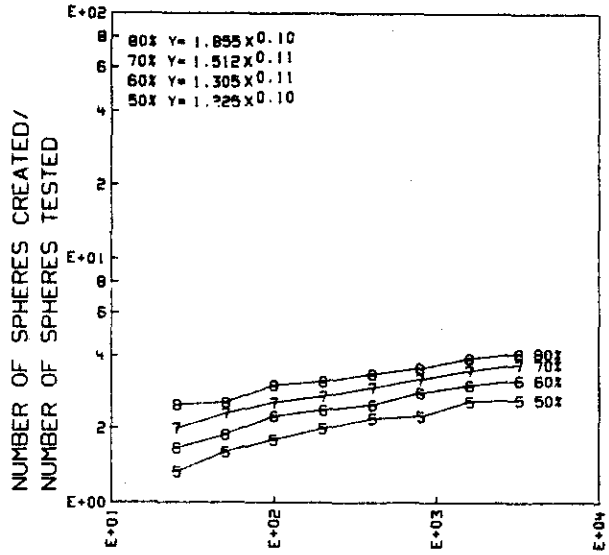
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma=4$



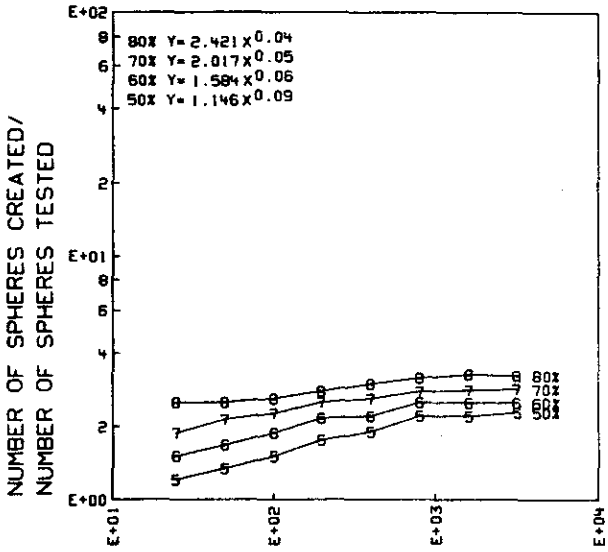
NUMBER OF ORIGINAL DATA POINTS
POLYHEDRON



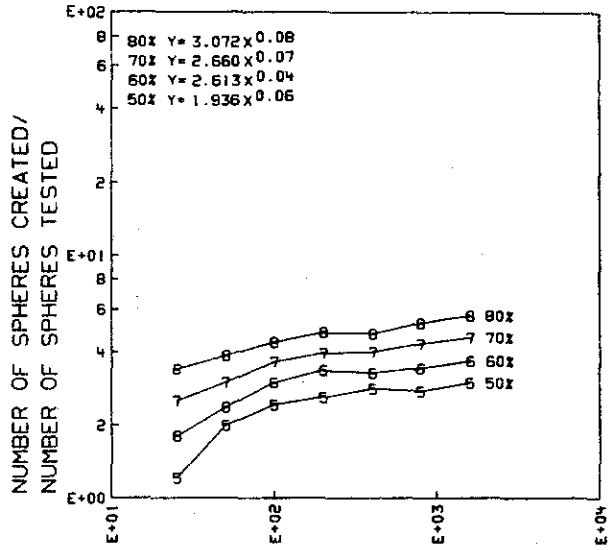
CUBE σ=1



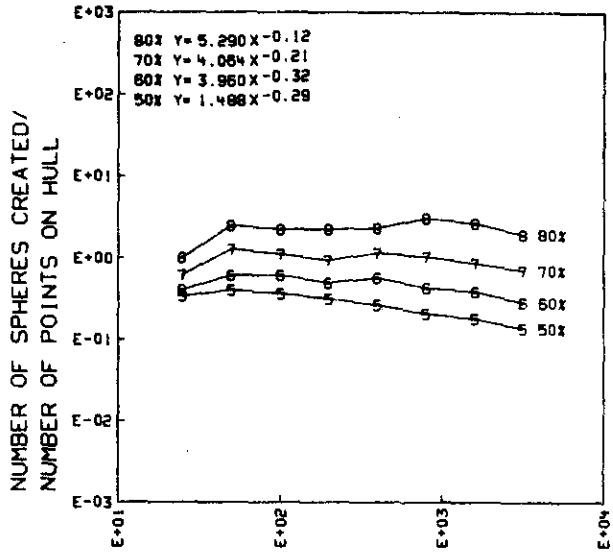
CUBE σ=2



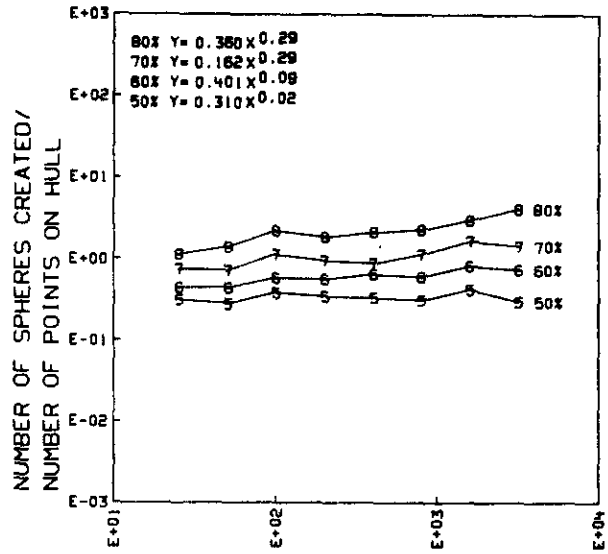
CUBE σ=4



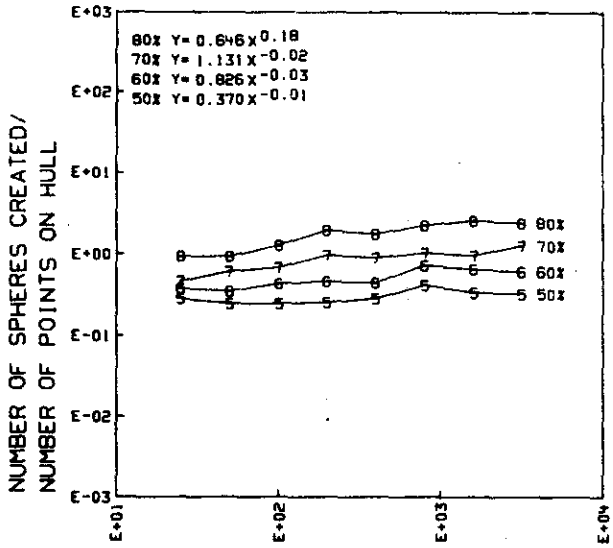
POLYHEDRON



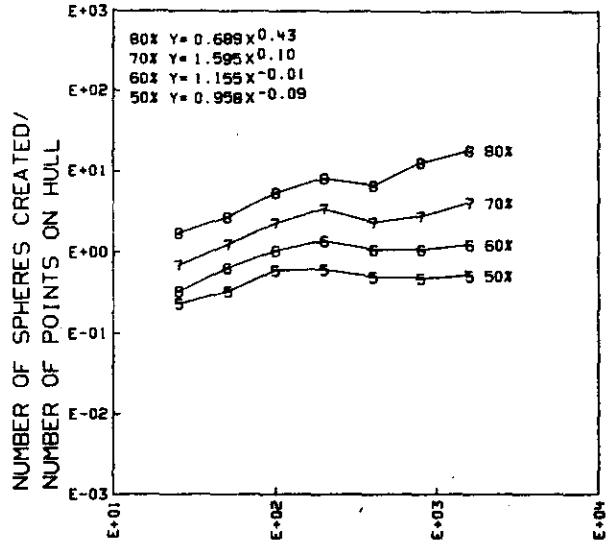
NUMBER OF ORIGINAL DATA POINTS
CUBE σ=1



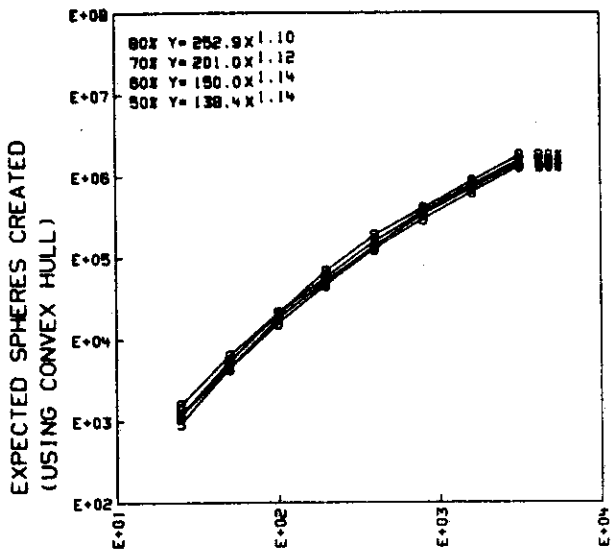
NUMBER OF ORIGINAL DATA POINTS
CUBE σ=2



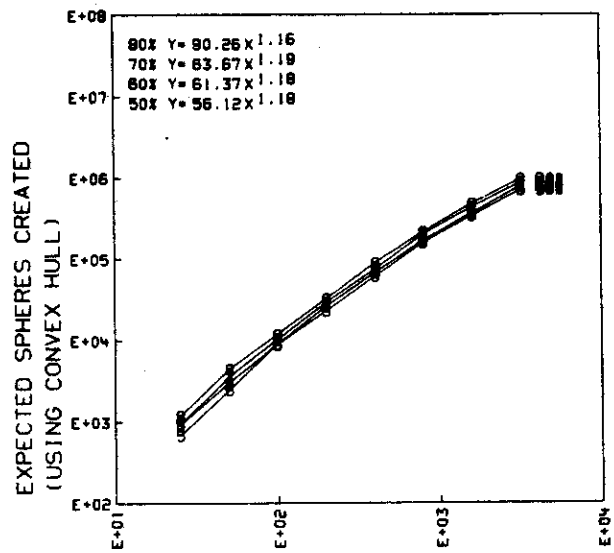
NUMBER OF ORIGINAL DATA POINTS
CUBE σ=4



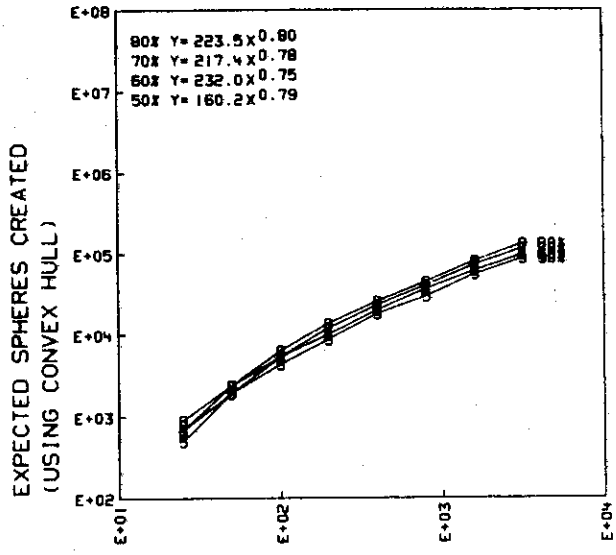
NUMBER OF ORIGINAL DATA POINTS
POLYHEDRON



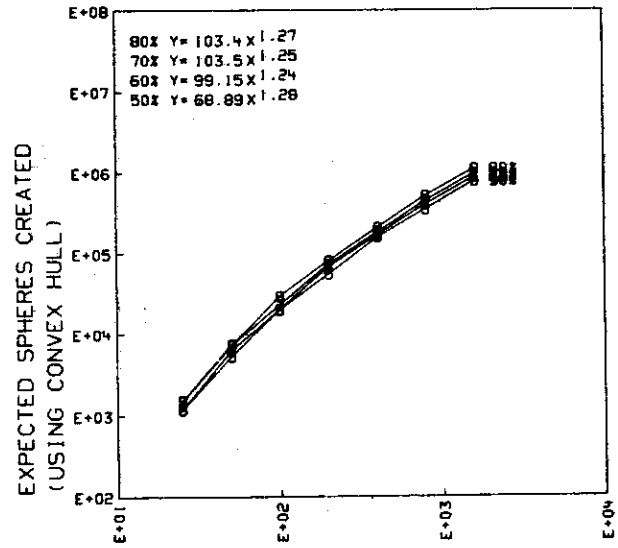
CUBE $\sigma=1$



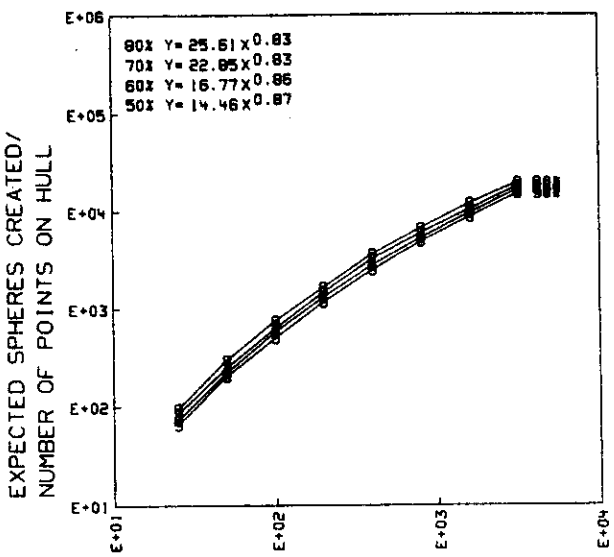
CUBE $\sigma=2$



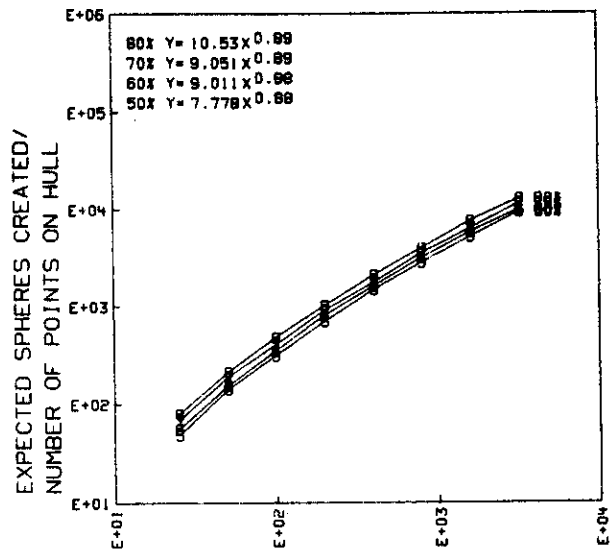
CUBE $\sigma=4$



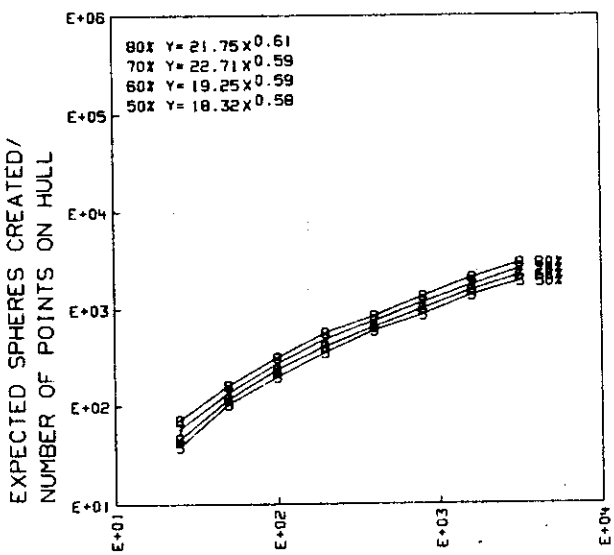
POLYHEDRON



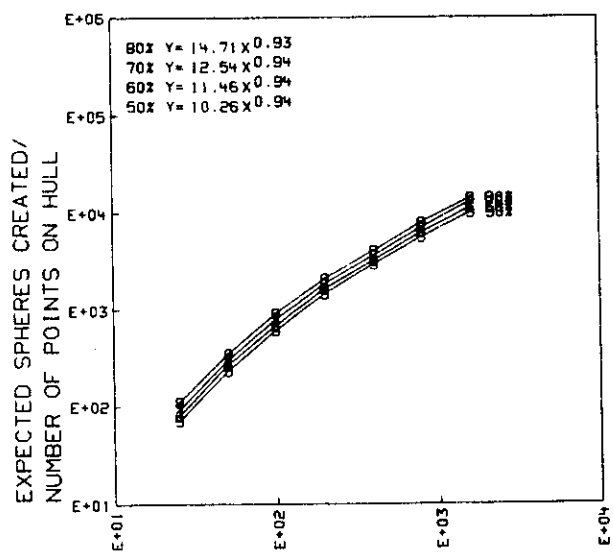
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma=1$



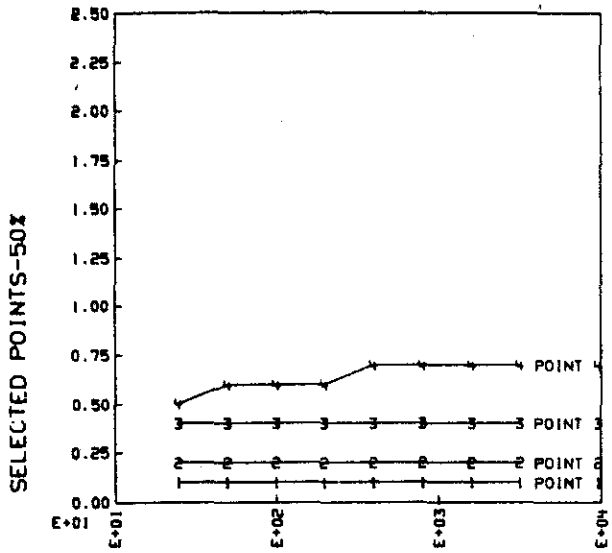
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma=2$



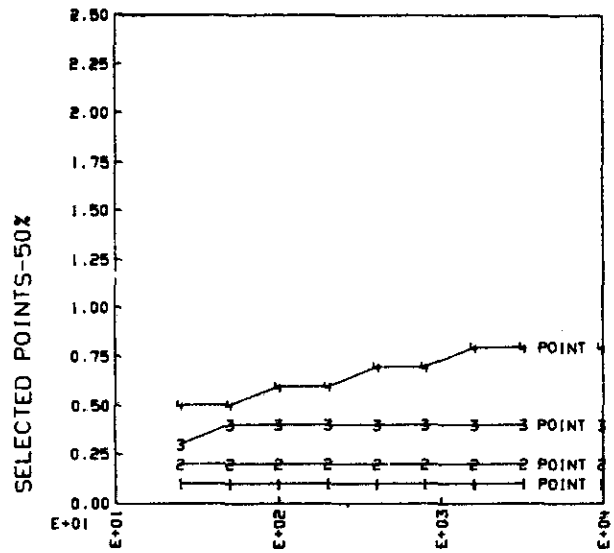
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma=4$



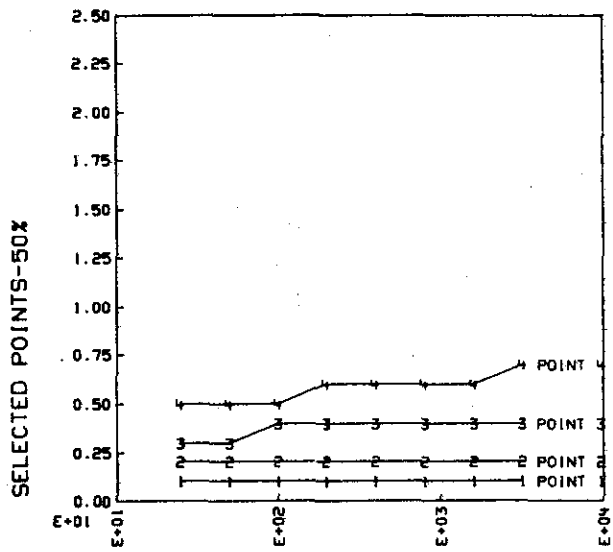
NUMBER OF ORIGINAL DATA POINTS
POLYHEDRON



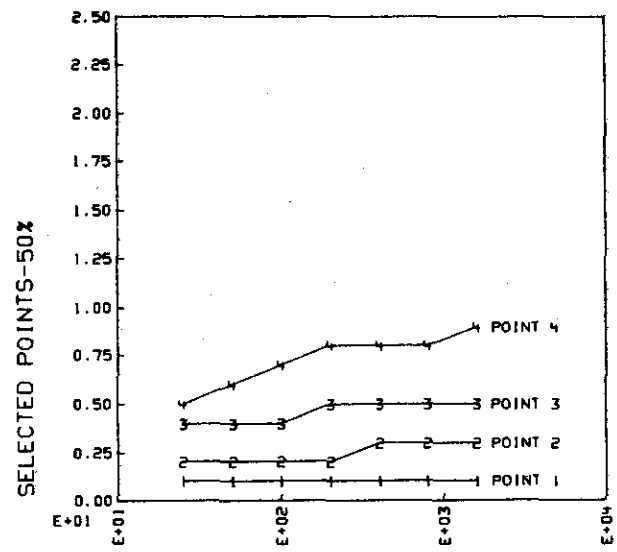
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma=1$



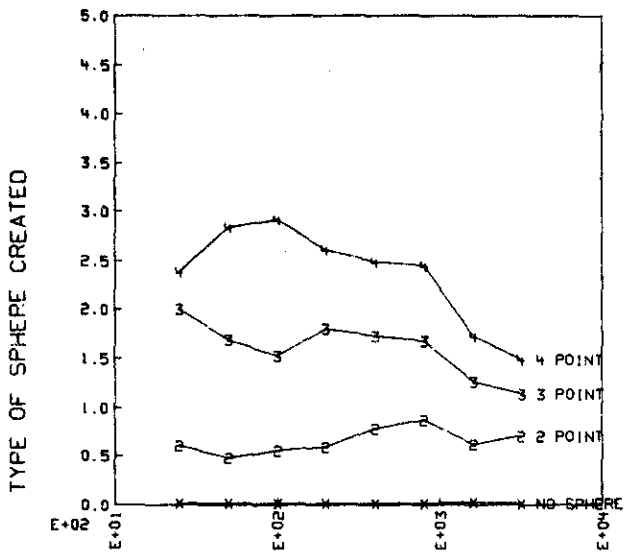
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma=2$



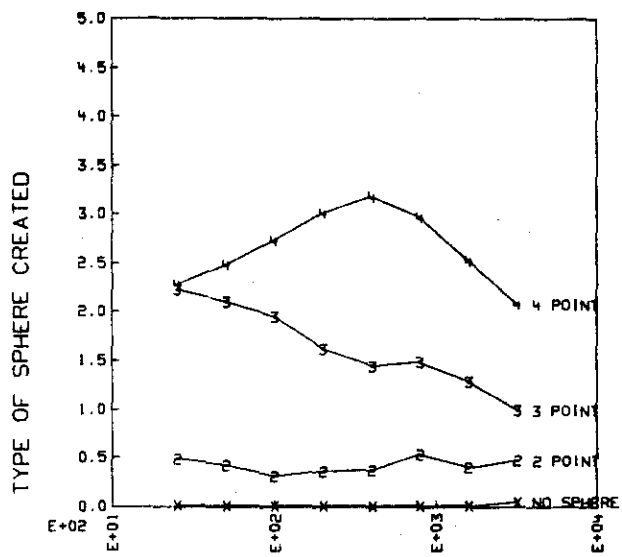
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma=4$



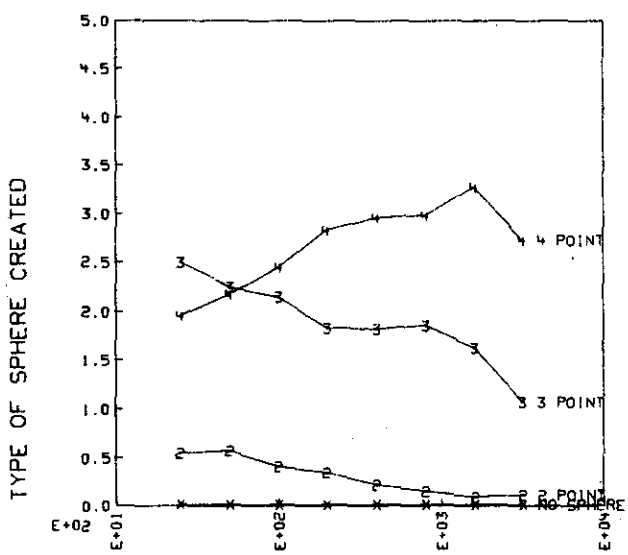
NUMBER OF ORIGINAL DATA POINTS
POLYHEDRON



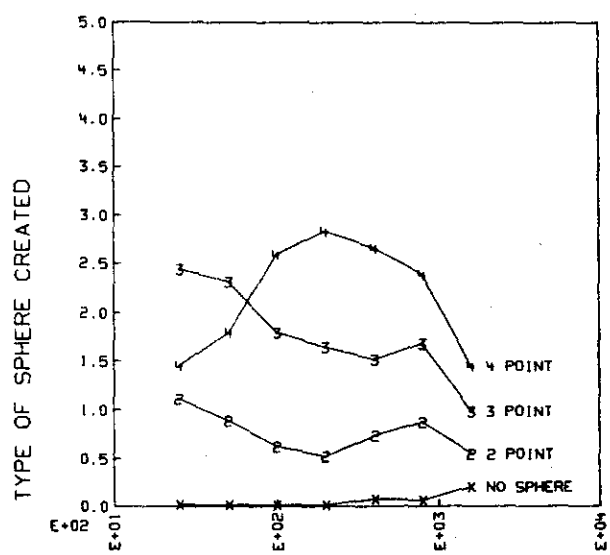
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma=1$



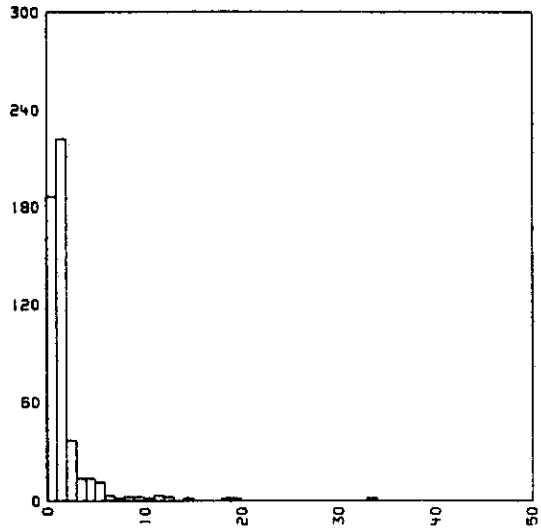
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma=2$



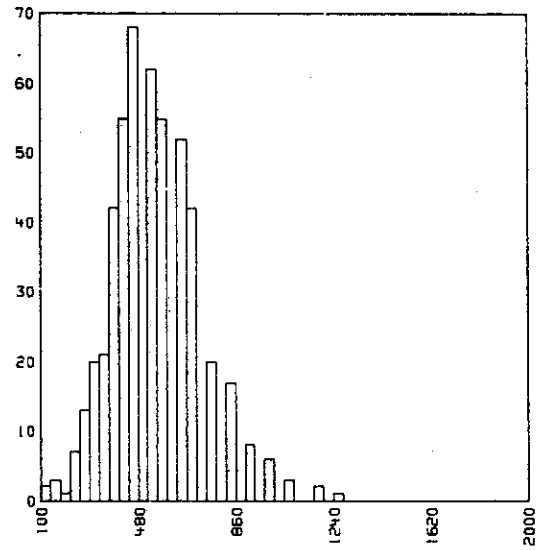
NUMBER OF ORIGINAL DATA POINTS
CUBE $\sigma=4$



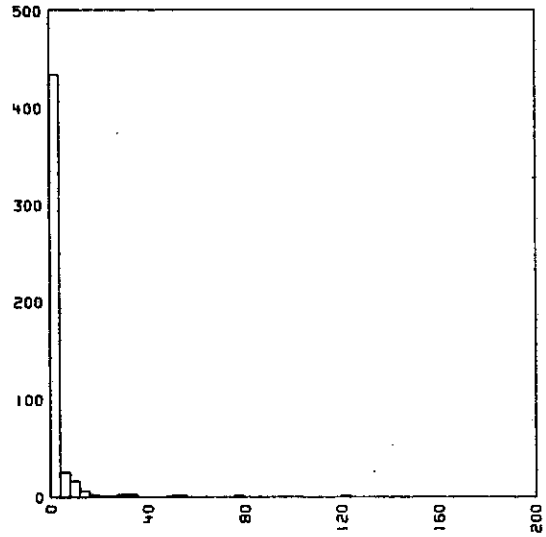
NUMBER OF ORIGINAL DATA POINTS
POLYHEDRON



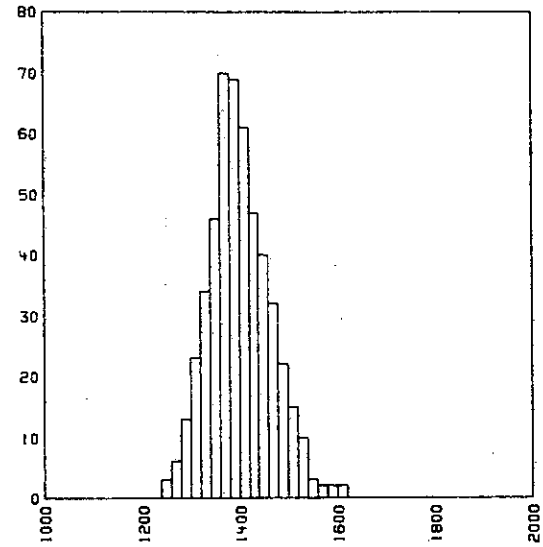
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

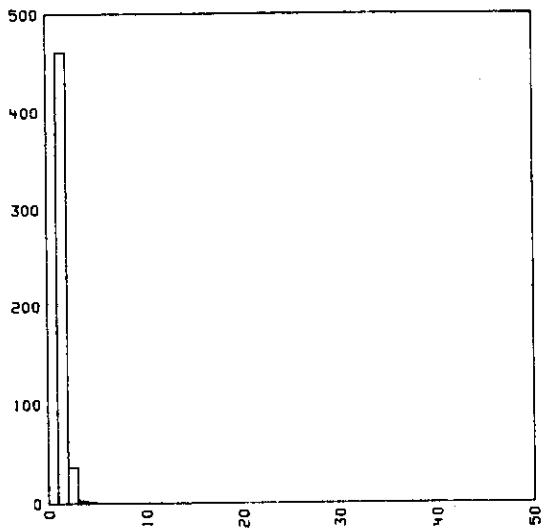


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

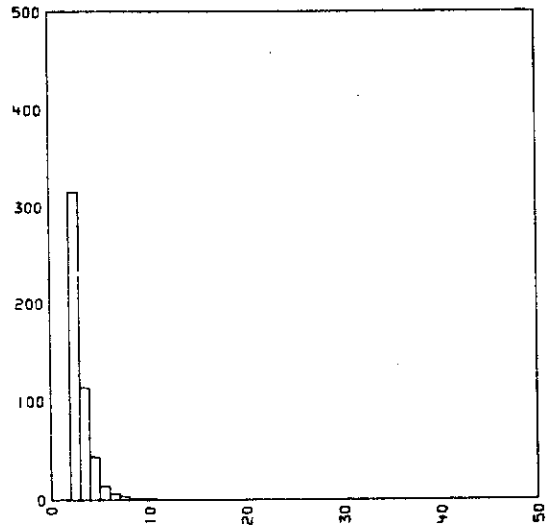


RADIUS

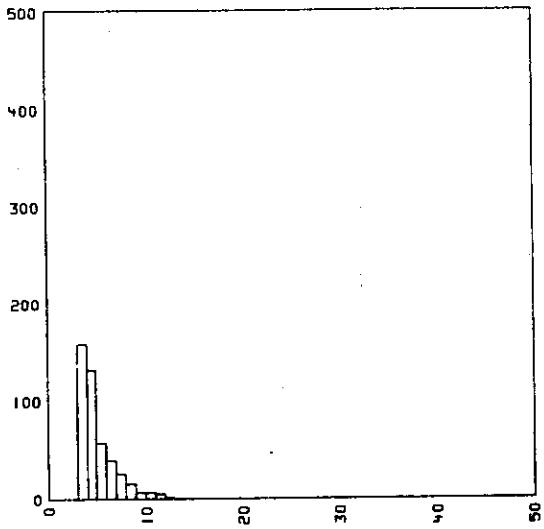
100 POINTS CUBE $\sigma=1$



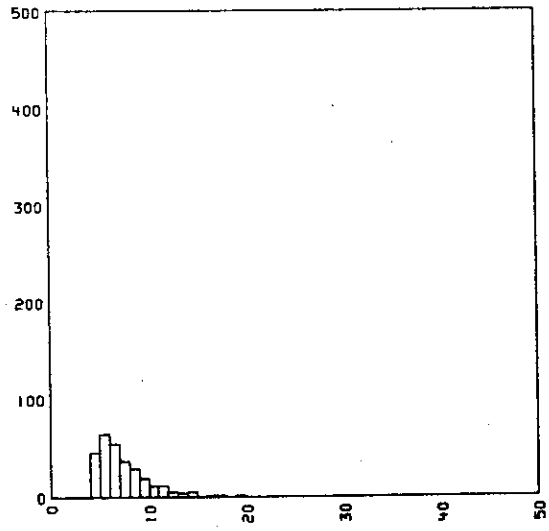
FIRST POINT



SECOND POINT

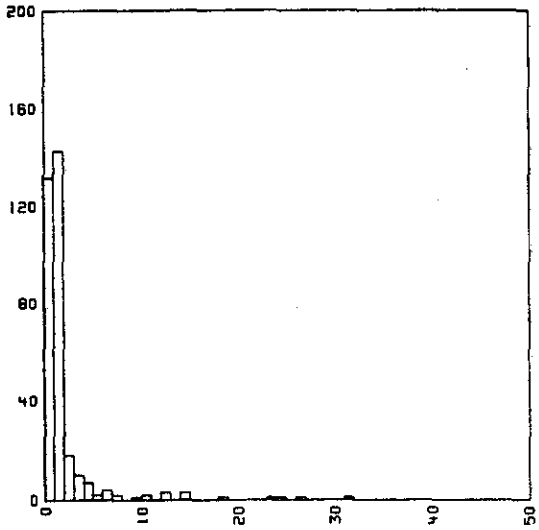


THIRD POINT

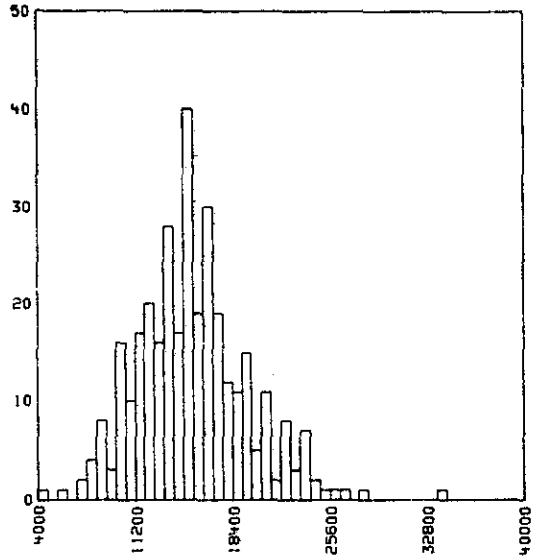


FOURTH POINT

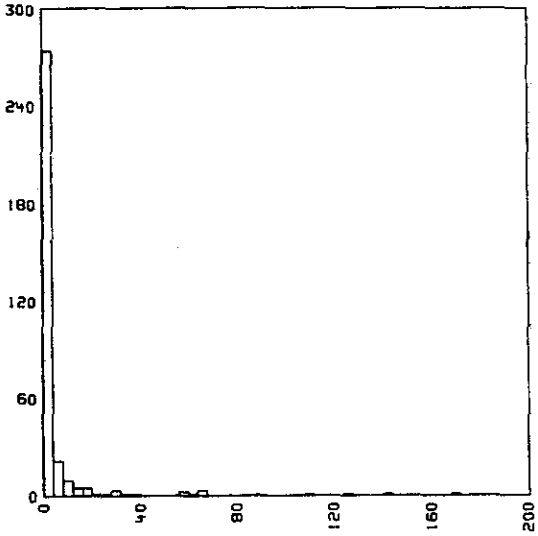
100 POINTS CUBE $\sigma=1$



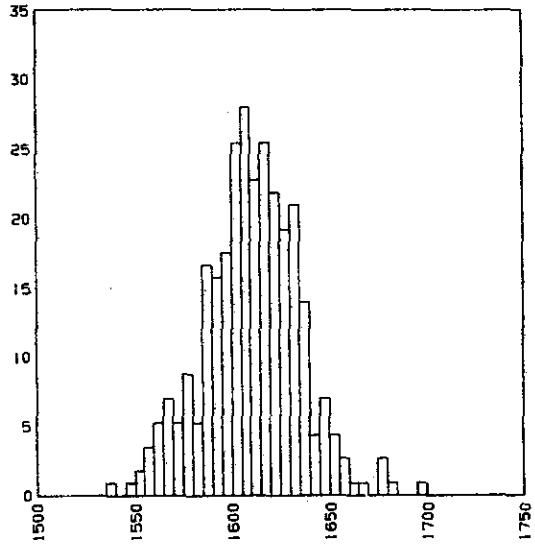
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

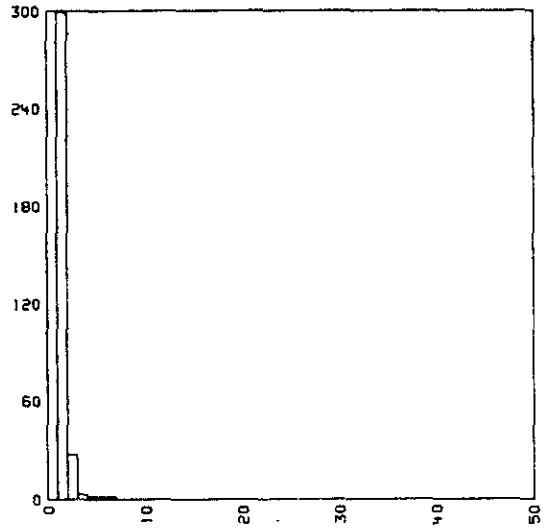


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

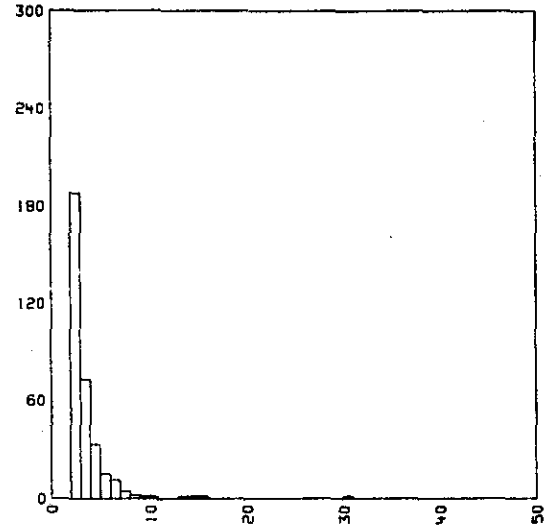


RADIUS

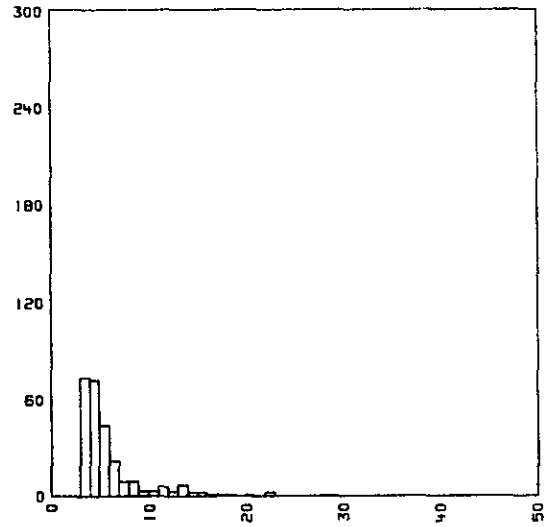
3200 POINTS CUBE $\sigma=1$



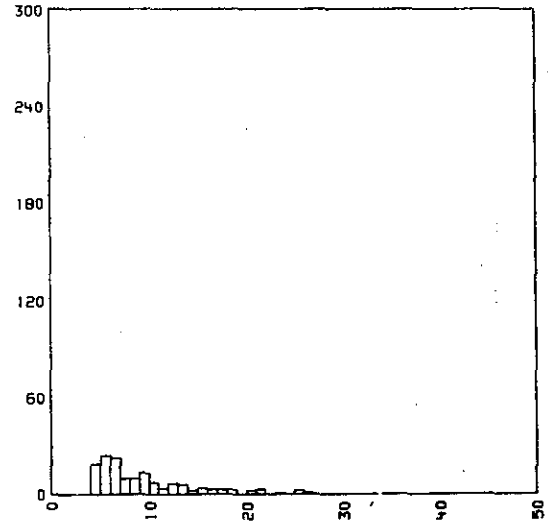
FIRST POINT



SECOND POINT

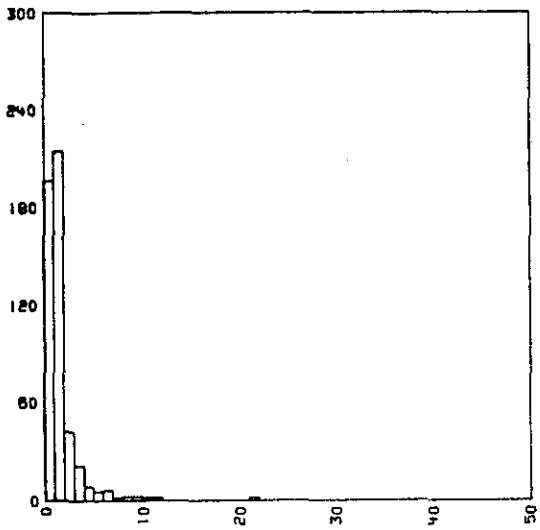


THIRD POINT

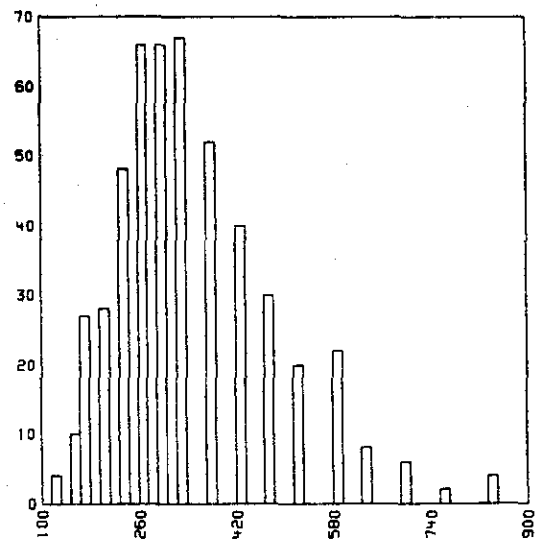


FOURTH POINT

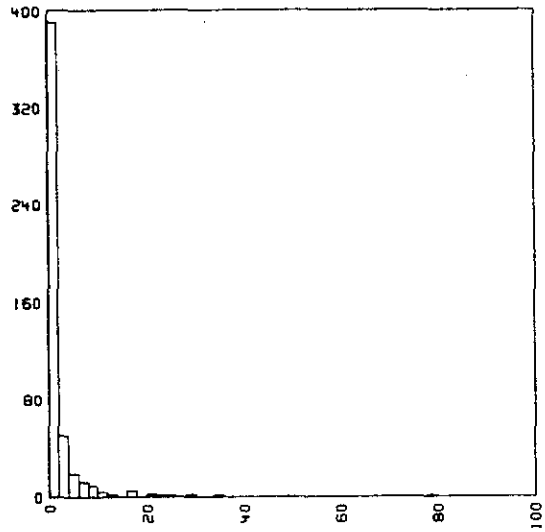
3200 POINTS CUBE $\sigma=1$



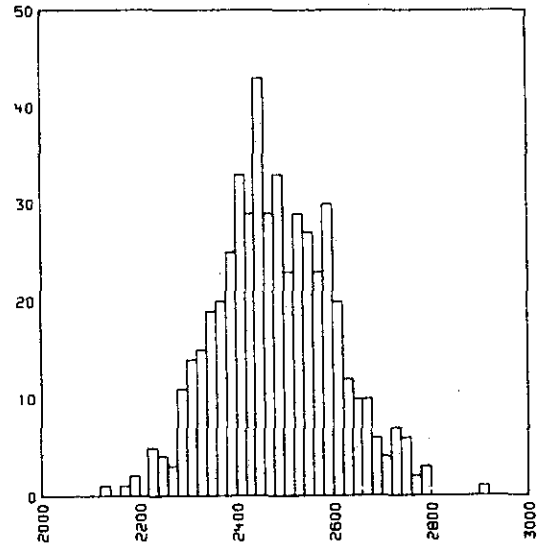
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

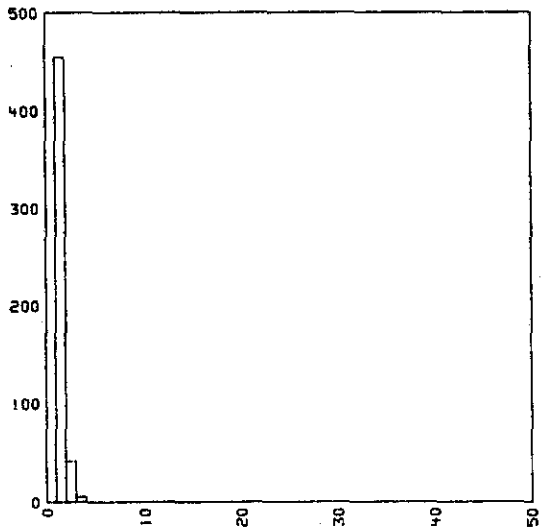


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

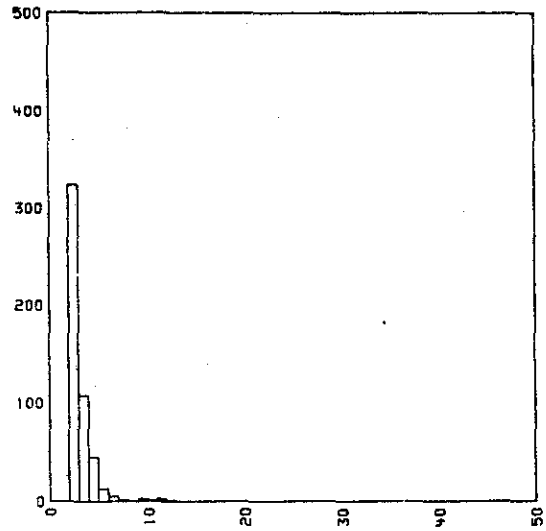


RADIUS

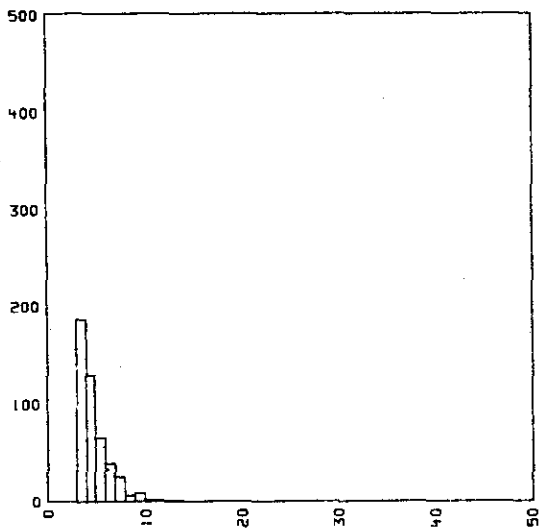
100 POINTS CUBE $\sigma=2$



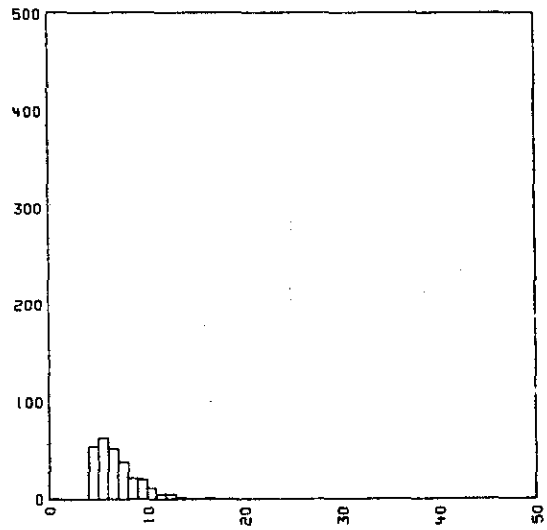
FIRST POINT



SECOND POINT

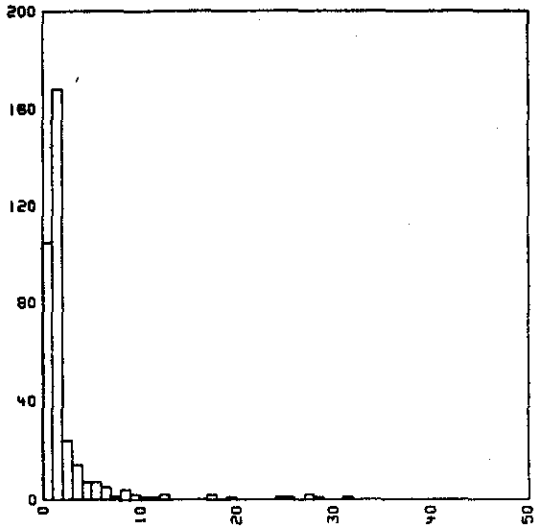


THIRD POINT

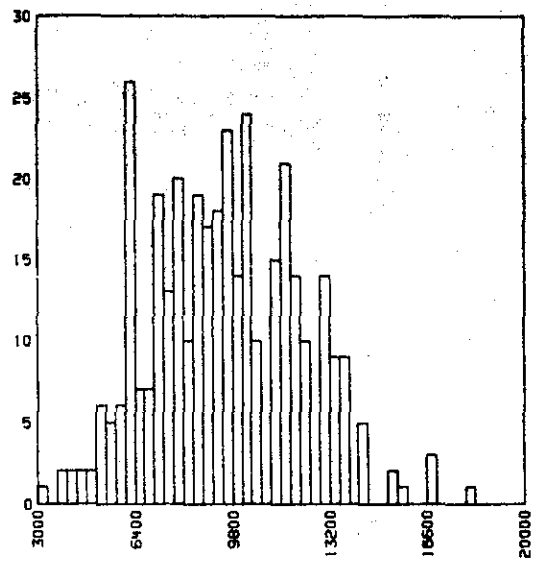


FOURTH POINT

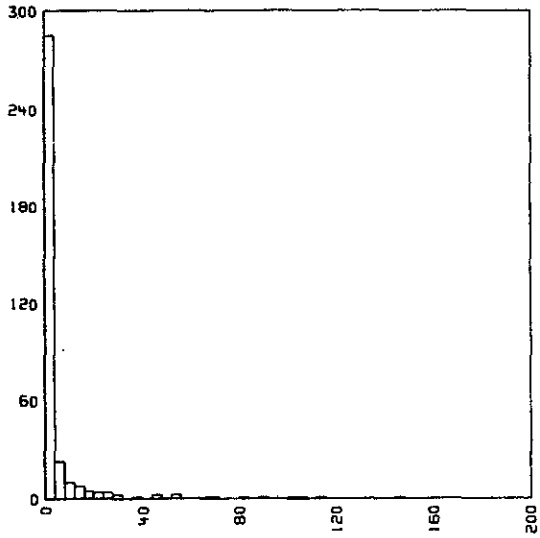
100 POINTS CUBE $\sigma=2$



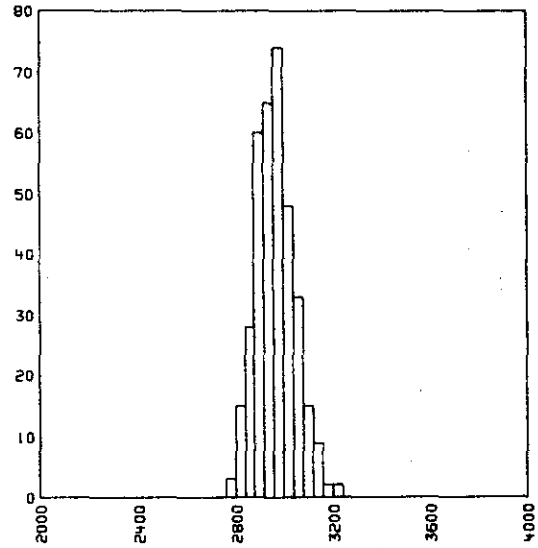
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

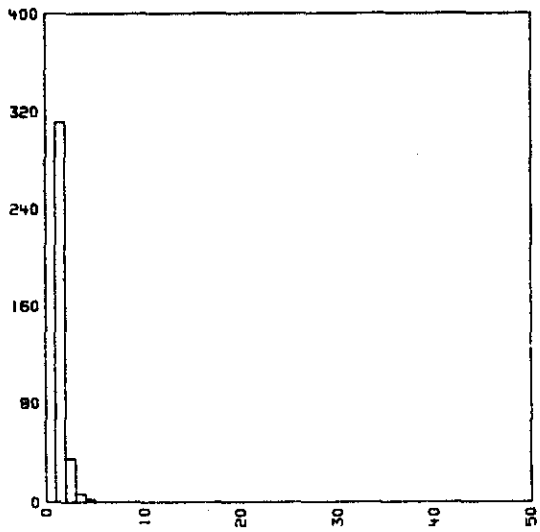


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

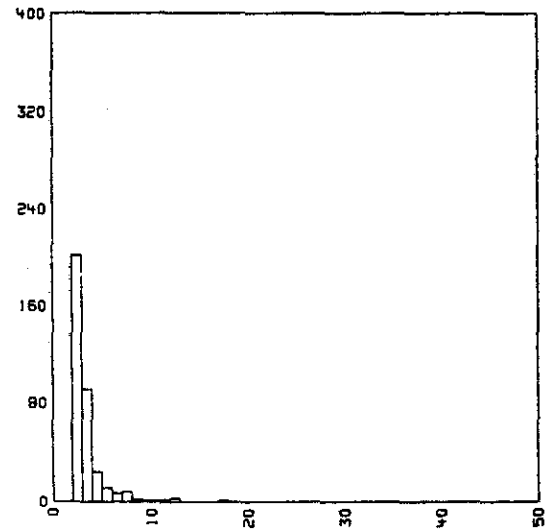


RADIUS

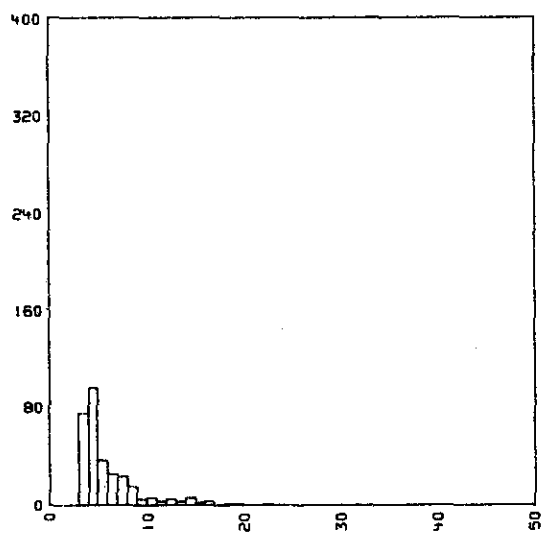
3200 POINTS CUBE $\alpha=2$



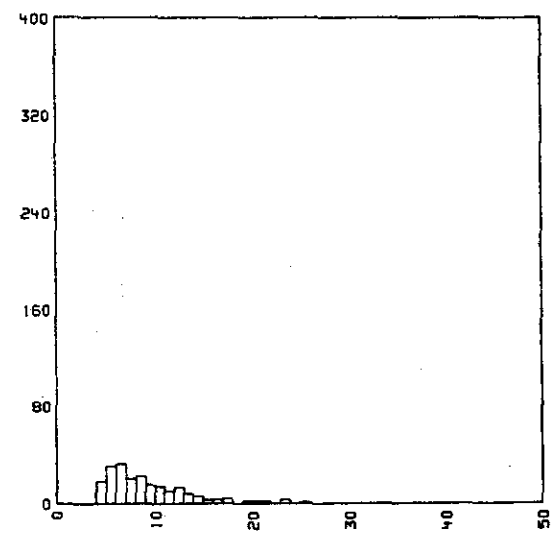
FIRST POINT



SECOND POINT

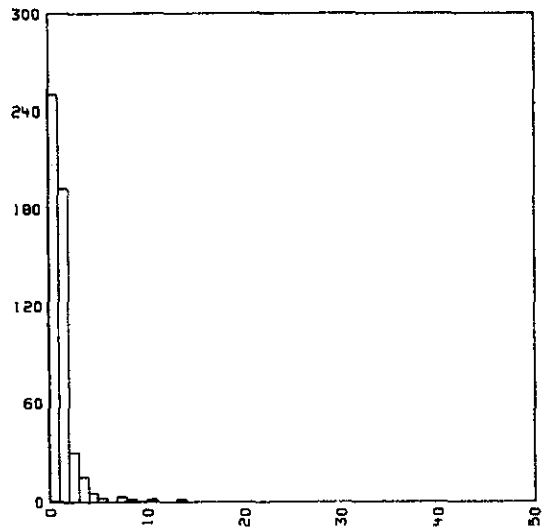


THIRD POINT

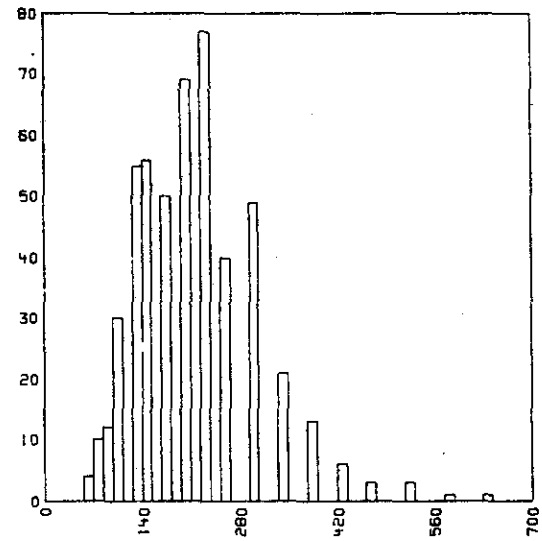


FOURTH POINT

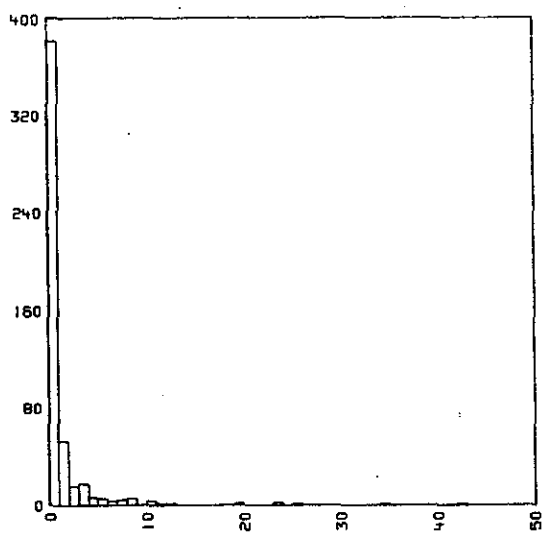
3200 POINTS CUBE $\sigma=2$



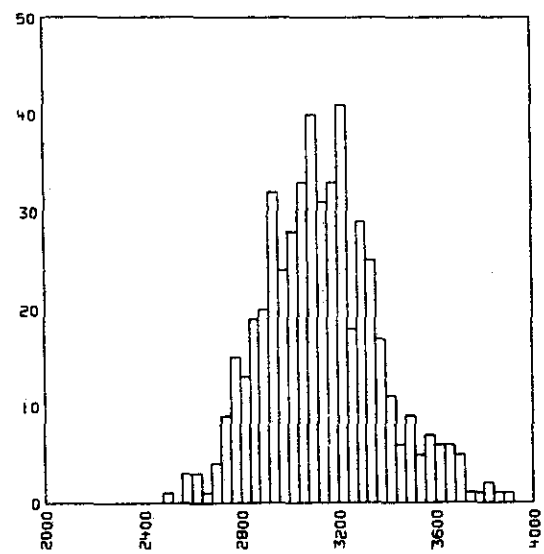
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

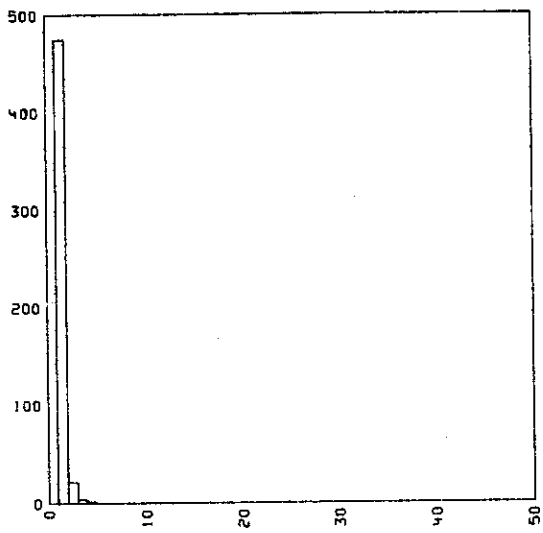


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

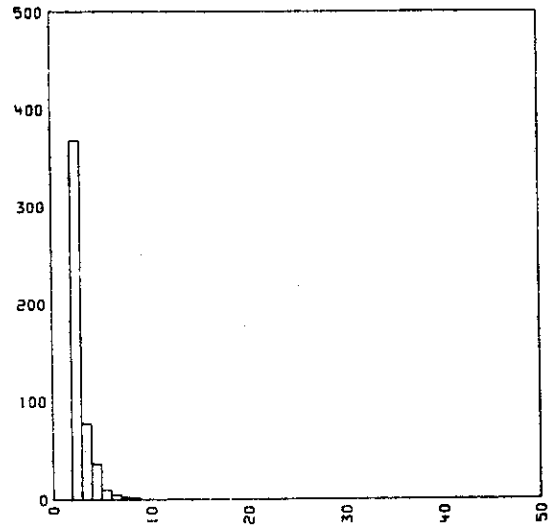


RADIUS

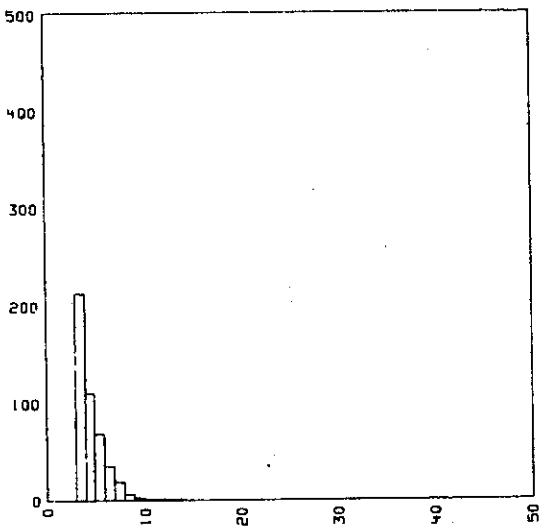
100 POINTS CUBE $\sigma=4$



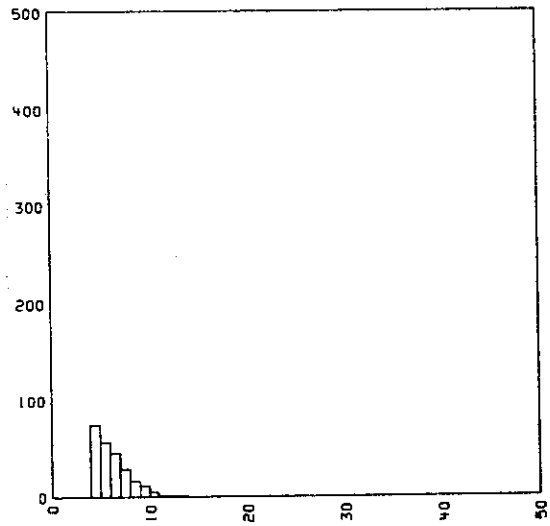
FIRST POINT



SECOND POINT

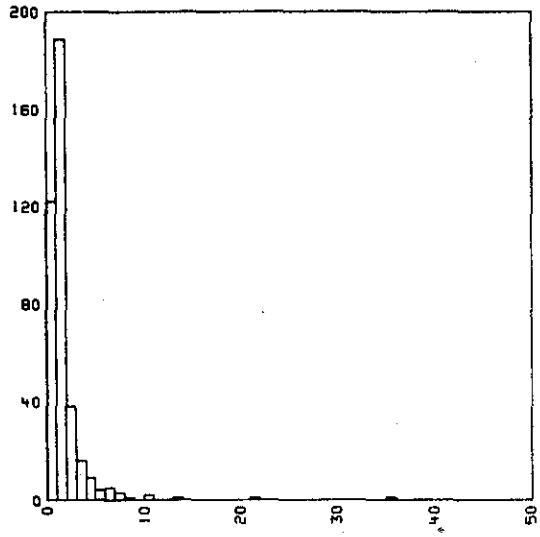


THIRD POINT

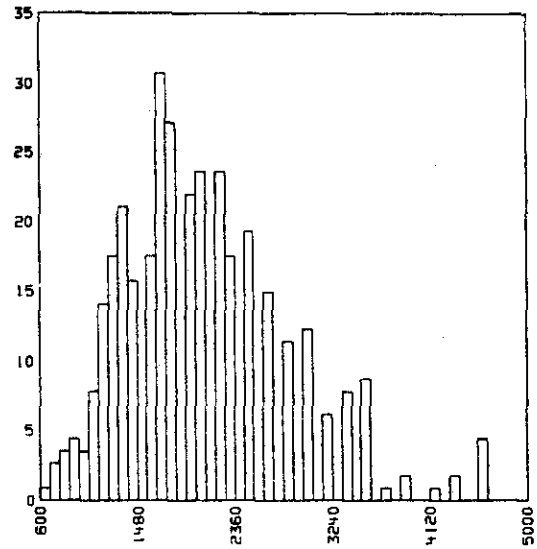


FOURTH POINT

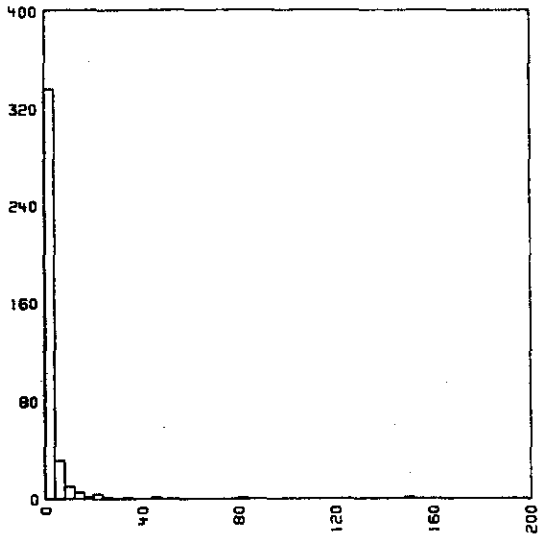
100 POINTS CUBE $\sigma=4$



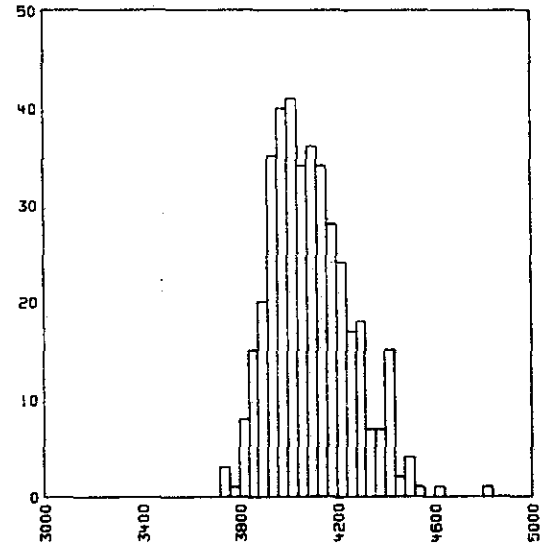
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

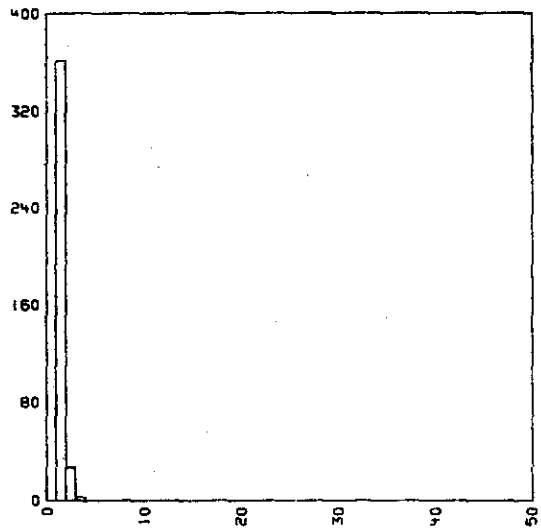


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

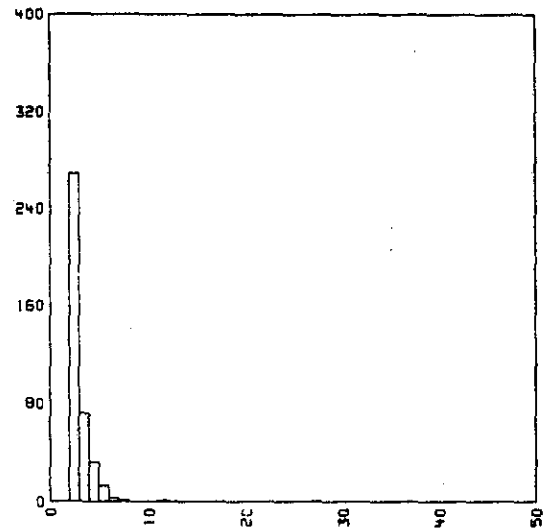


RADIUS

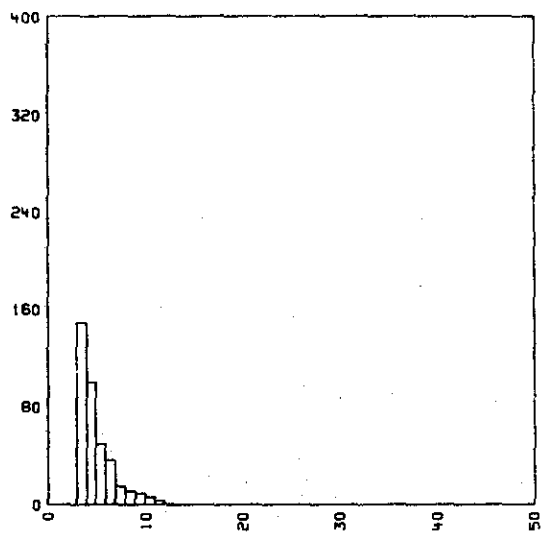
3200 POINTS CUBE $\sigma=4$



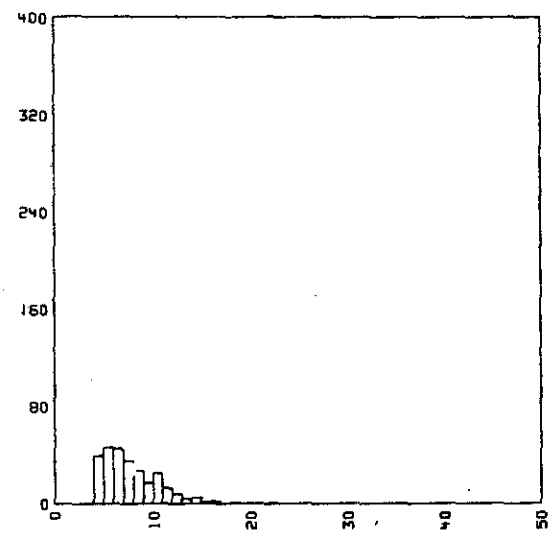
FIRST POINT



SECOND POINT

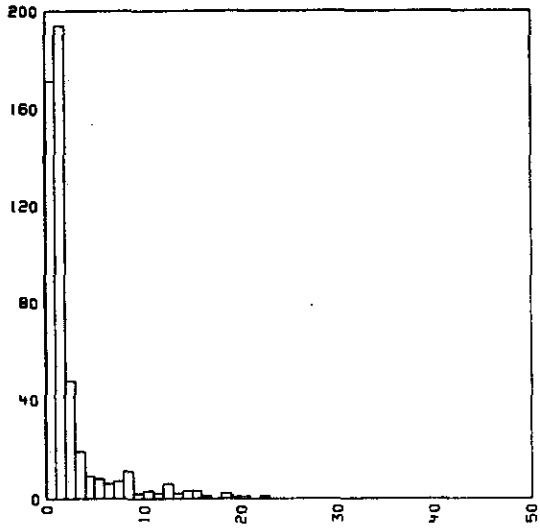


THIRD POINT

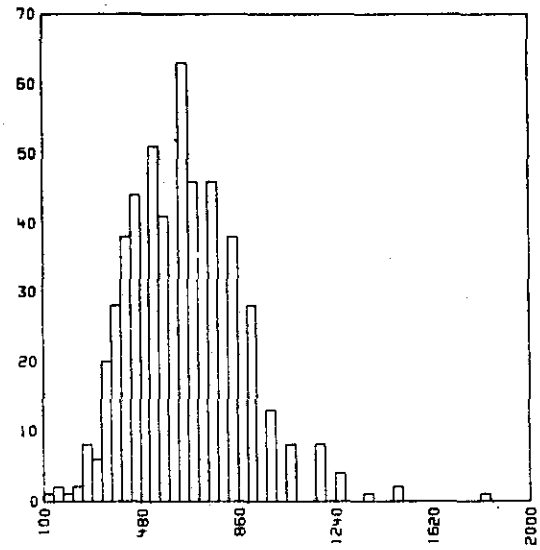


FOURTH POINT

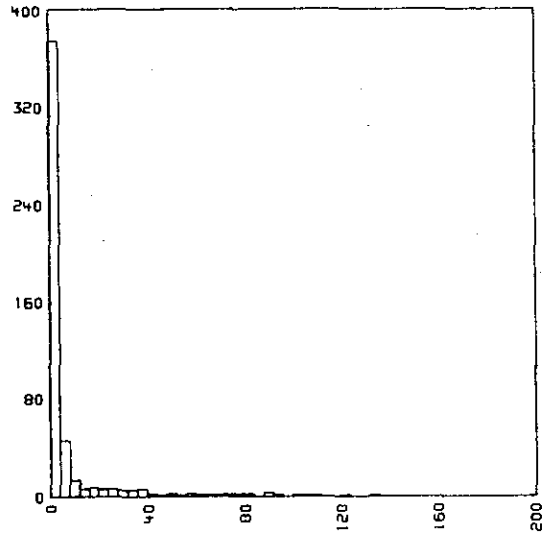
3200 POINTS CUBE $\sigma=4$



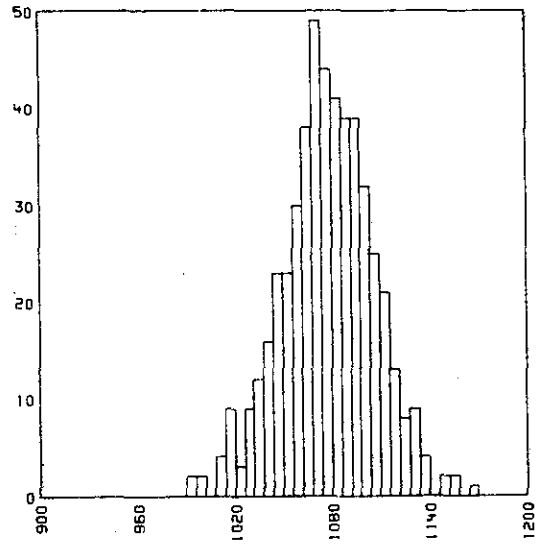
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

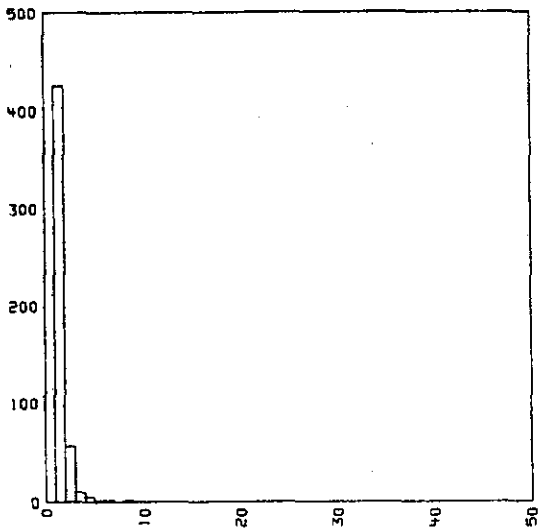


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

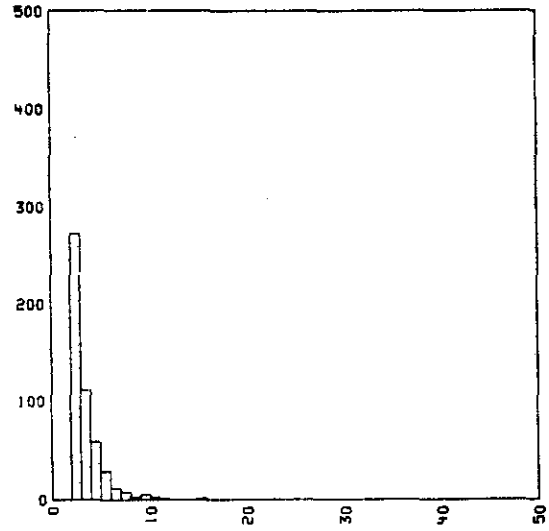


RADIUS

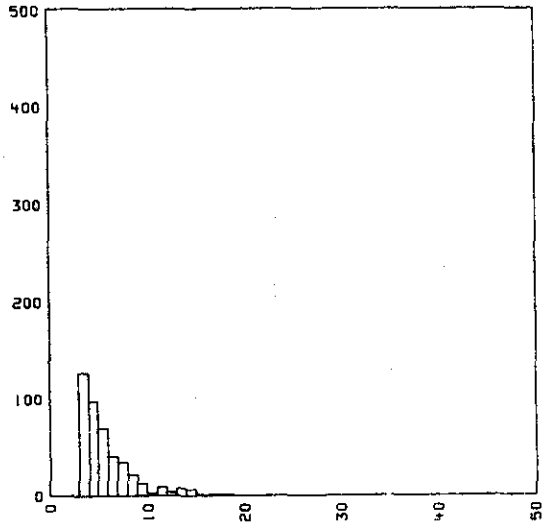
100 POINTS POLYHEDRON



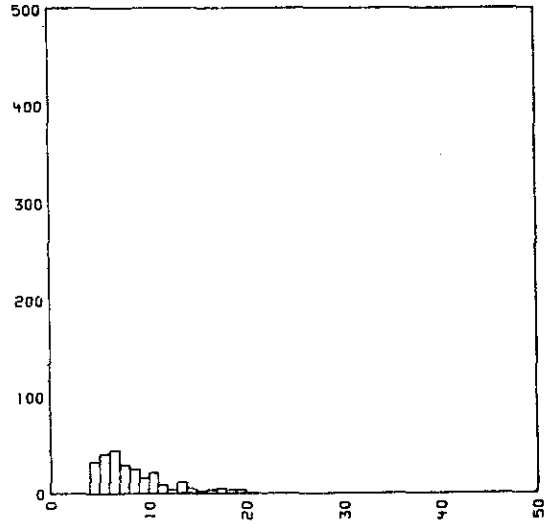
FIRST POINT



SECOND POINT

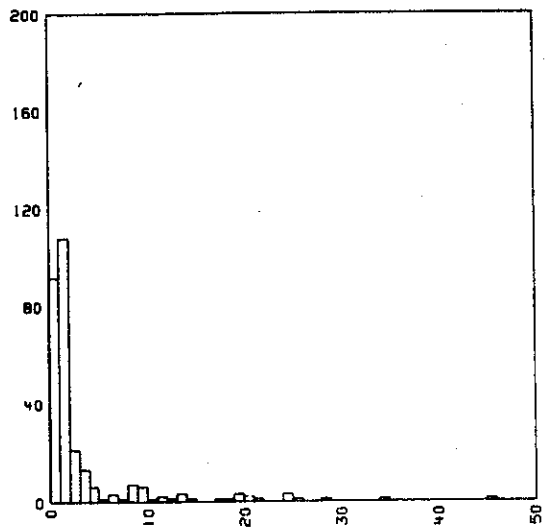


THIRD POINT

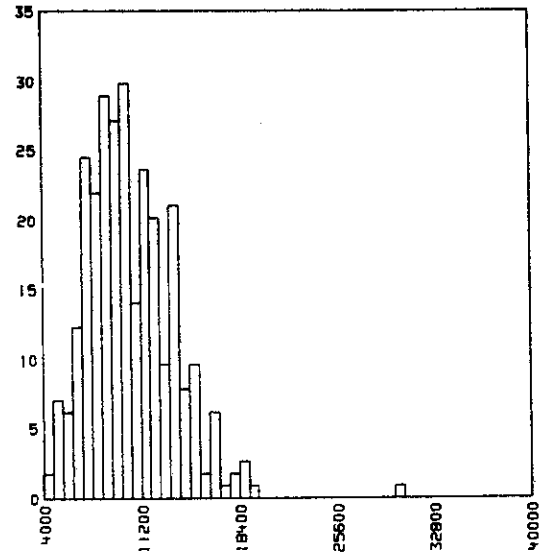


FOURTH POINT

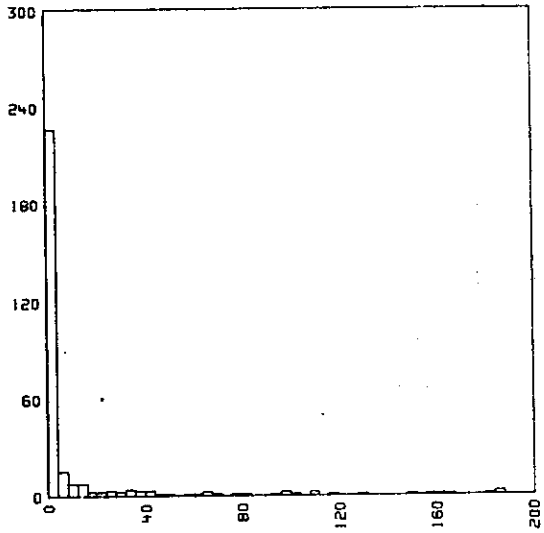
100 POINTS POLYHEDRON



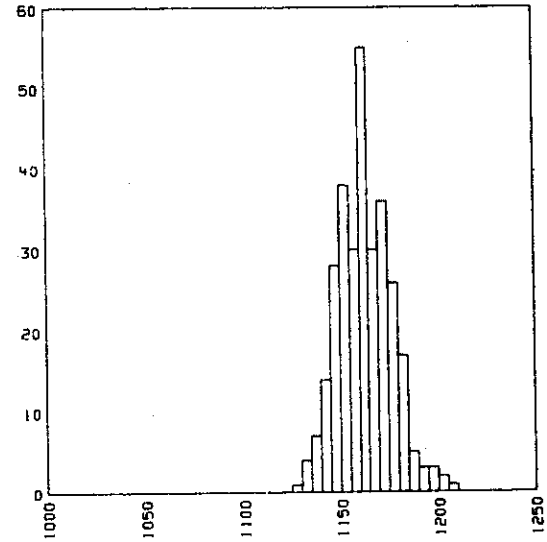
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

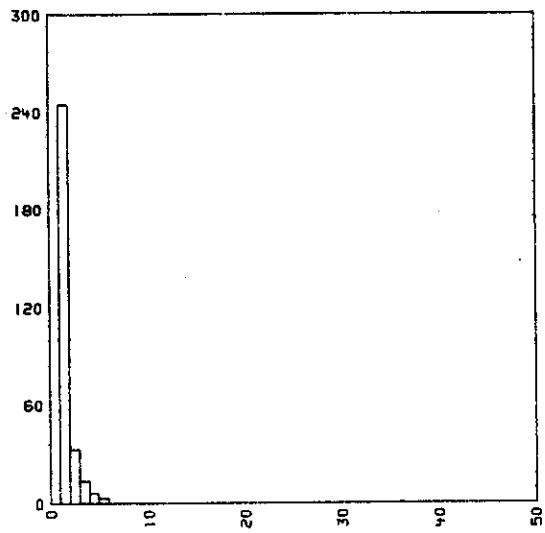


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

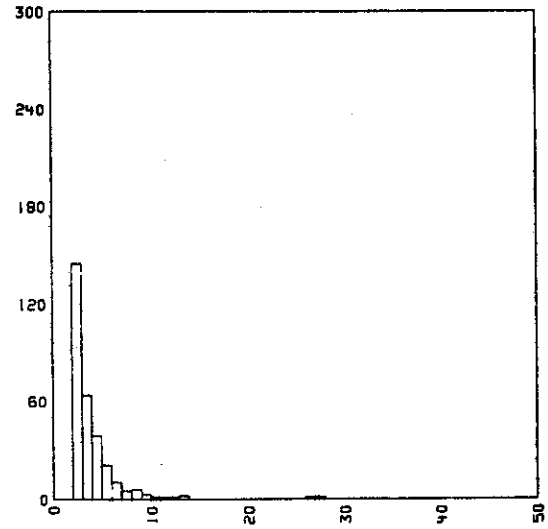


RADIUS

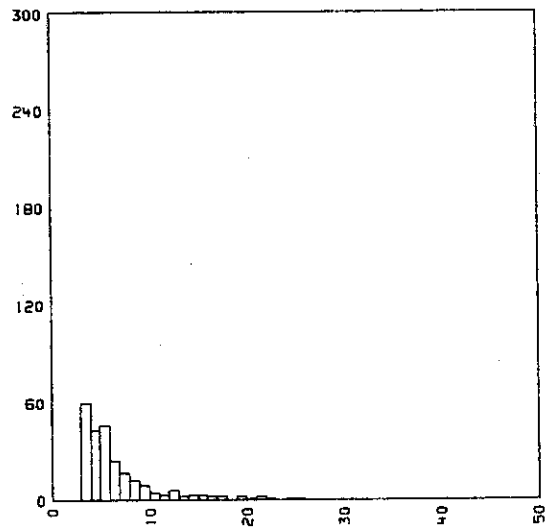
1600 POINTS POLYHEDRON



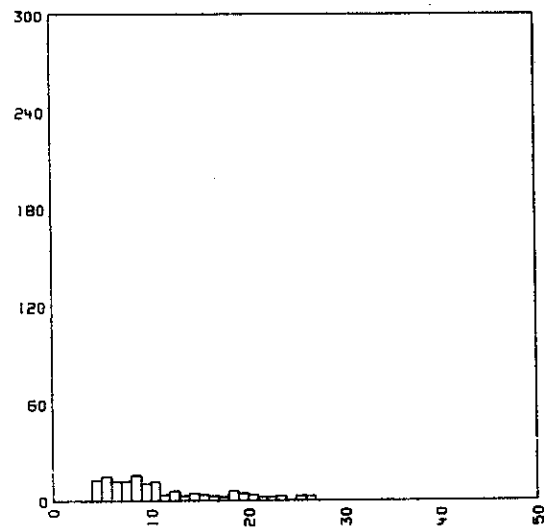
FIRST POINT



SECOND POINT

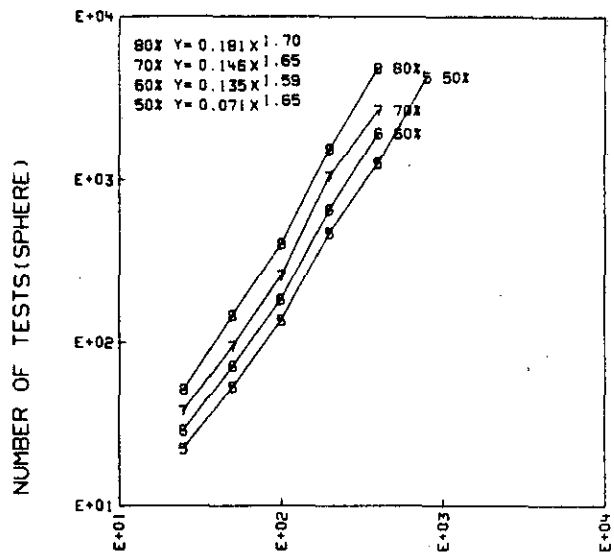


THIRD POINT

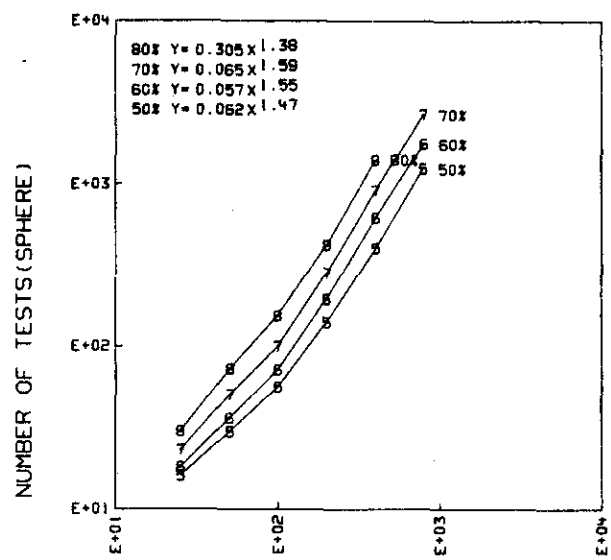


FOURTH POINT

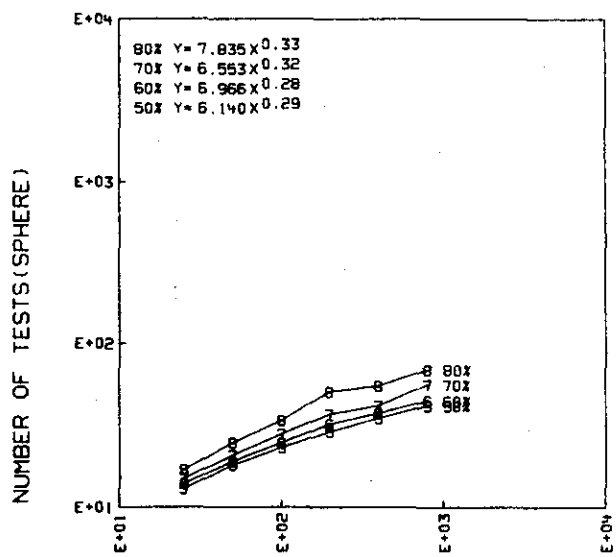
1600 POINTS POLYHEDRON



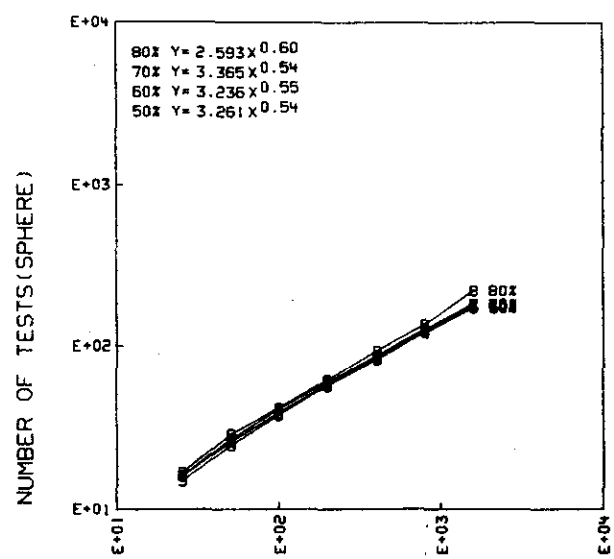
NUMBER OF ORIGINAL DATA POINTS
SPHERE σ=1



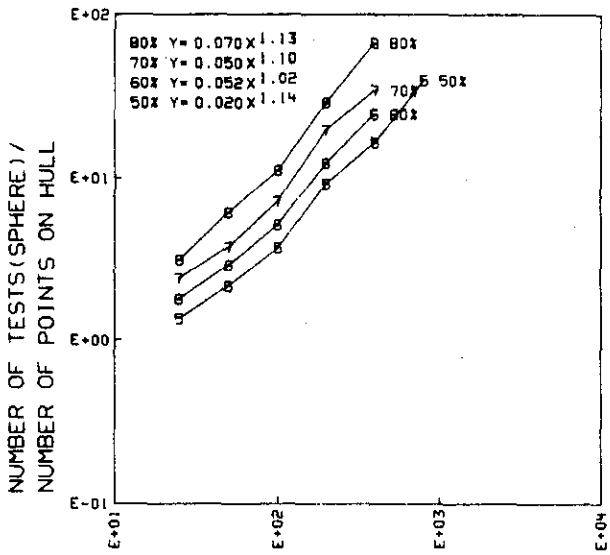
NUMBER OF ORIGINAL DATA POINTS
SPHERE σ=2

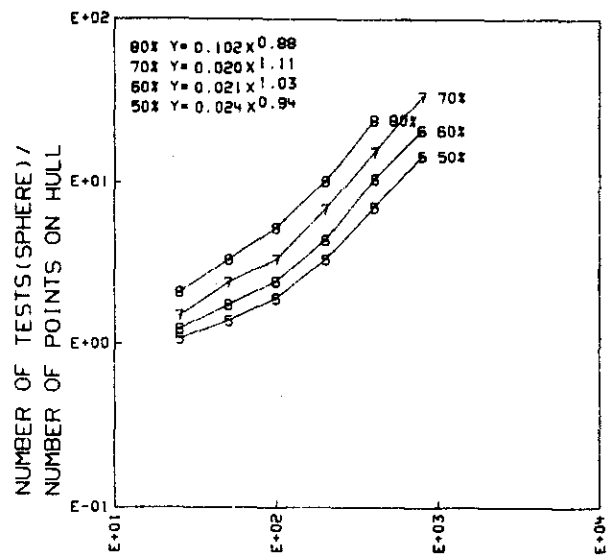


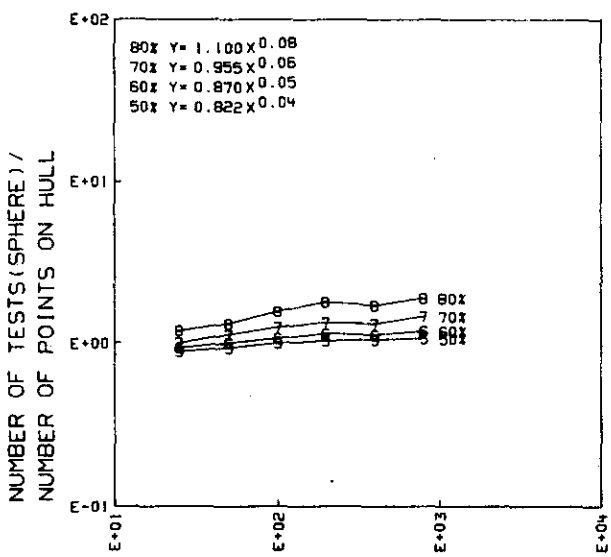
NUMBER OF ORIGINAL DATA POINTS
SPHERE σ=4

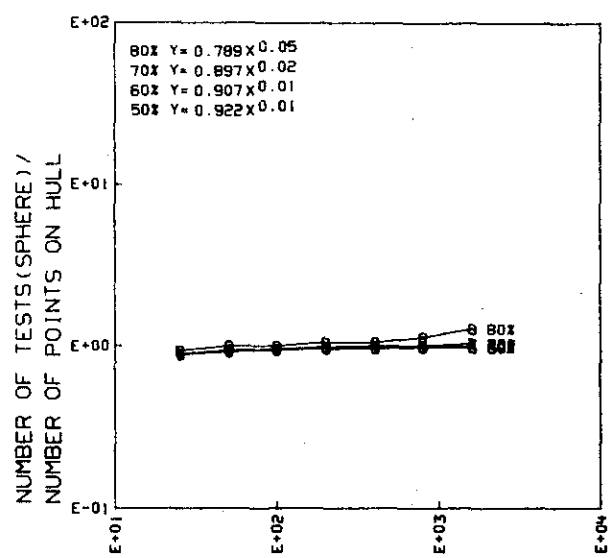


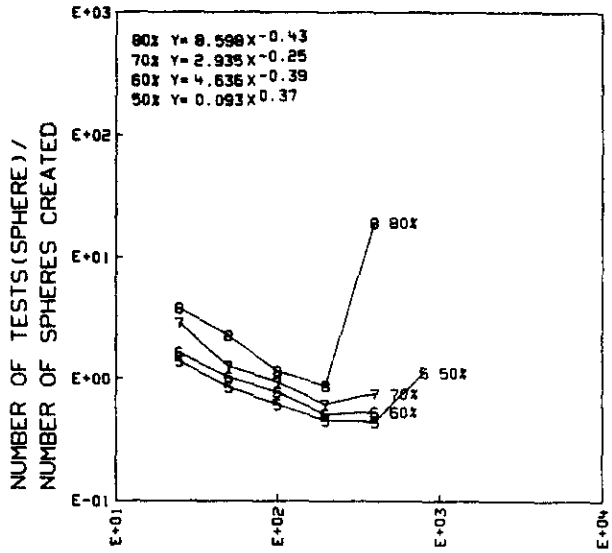
NUMBER OF ORIGINAL DATA POINTS
ELIPSE


80% Y= 0.070X^1.13
70% Y= 0.050X^1.10
60% Y= 0.052X^1.02
50% Y= 0.020X^1.14
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL
NUMBER OF ORIGINAL DATA POINTS
SPHERE σ=1

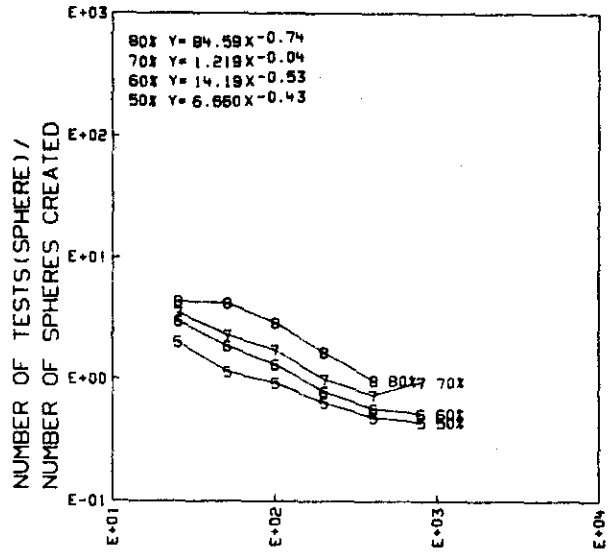

80% Y= 0.102X^0.88
70% Y= 0.020X^1.11
60% Y= 0.021X^1.03
50% Y= 0.024X^0.94
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL
NUMBER OF ORIGINAL DATA POINTS
SPHERE σ=2


80% Y= 1.100X^0.08
70% Y= 0.955X^0.06
60% Y= 0.870X^0.05
50% Y= 0.822X^0.04
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL
NUMBER OF ORIGINAL DATA POINTS
SPHERE σ=4

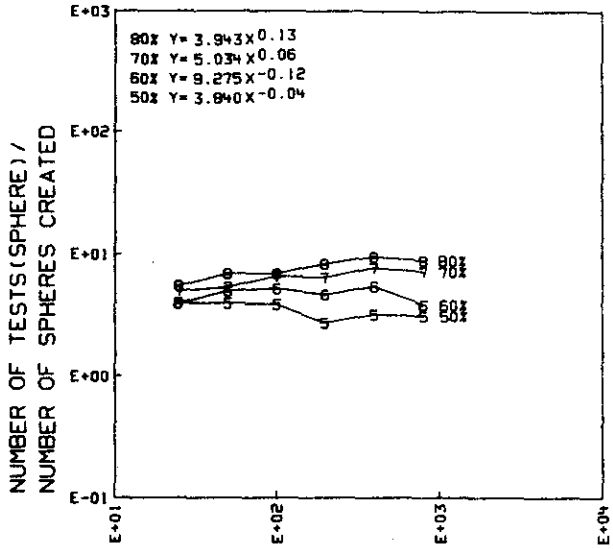

80% Y= 0.789X^0.05
70% Y= 0.897X^0.02
60% Y= 0.907X^0.01
50% Y= 0.922X^0.01
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL
NUMBER OF ORIGINAL DATA POINTS
ELIPSE



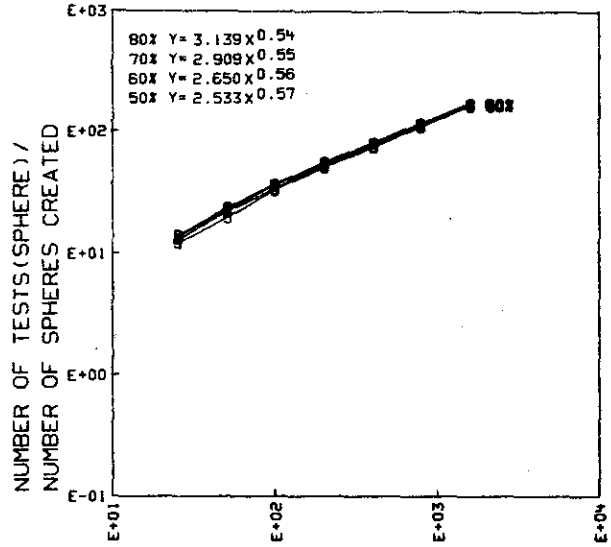
SPHERE σ=1



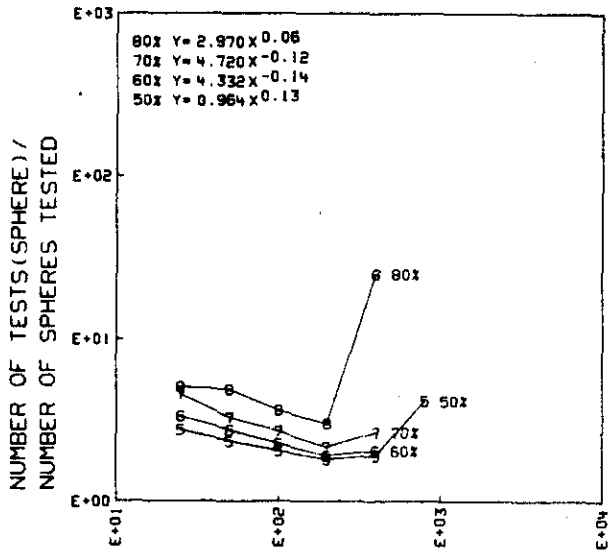
SPHERE σ=2



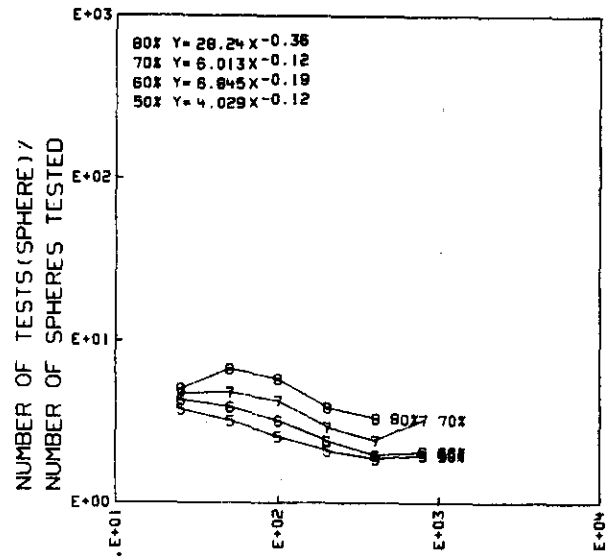
SPHERE σ=4



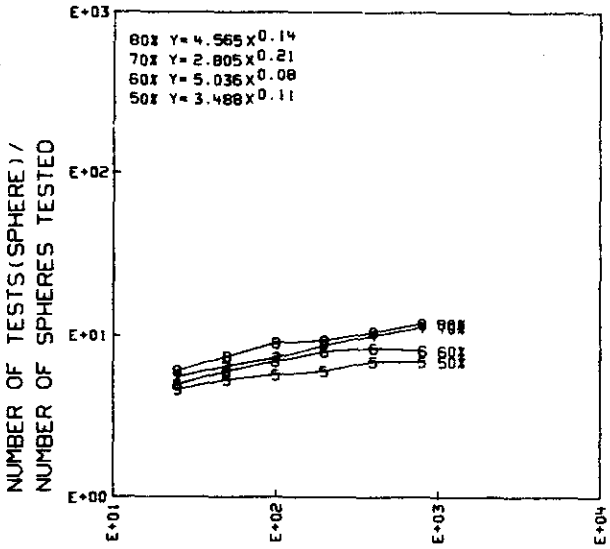
ELIPSE



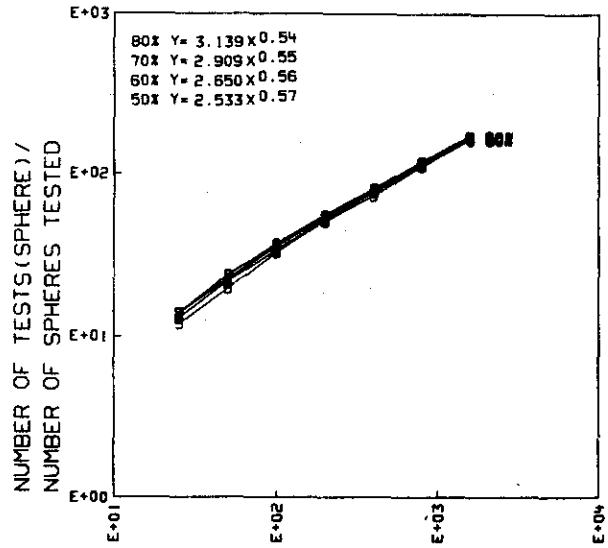
NUMBER OF ORIGINAL DATA POINTS
SPHERE $\sigma=1$



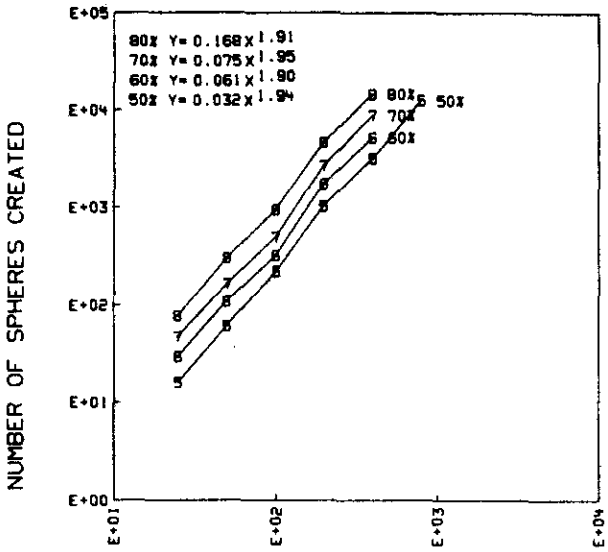
NUMBER OF ORIGINAL DATA POINTS
SPHERE $\sigma=2$



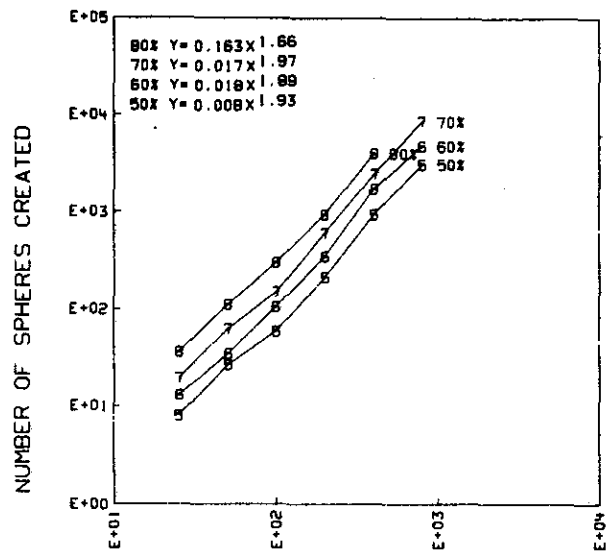
NUMBER OF ORIGINAL DATA POINTS
SPHERE $\sigma=4$



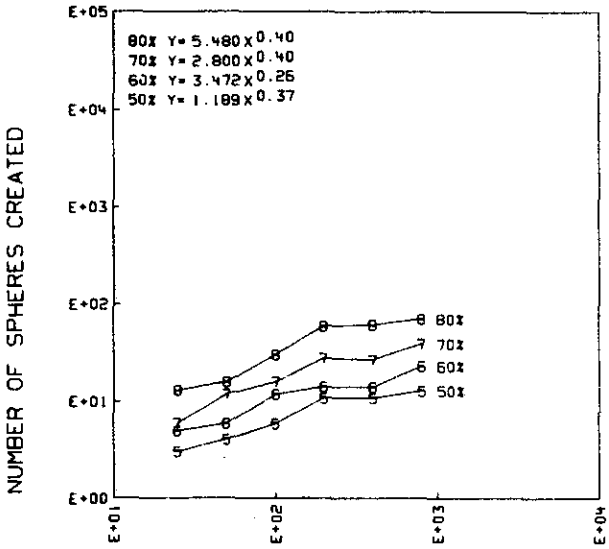
NUMBER OF ORIGINAL DATA POINTS
ELIPSE



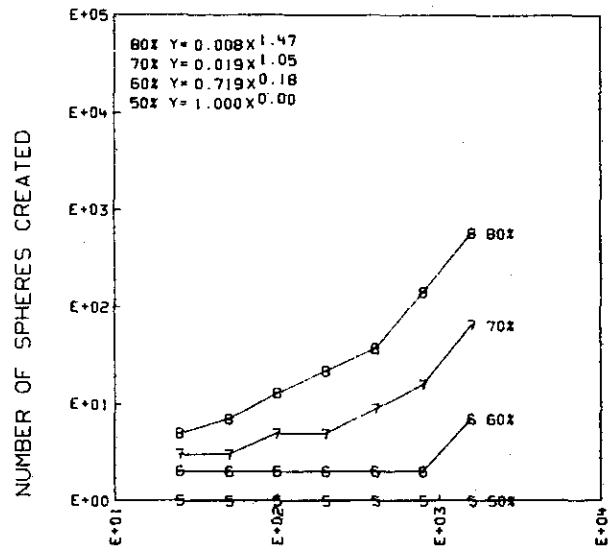
NUMBER OF ORIGINAL DATA POINTS
SPHERE $\sigma=1$



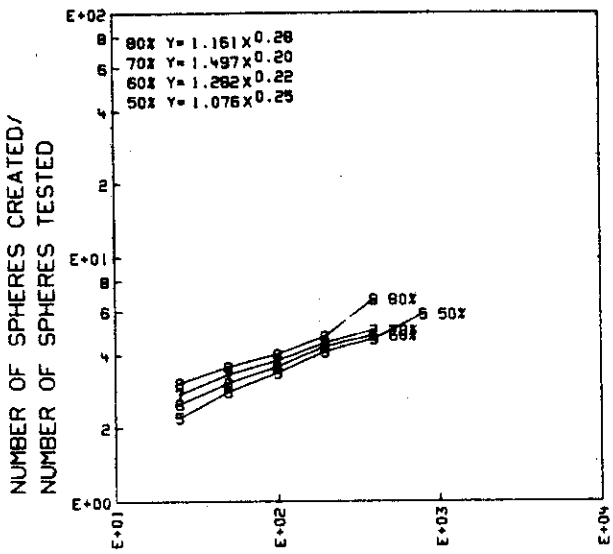
NUMBER OF ORIGINAL DATA POINTS
SPHERE $\sigma=2$



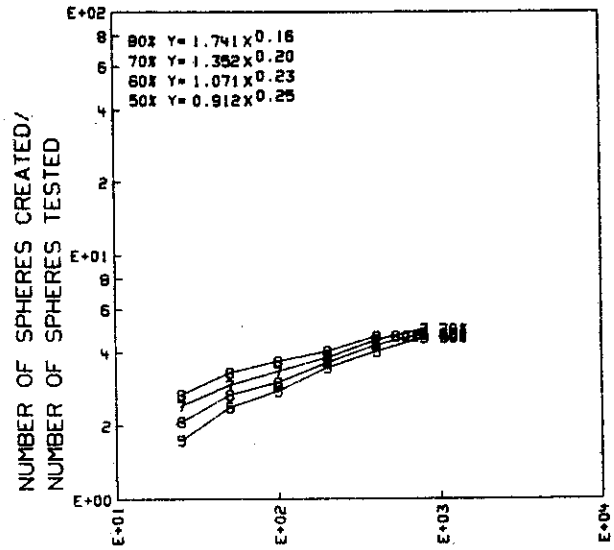
NUMBER OF ORIGINAL DATA POINTS
SPHERE $\sigma=4$



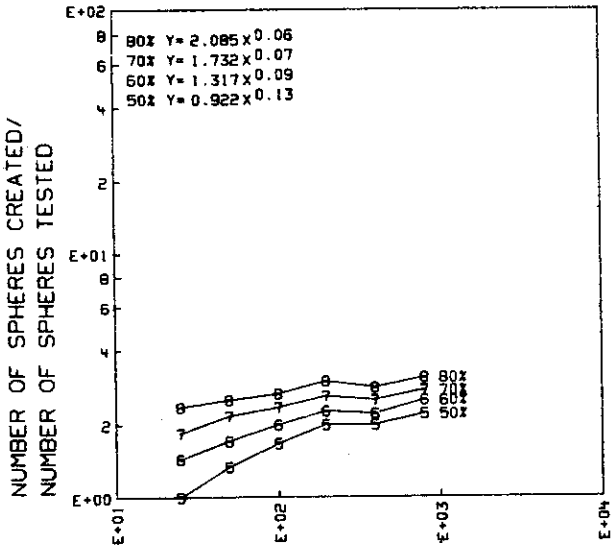
NUMBER OF ORIGINAL DATA POINTS
ELIPSE



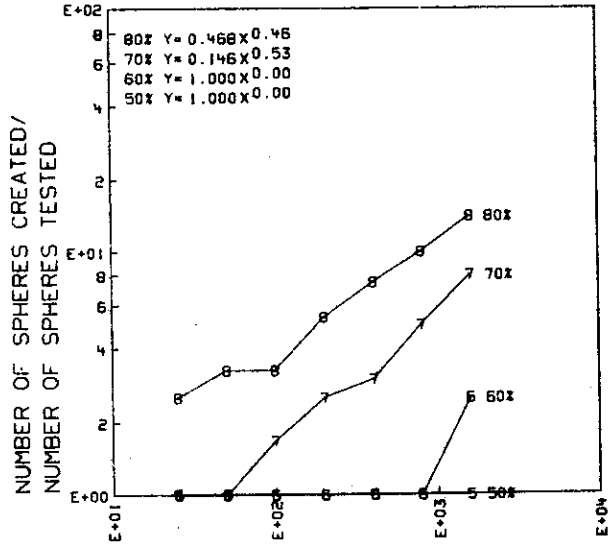
SPHERE σ=1



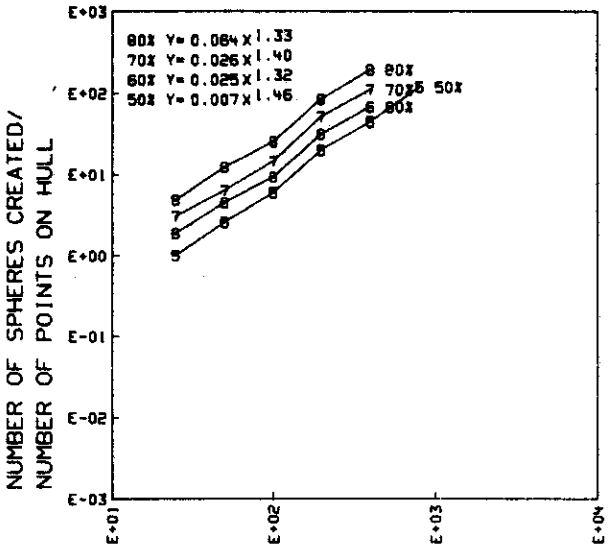
SPHERE σ=2



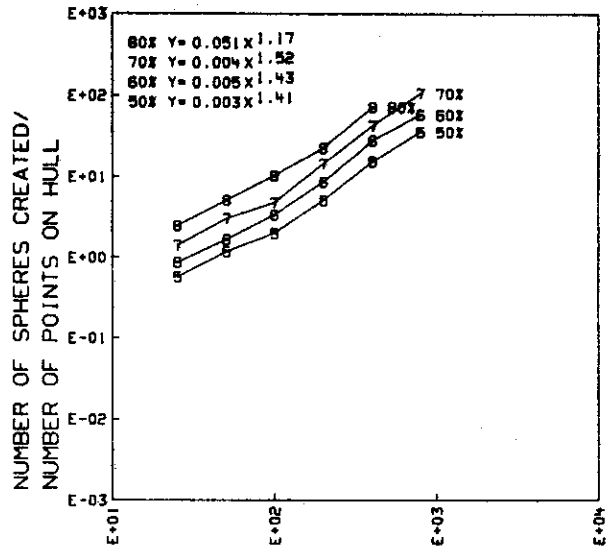
SPHERE σ=4



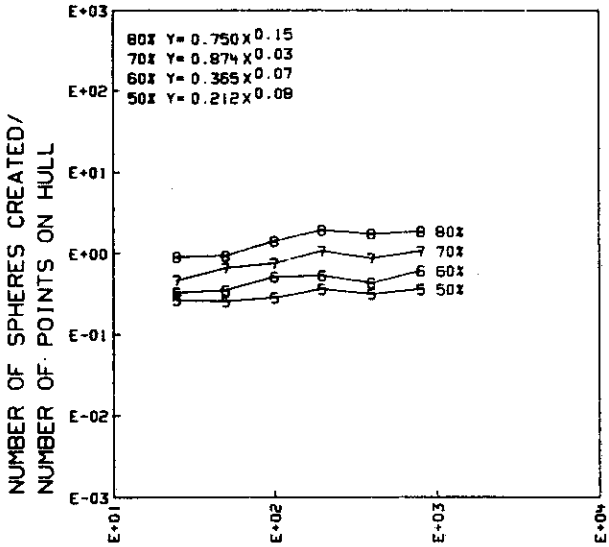
ELIPSE



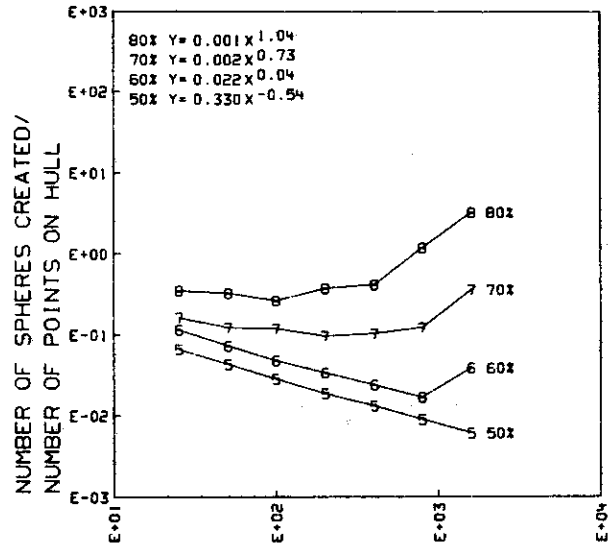
NUMBER OF ORIGINAL DATA POINTS
SPHERE $\sigma=1$



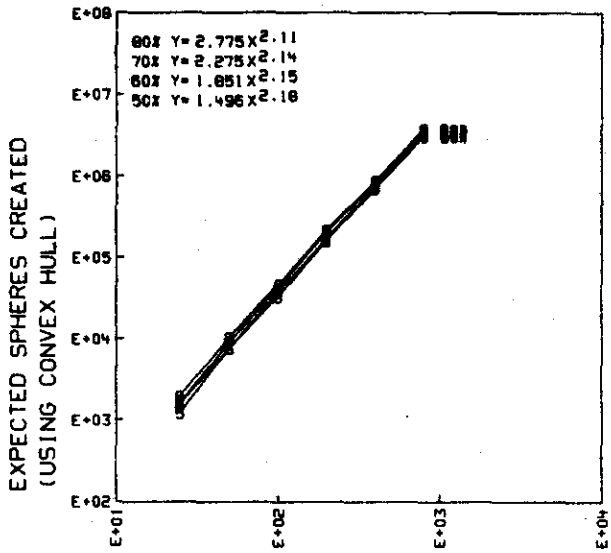
NUMBER OF ORIGINAL DATA POINTS
SPHERE $\sigma=2$



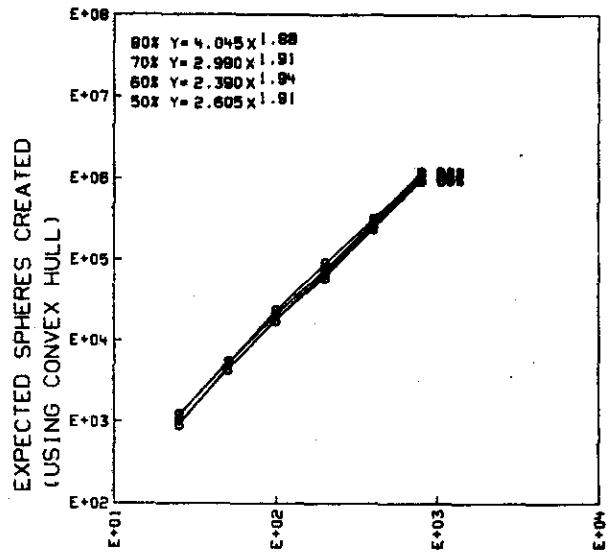
NUMBER OF ORIGINAL DATA POINTS
SPHERE $\sigma=4$



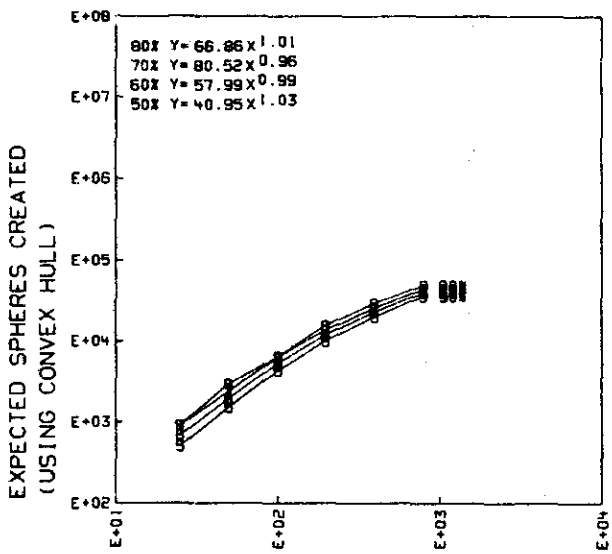
NUMBER OF ORIGINAL DATA POINTS
ELIPSE



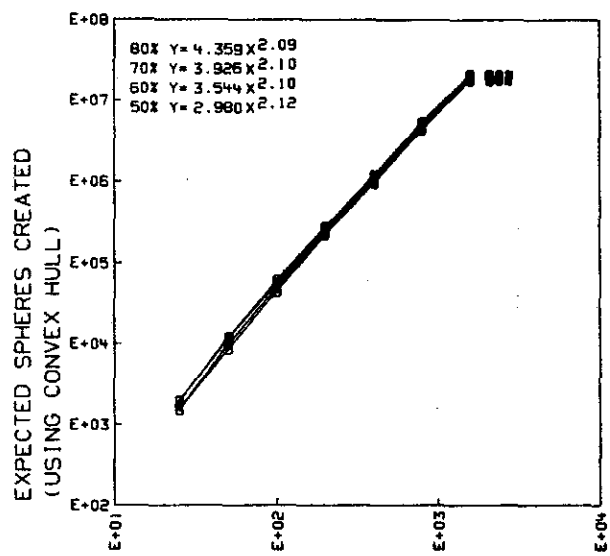
NUMBER OF ORIGINAL DATA POINTS
SPHERE $\sigma = 1$



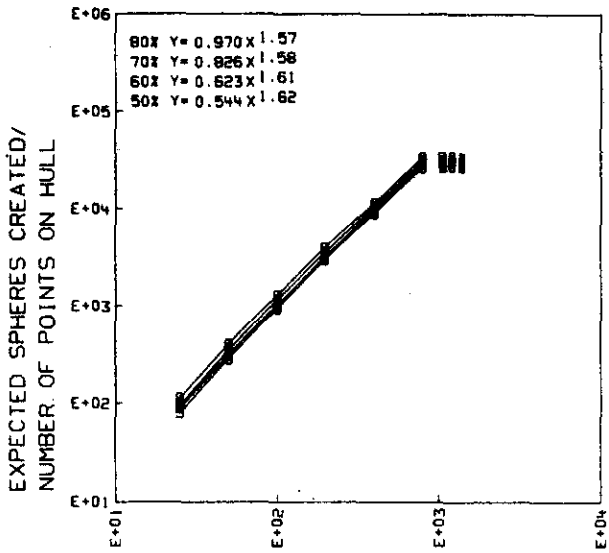
NUMBER OF ORIGINAL DATA POINTS
SPHERE $\sigma = 2$



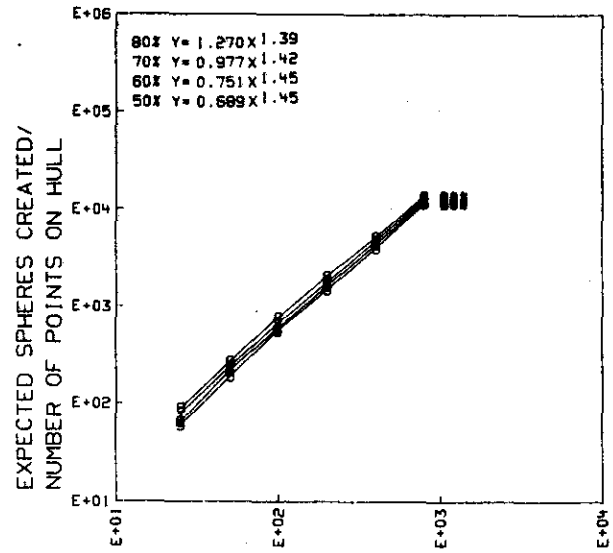
NUMBER OF ORIGINAL DATA POINTS
SPHERE $\sigma = 4$



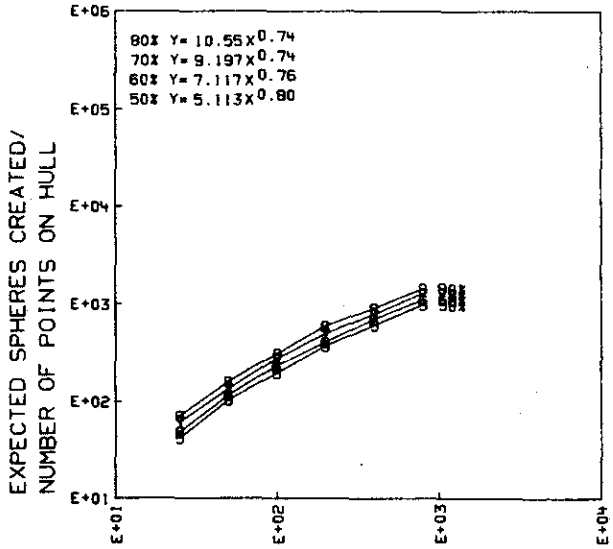
NUMBER OF ORIGINAL DATA POINTS
ELIPSE



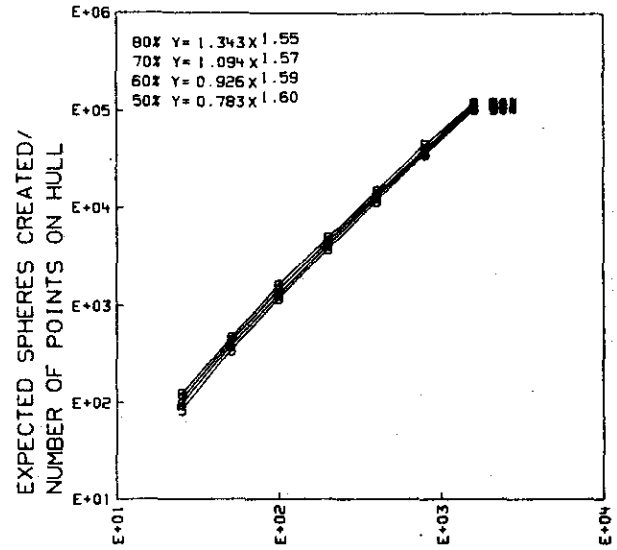
NUMBER OF ORIGINAL DATA POINTS
SPHERE σ=1



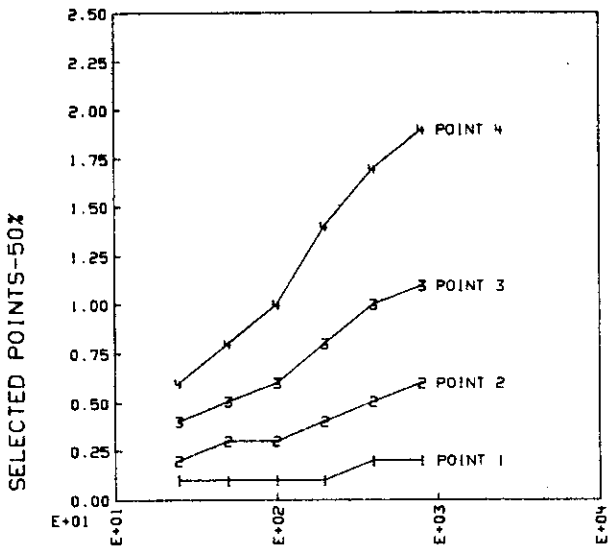
NUMBER OF ORIGINAL DATA POINTS
SPHERE σ=2



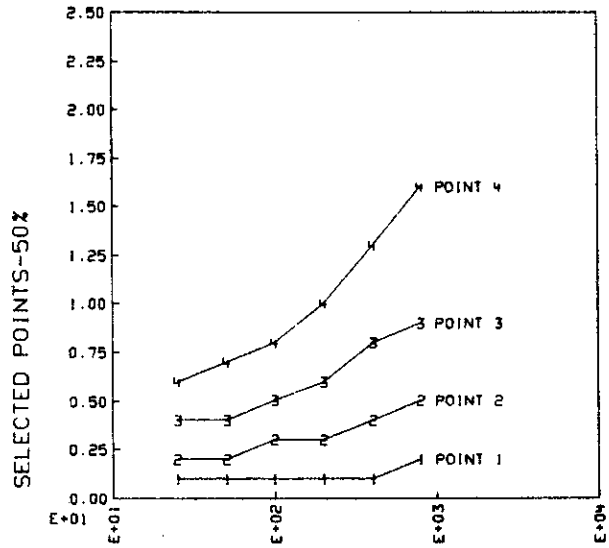
NUMBER OF ORIGINAL DATA POINTS
SPHERE σ=4



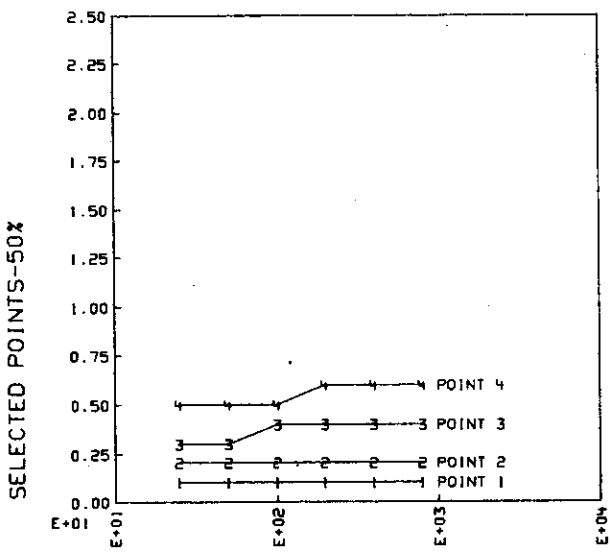
NUMBER OF ORIGINAL DATA POINTS
ELIPSE



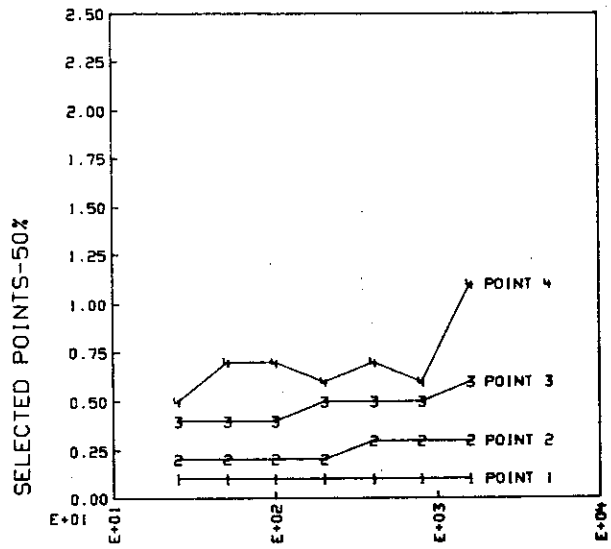
NUMBER OF ORIGINAL DATA POINTS
SPHERE $\sigma=1$



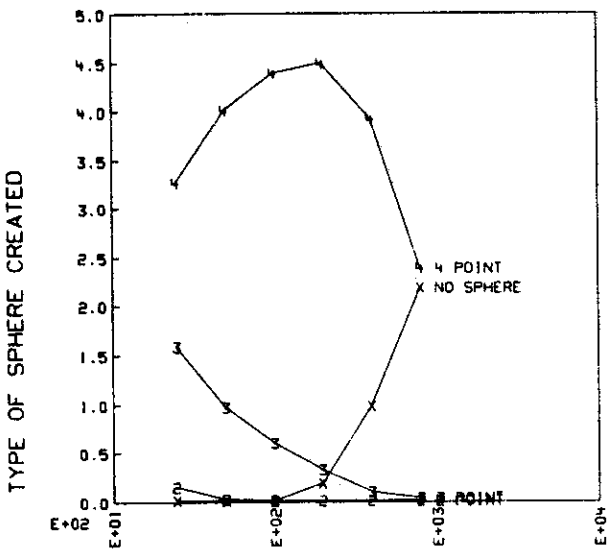
NUMBER OF ORIGINAL DATA POINTS
SPHERE $\sigma=2$

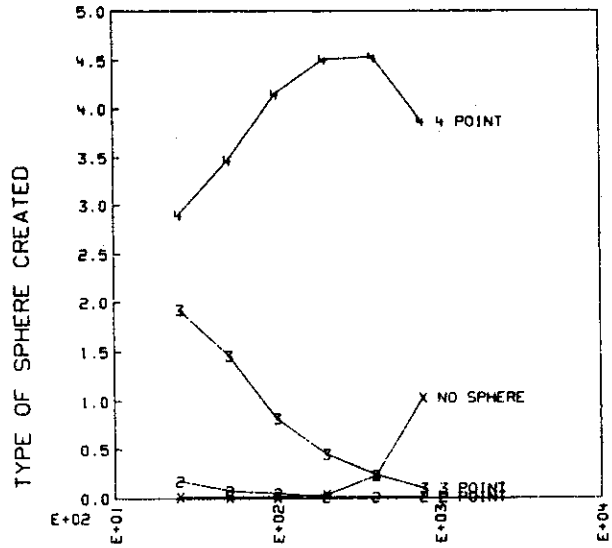


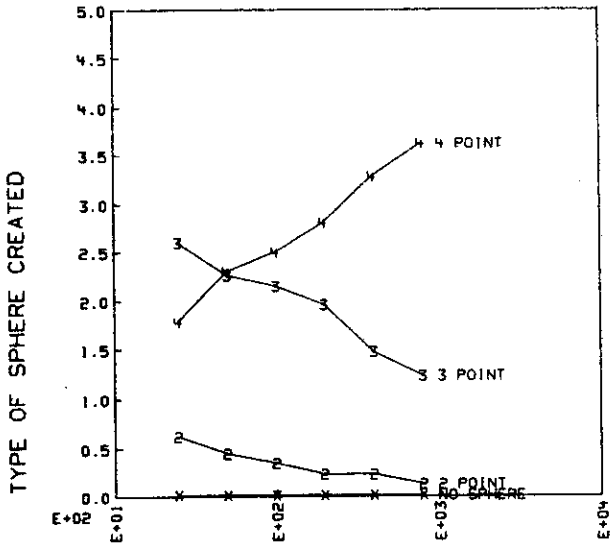
NUMBER OF ORIGINAL DATA POINTS
SPHERE $\sigma=4$

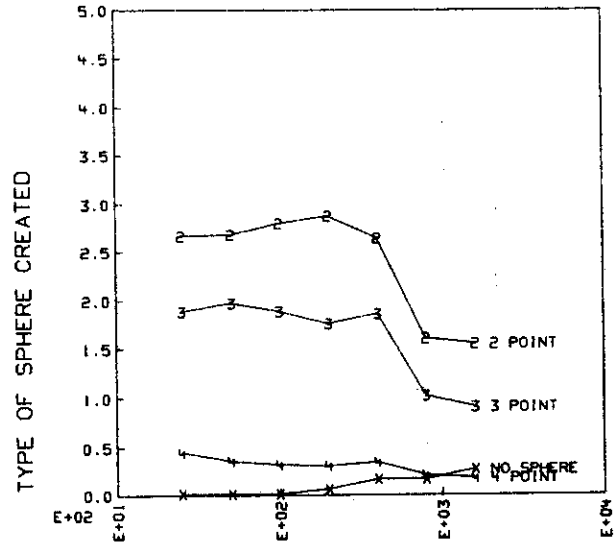


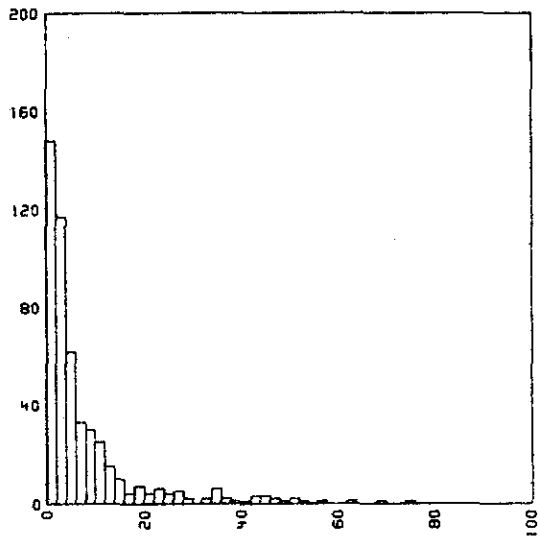
NUMBER OF ORIGINAL DATA POINTS
ELIPSE


TYPE OF SPHERE CREATED
4 POINT
X NO SPHERE
NUMBER OF ORIGINAL DATA POINTS
SPHERE σ=1

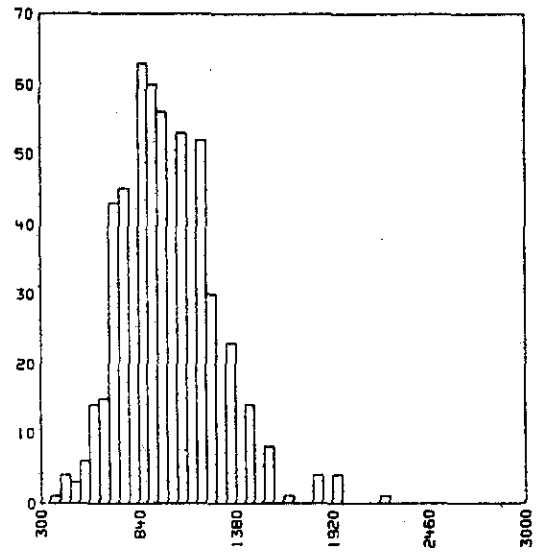

TYPE OF SPHERE CREATED
4 POINT
X NO SPHERE
NUMBER OF ORIGINAL DATA POINTS
SPHERE σ=2


TYPE OF SPHERE CREATED
4 POINT
3 3 POINT
NUMBER OF ORIGINAL DATA POINTS
SPHERE σ=4

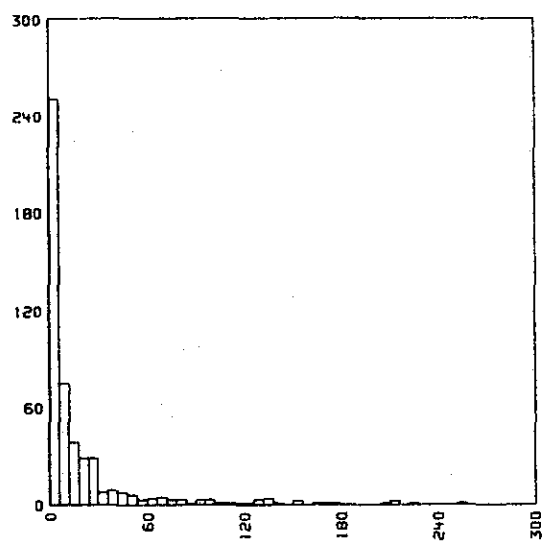

TYPE OF SPHERE CREATED
2 2 POINT
3 3 POINT
NUMBER OF ORIGINAL DATA POINTS
ELIPSE



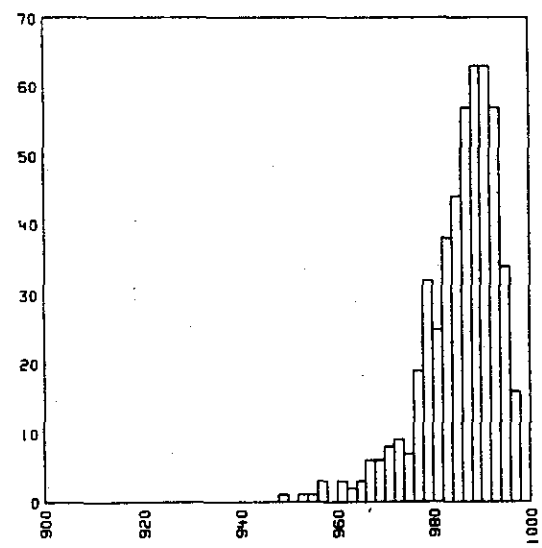
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

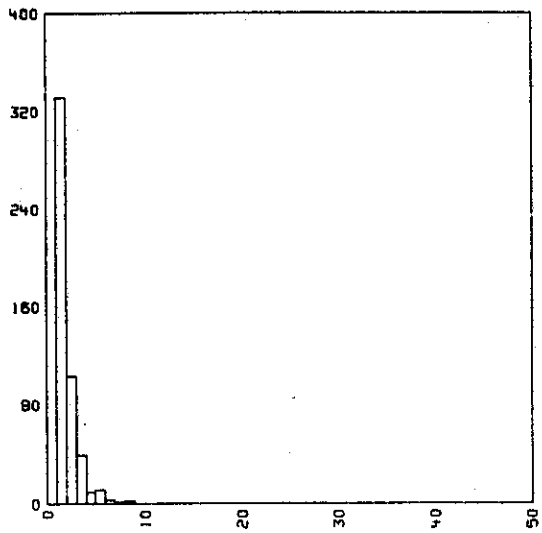


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

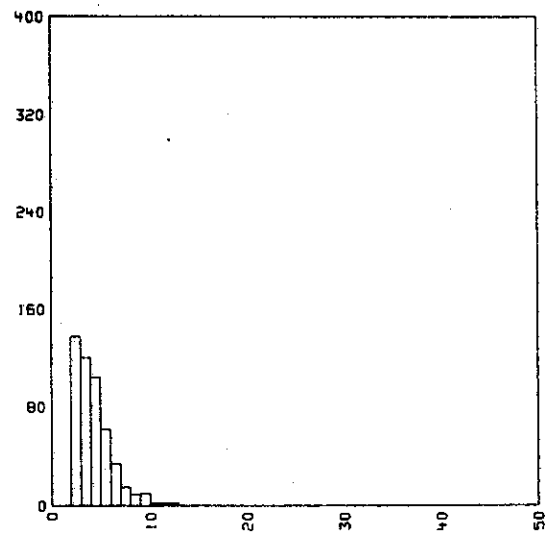


RADIUS

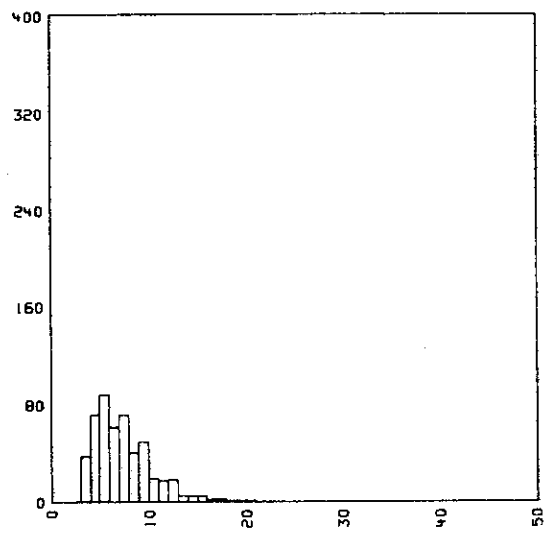
100 POINTS SPHERE $\sigma=1$



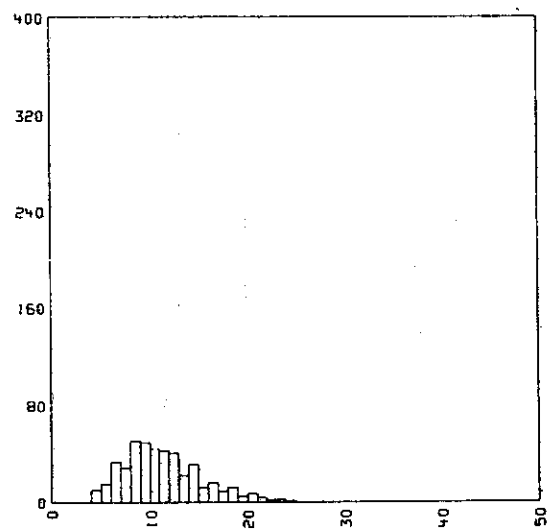
FIRST POINT



SECOND POINT

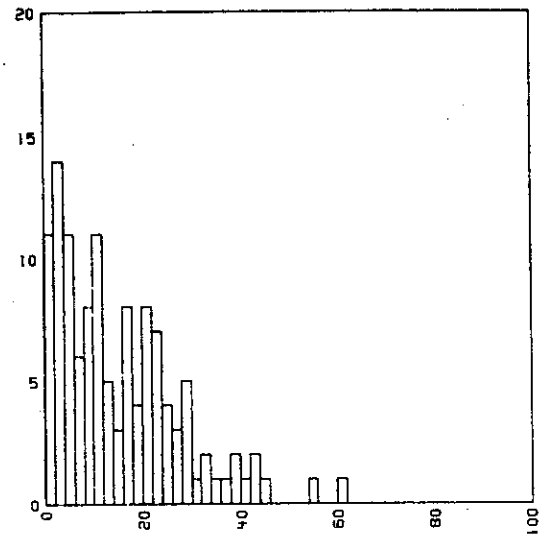


THIRD POINT

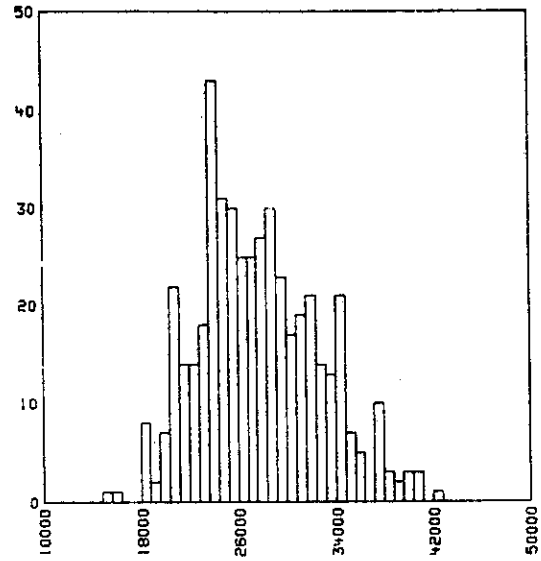


FOURTH POINT

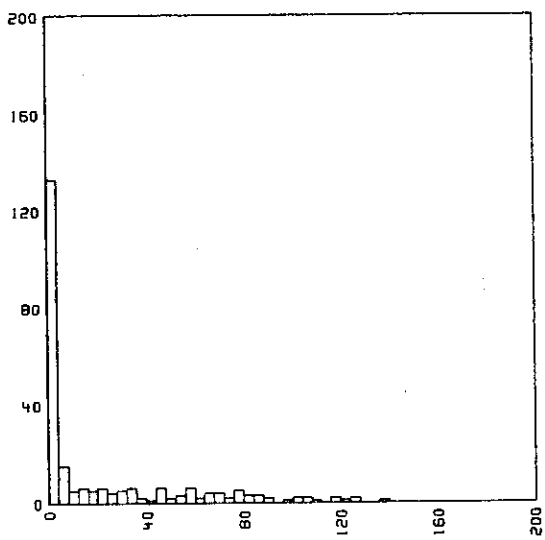
100 POINTS SPHERE $\sigma = 1$



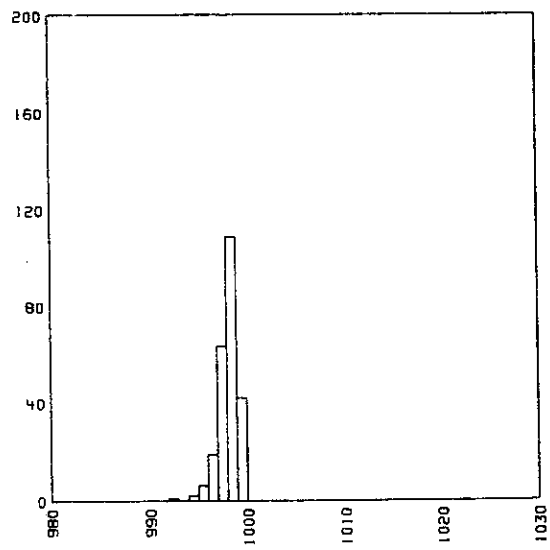
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

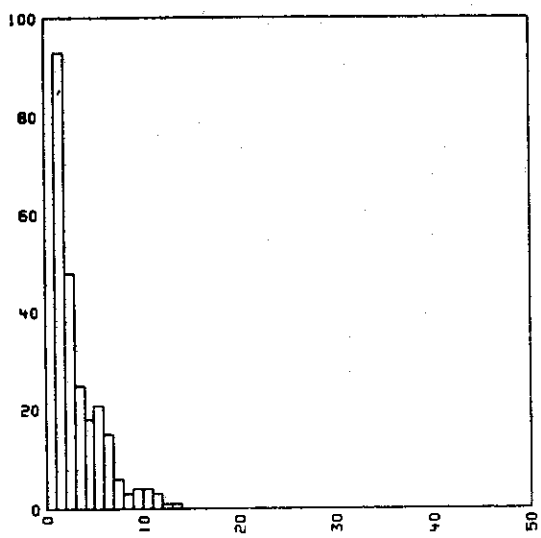


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

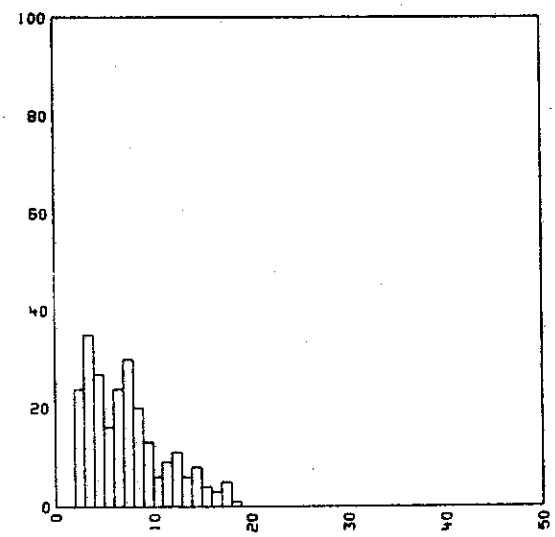


RADIUS

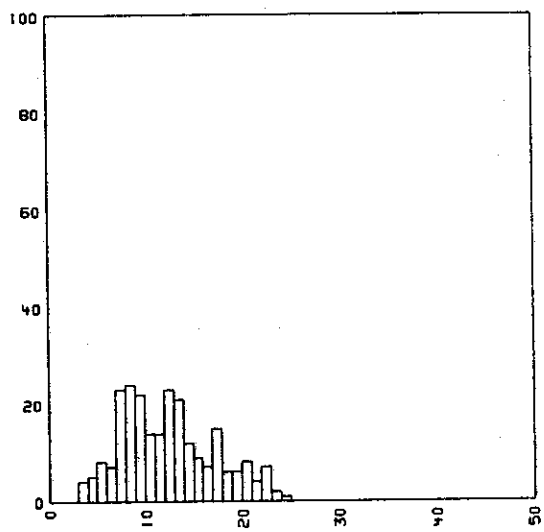
800 POINTS SPHERE $\sigma = 1$



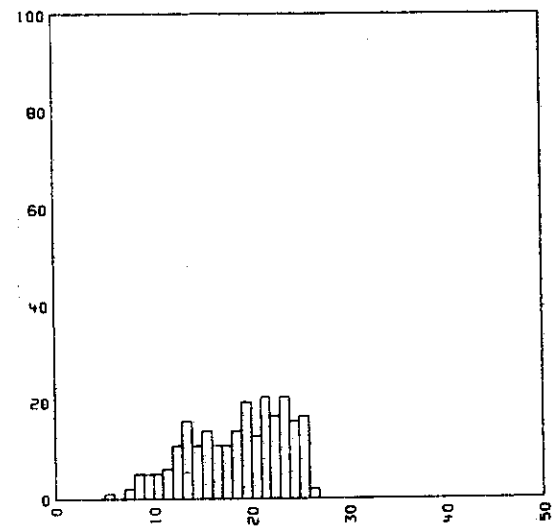
FIRST POINT



SECOND POINT

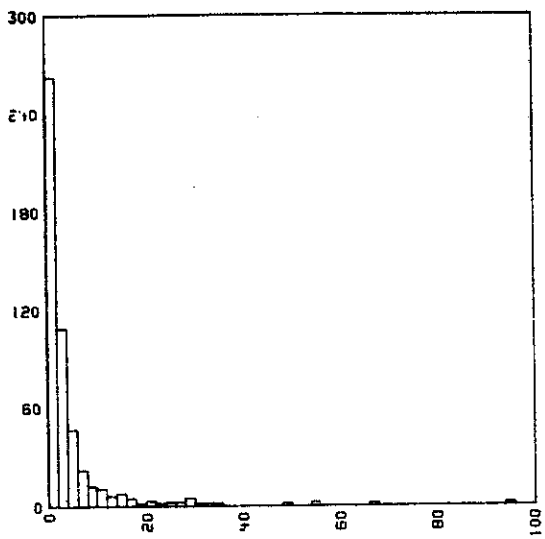


THIRD POINT

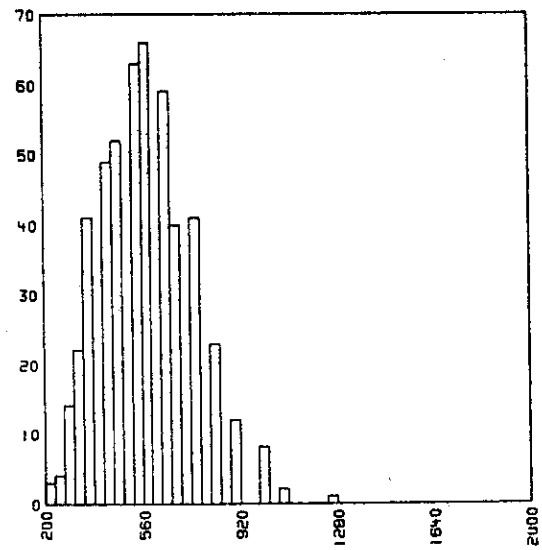


FOURTH POINT

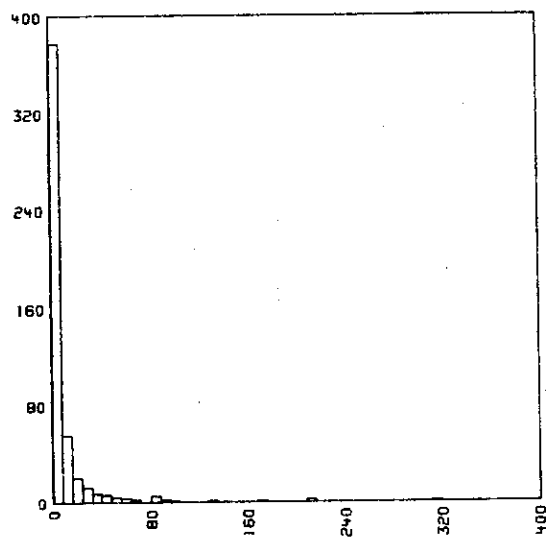
800 POINTS SPHERE $\sigma = 1$



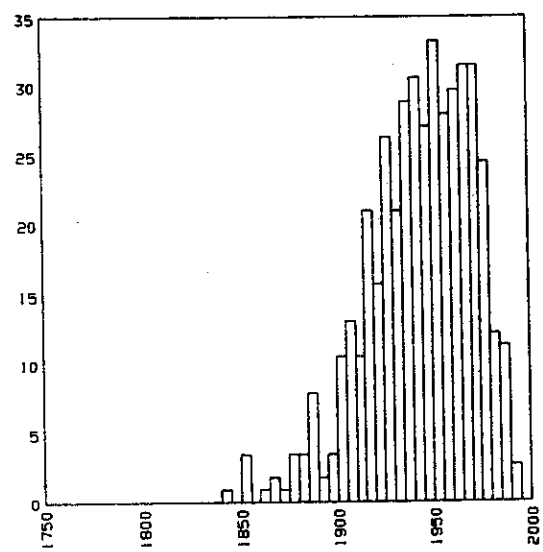
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

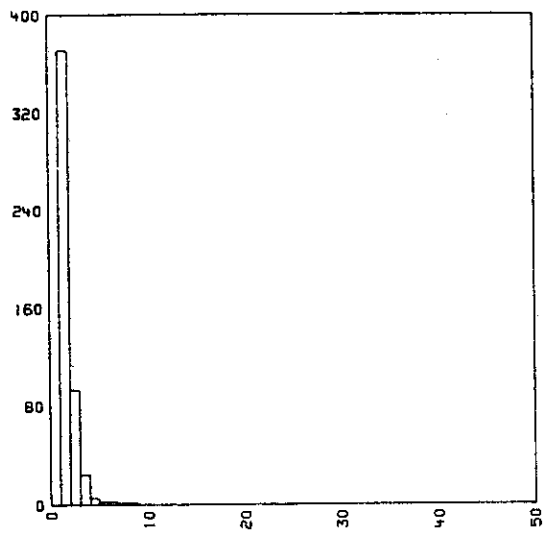


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

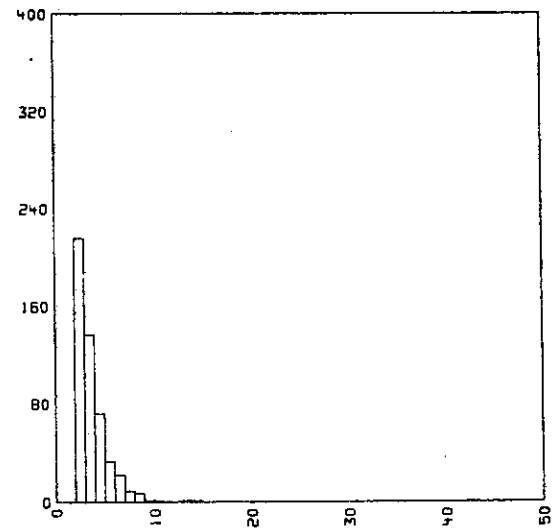


RADIUS

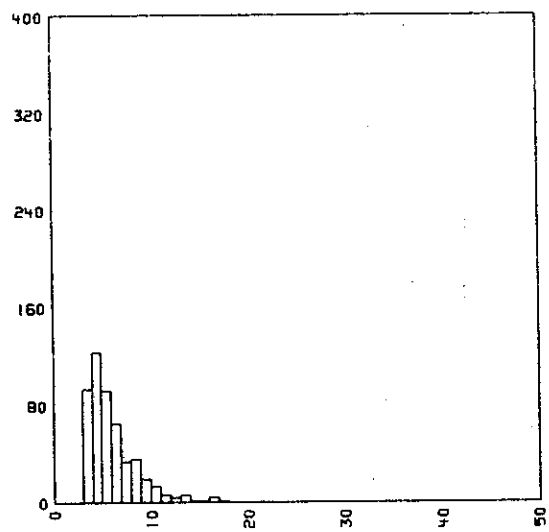
100 POINTS SPHERE $\sigma=2$



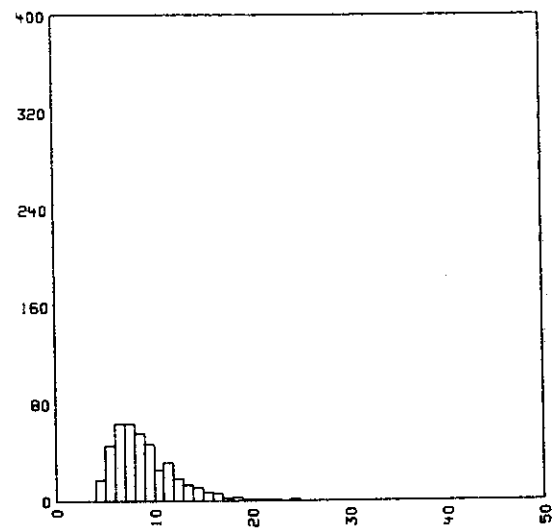
FIRST POINT



SECOND POINT

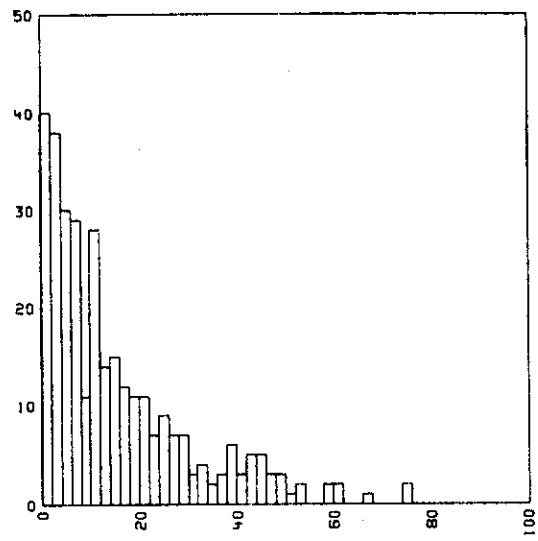


THIRD POINT

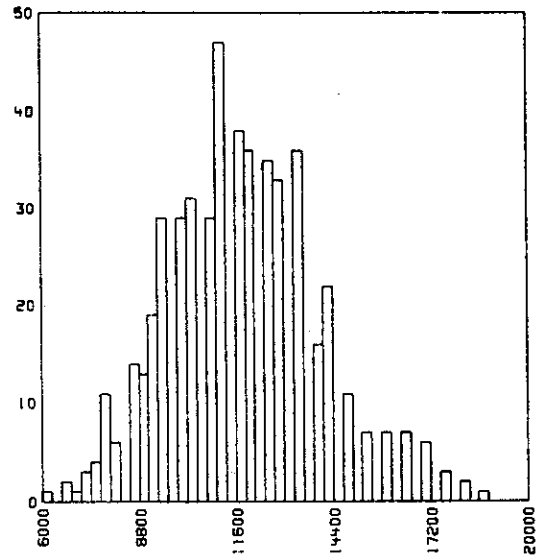


FOURTH POINT

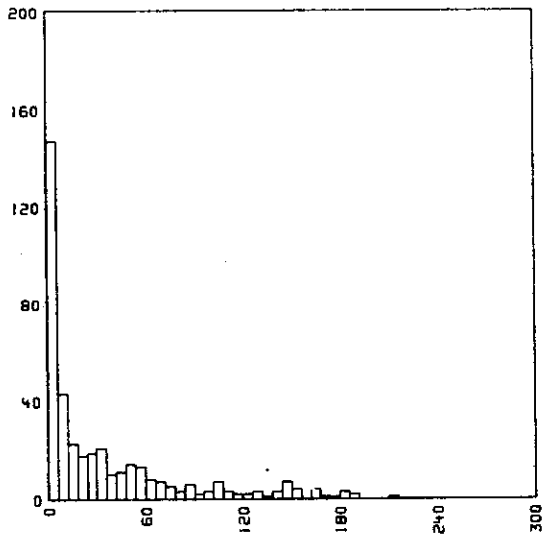
100 POINTS SPHERE $\sigma=2$



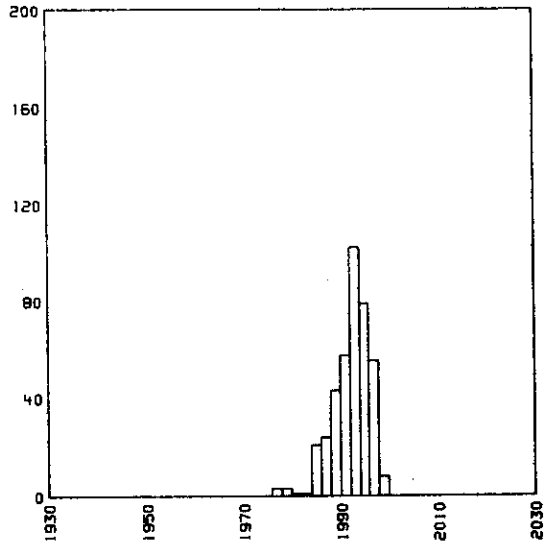
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

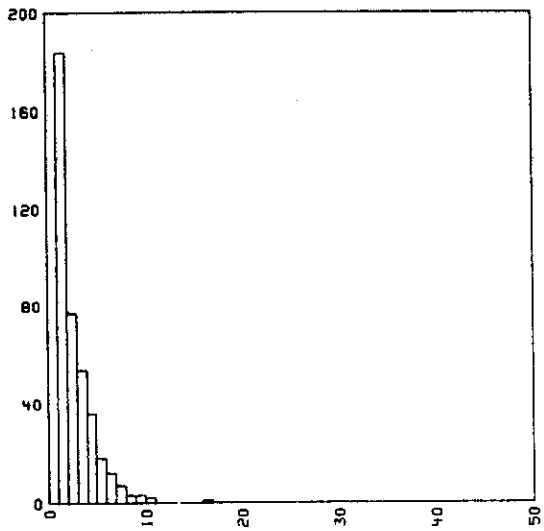


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

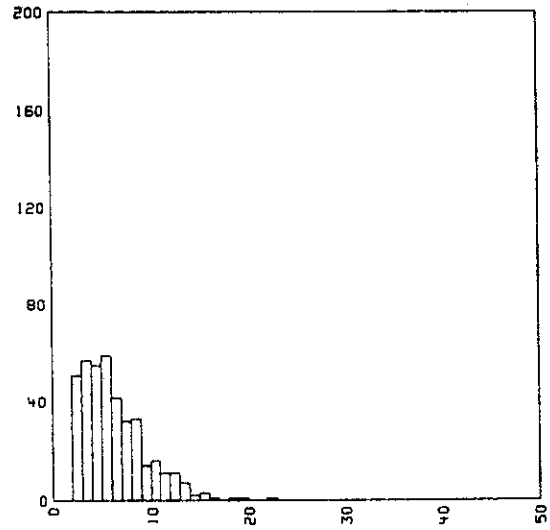


RADIUS

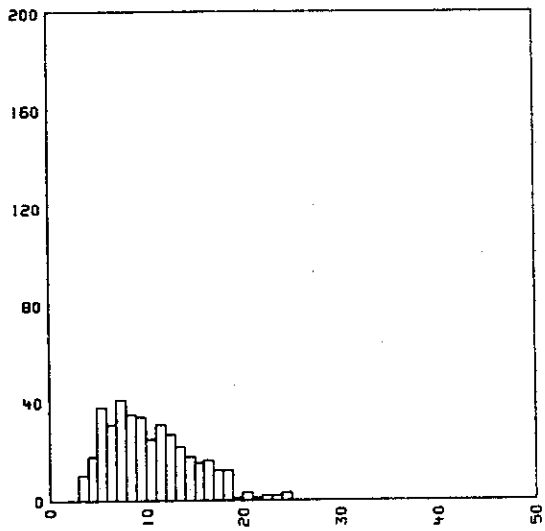
800 POINTS SPHERE $\sigma=2$



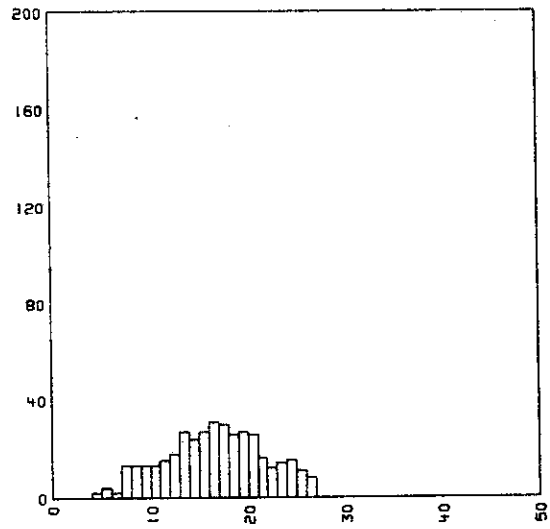
FIRST POINT



SECOND POINT

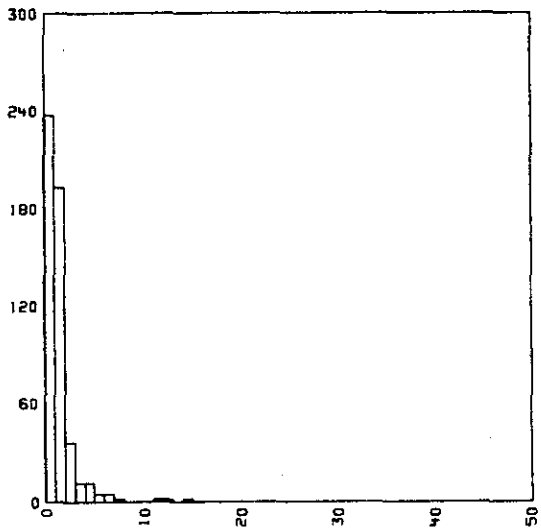


THIRD POINT

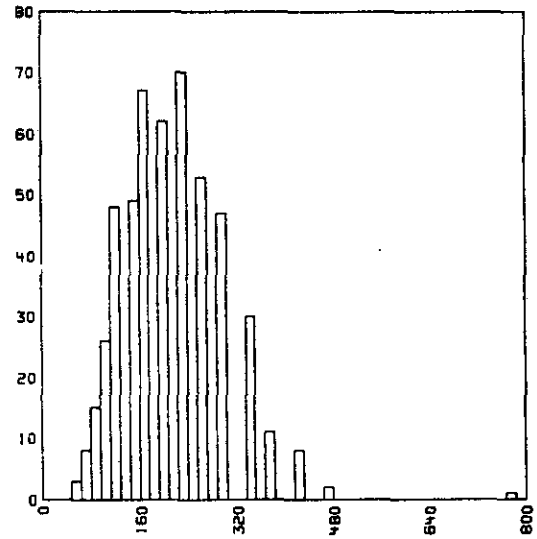


FOURTH POINT

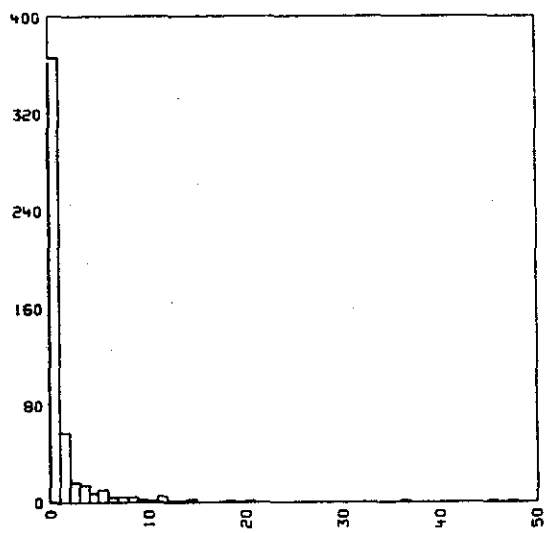
800 POINTS SPHERE $\sigma=2$



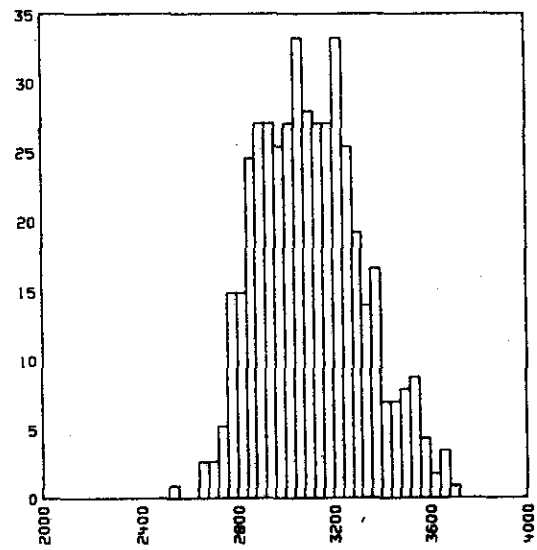
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

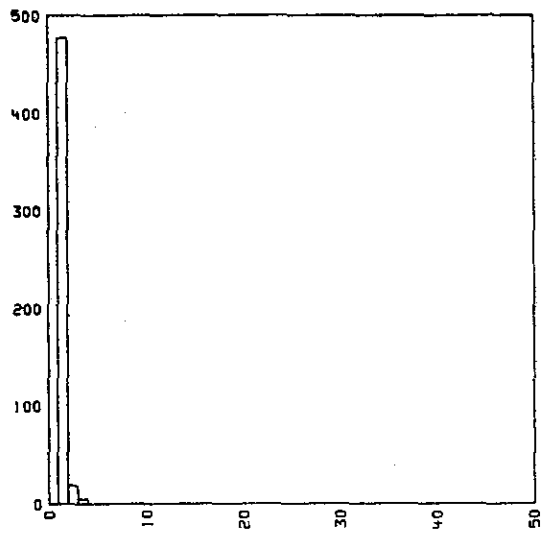


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

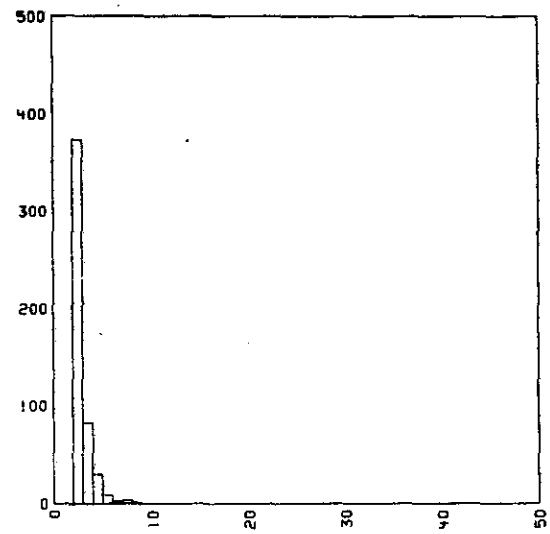


RADIUS

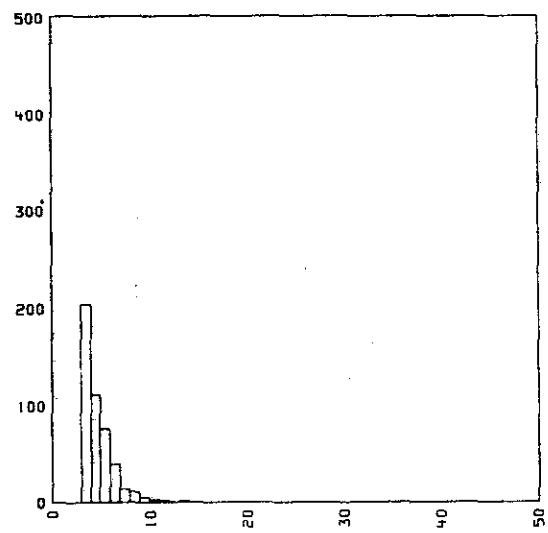
100 POINTS SPHERE $\sigma=4$



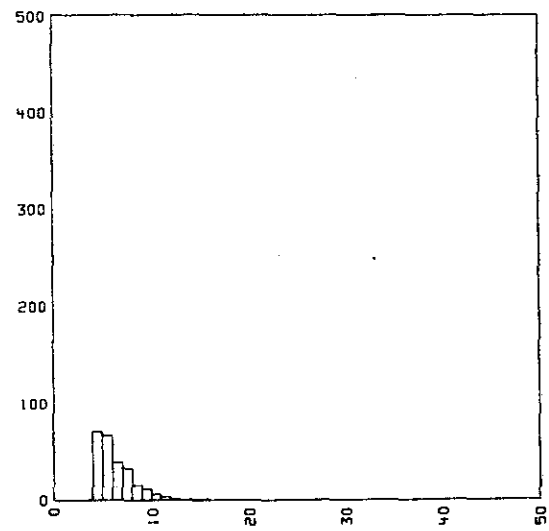
FIRST POINT



SECOND POINT

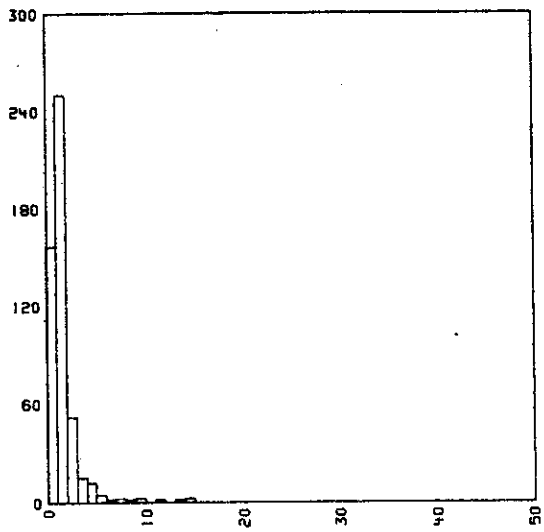


THIRD POINT

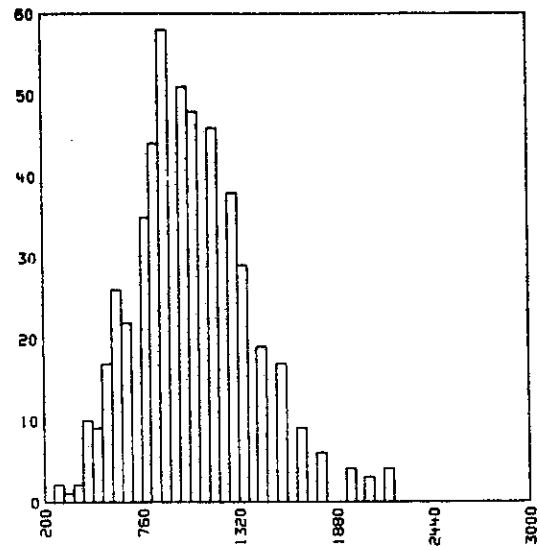


FOURTH POINT

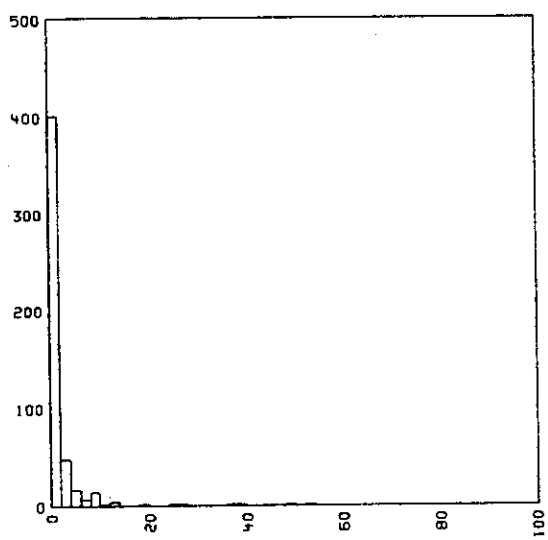
100 POINTS SPHERE $\sigma=4$



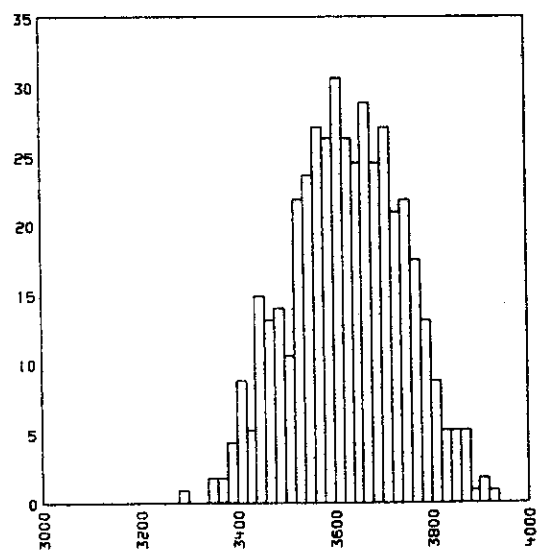
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

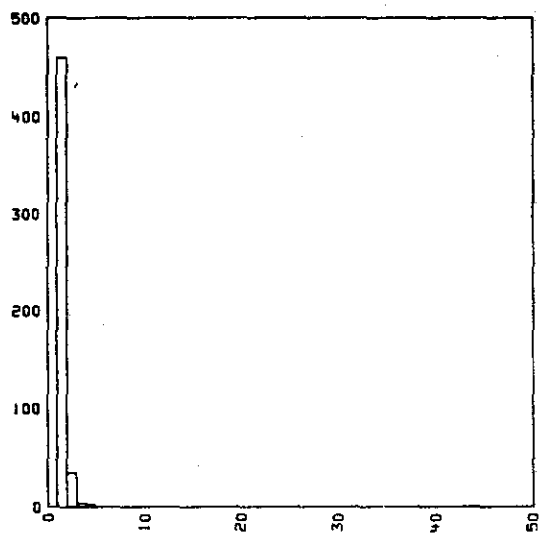


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

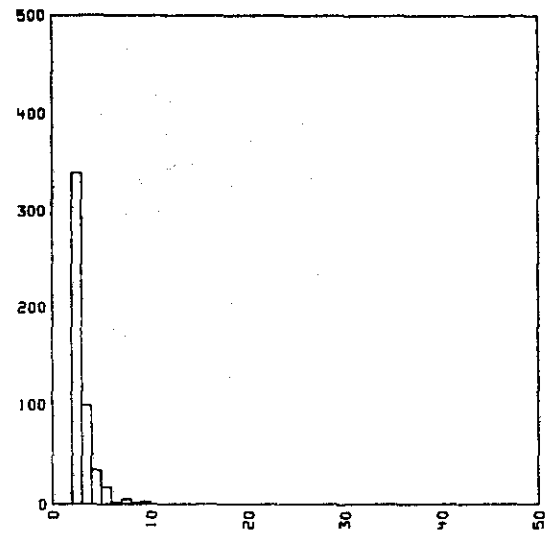


RADIUS

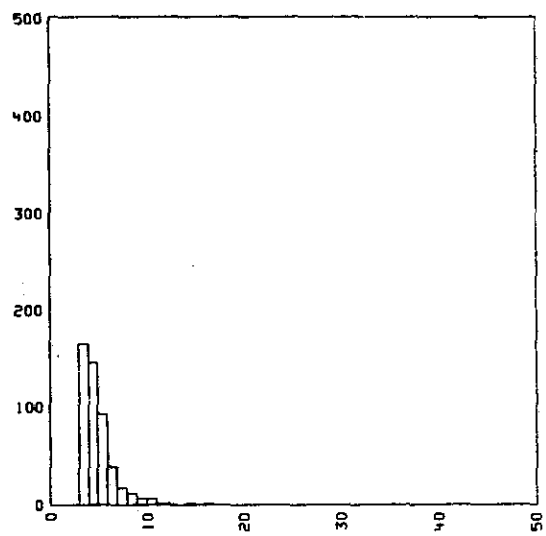
800 POINTS SPHERE $\sigma=4$



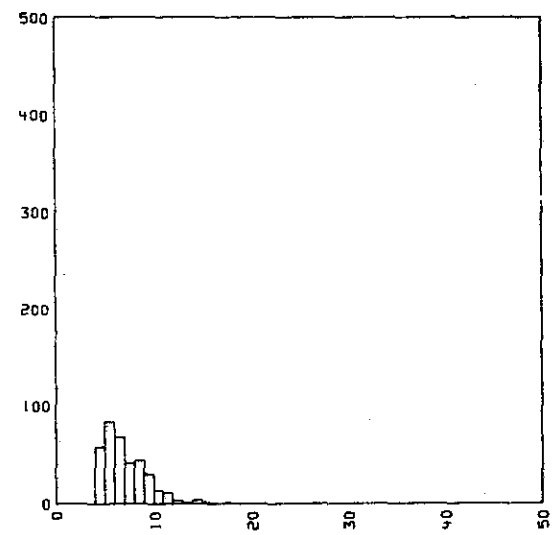
FIRST POINT



SECOND POINT

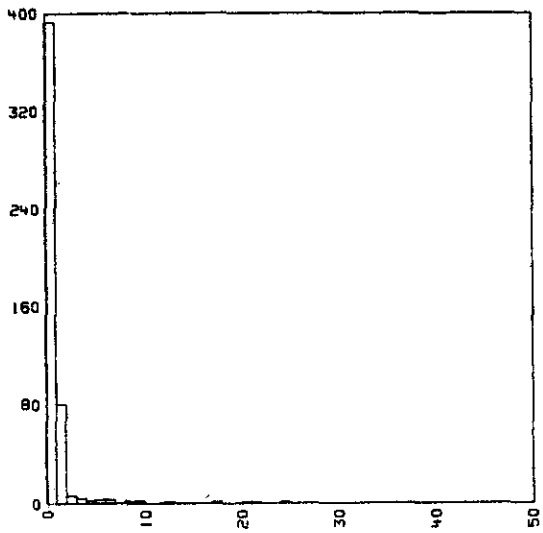


THIRD POINT

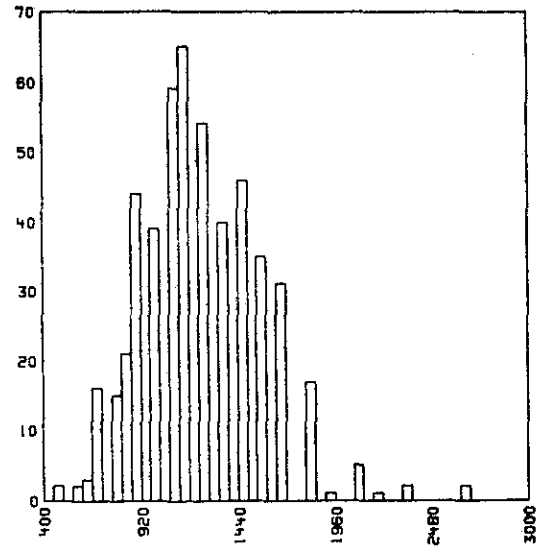


FOURTH POINT

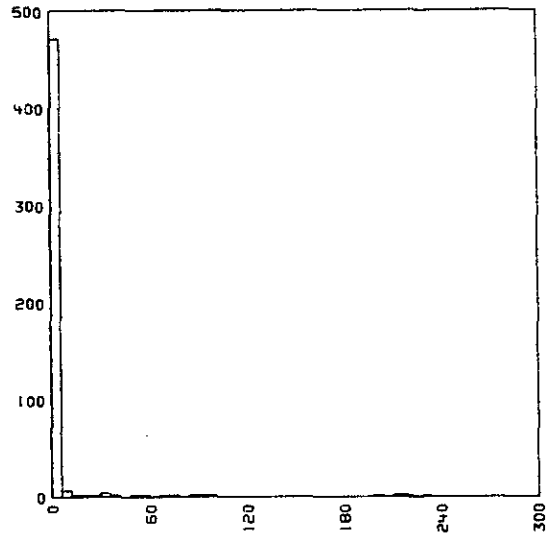
800 POINTS SPHERE $\sigma=4$



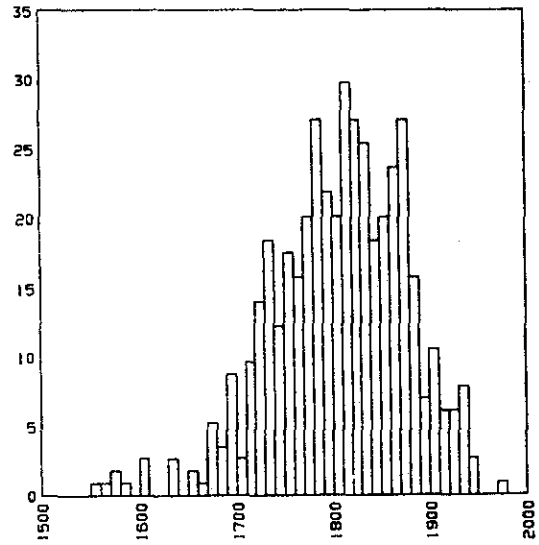
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

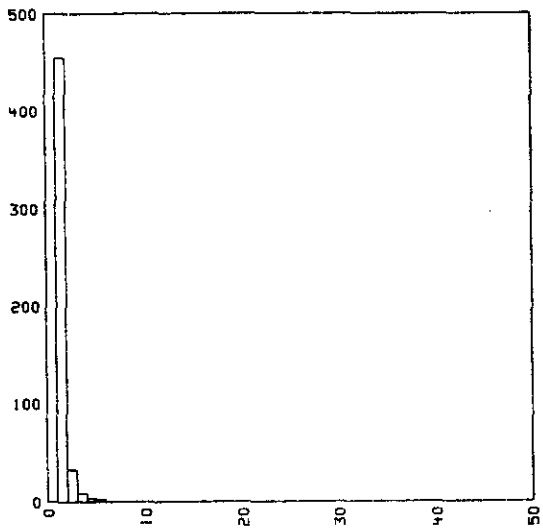


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

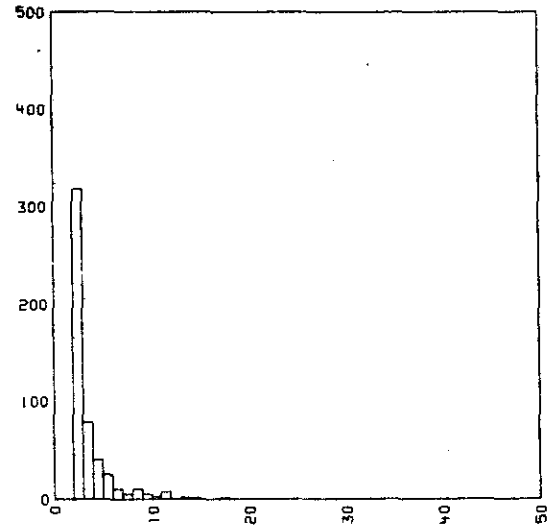


RADIUS

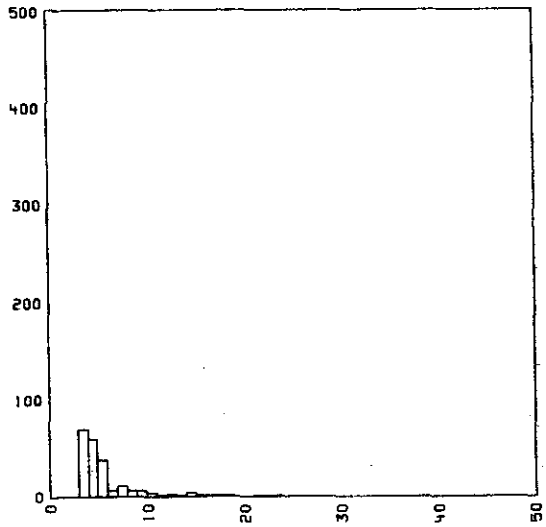
100 POINTS ELIPSE



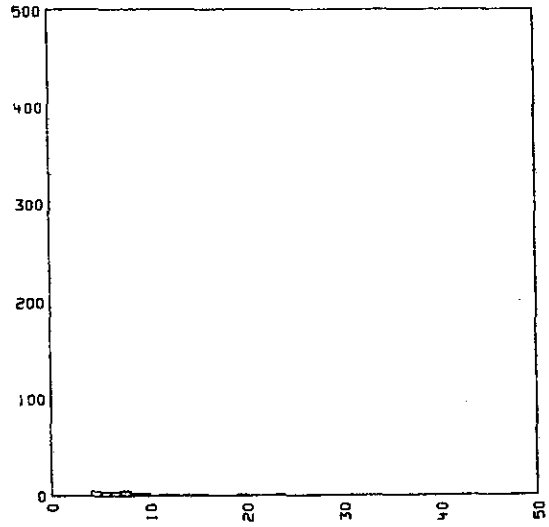
FIRST POINT



SECOND POINT

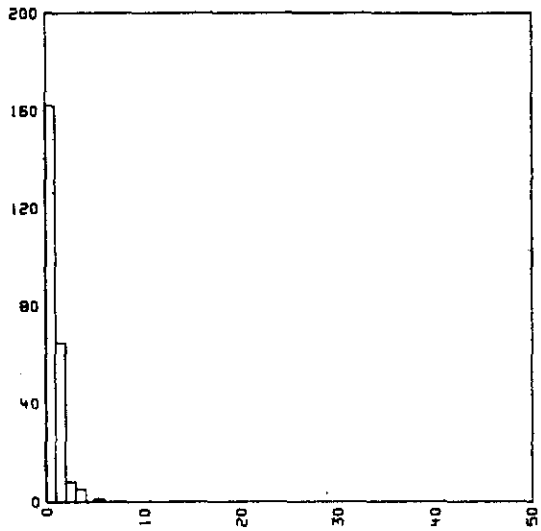


THIRD POINT

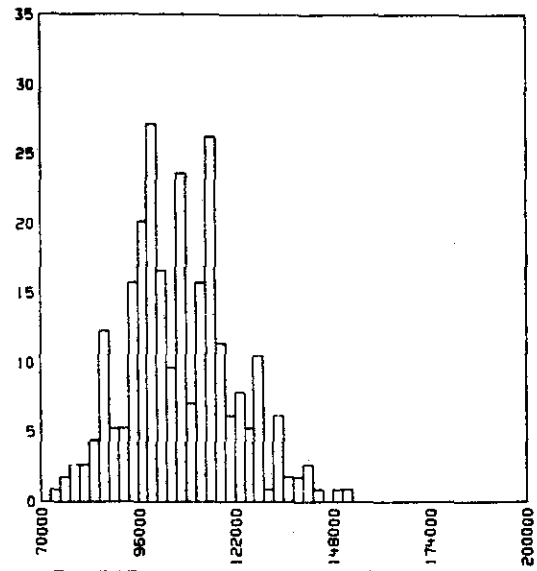


FOURTH POINT

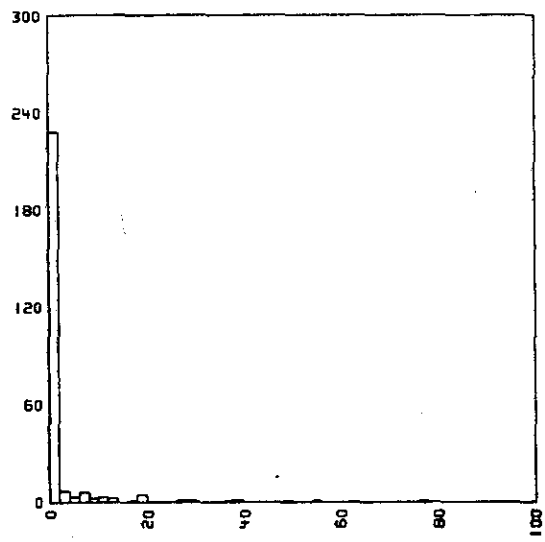
100 POINTS ELIPSE



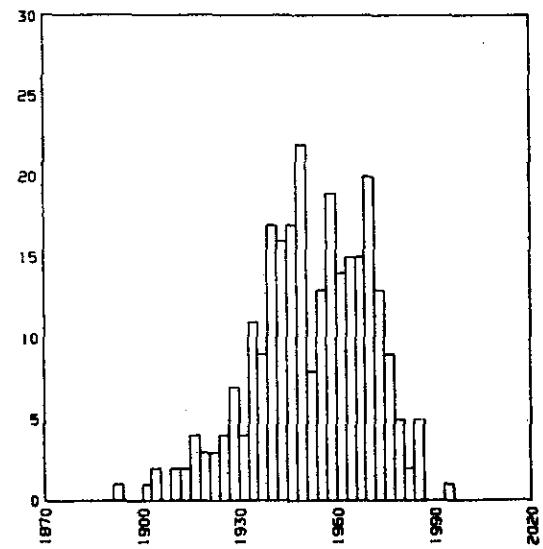
NUMBER OF TESTS(SPHERE)/
NUMBER OF POINTS ON HULL



EXPECTED SPHERES CREATED/
NUMBER OF POINTS ON HULL

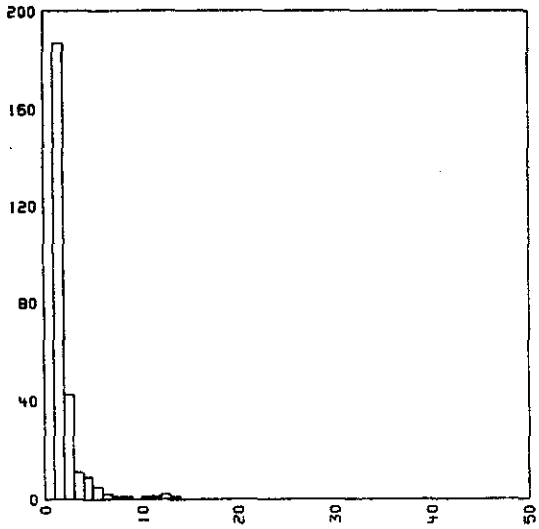


NUMBER OF SPHERES CREATED/
NUMBER OF POINTS ON HULL

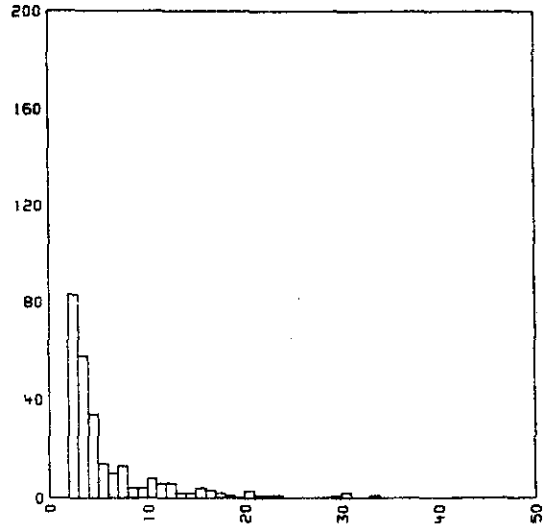


RADIUS

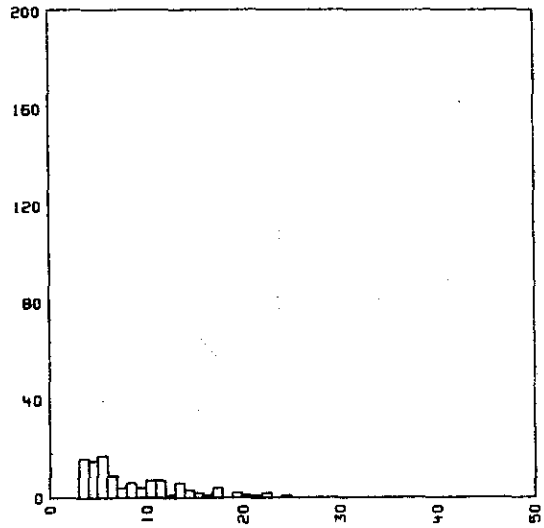
1600 POINTS ELIPSE



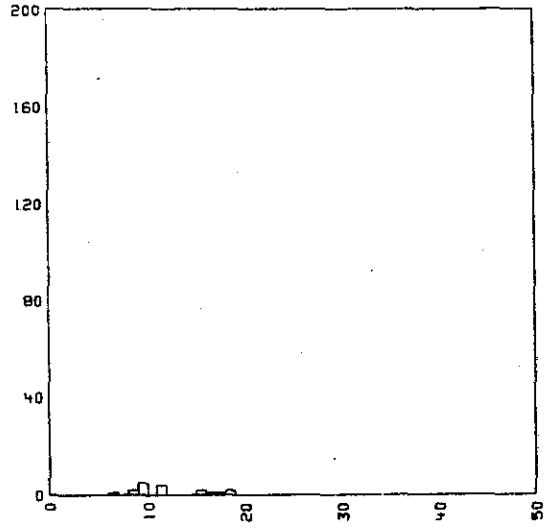
FIRST POINT



SECOND POINT



THIRD POINT



FOURTH POINT

1600 POINTS ELIPSE

APPENDIX III

Hidden Line Data

For the hidden line algorithm, the following quantities were recorded.

A. Front Faces (FF). Total of number of faces that are potentially visible.

B. Objects Considered (invisibility) (OC/I). Total number of objects that are potentially considered to compute invisibility.

C. Objects in window (invisibility) (OW/I). Total number of objects whose window covers the point under test.

D. Extreme Edges Considered (invisibility) (EEC/I). Total number of extreme edges considered when determing if a face covers a point for the invisibility calculation.

E. Intersections Computed (invisibility) (IC/I). Number of intersections computed using EEC/I.

F. Edges to be Displayed (ED). Number of edges that are processed for display.

Note: The remainder of the variables concern the hidden line calculation.

G. Objects Considered (OC/HL). Total number of objects that can potentially hide each of the edges (ED).

H. Objects in Window (OW/HL). Total number of objects whose window is intersected by the edges under test.

I. Extreme Edges Considered (EEC/HL). Total number of edges in the objects (OW/HL).

J. Intersections Computed (IC/HL). Total number of intersection calculations performed.

K. Actual Intersections (AI/HL). Total number of intersection calculations that resulted in an intersection.

L. Time (milliseconds).

The values of the above variables were collected for each of the eight data sets as follows: the data set was first rotated about the Y axis (Yaw) for 360° with data collected at 1° intervals. Then the data was rotated about the X axis for 360° (Pitch) again with data collected every degree. The viewing position was on the Z axis with the entire data set in view.

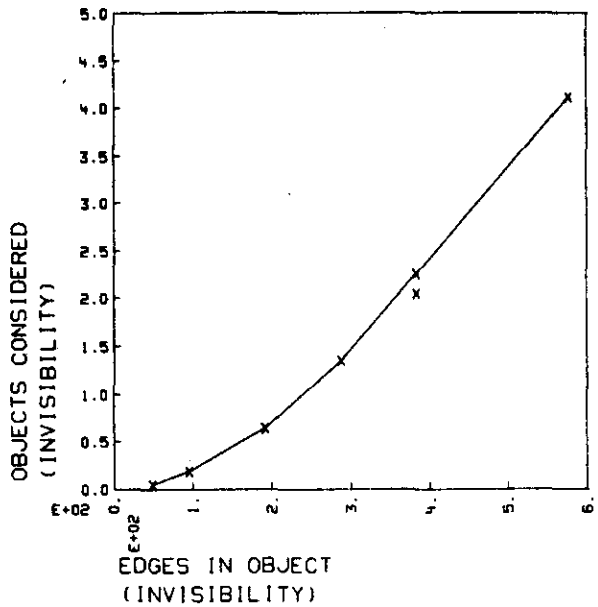
Summary Graphs

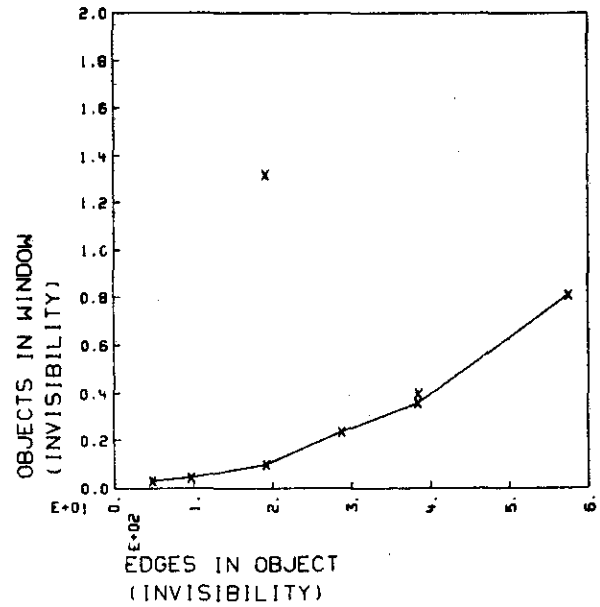
The first 16 graphs (4 pages) contain a plot of the averages of some of the variables for each of the data sets. The six "cube" data sets are plotted as X's and are connected. The "city" type data is referred to as 16 rectangles, while the last data set is the 32 randomly positioned cubes. These two points are not connected and displayed as isolated X's.

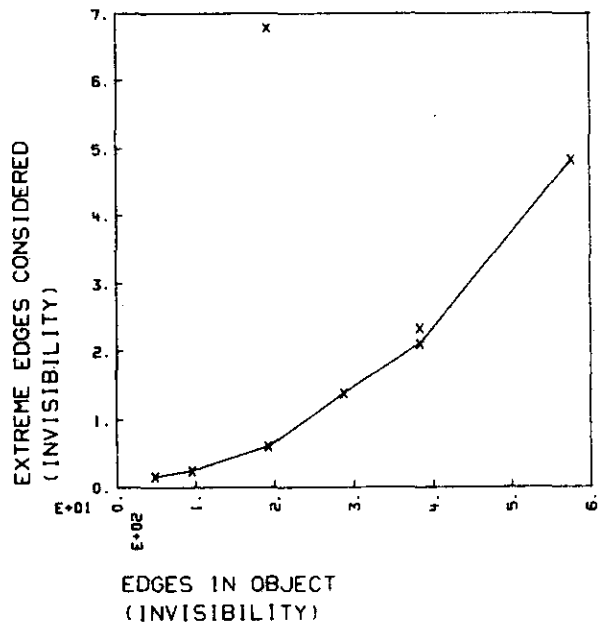
The most interesting result is the graph of time. It shows that for the data sets we examined, the computation time is roughly linear. It is also noteworthy that the data set containing nearly 600 edges was processed in less than 1/2 second.

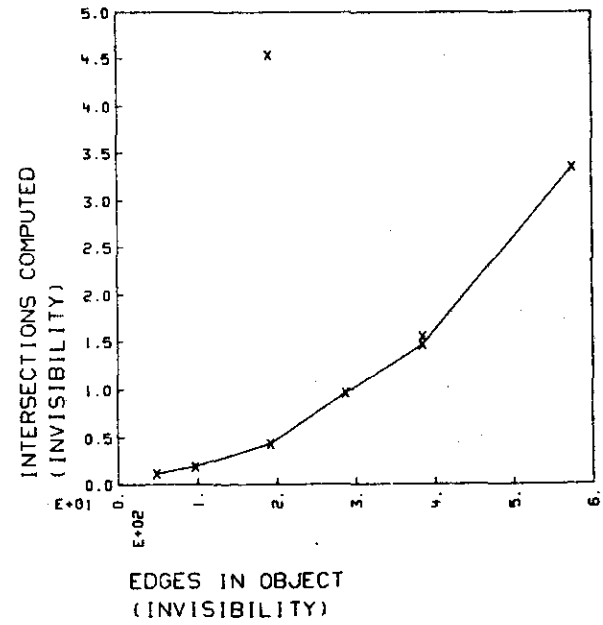
The various ratios are also of interest. (AI/HL)/(IC/HL) is approximately 75-80% for all cases. This means that 75-80% of the intersections computed resulted in an intersection. This is the single most important result as the intersection calculation represents the major effort in all hidden line algorithms.

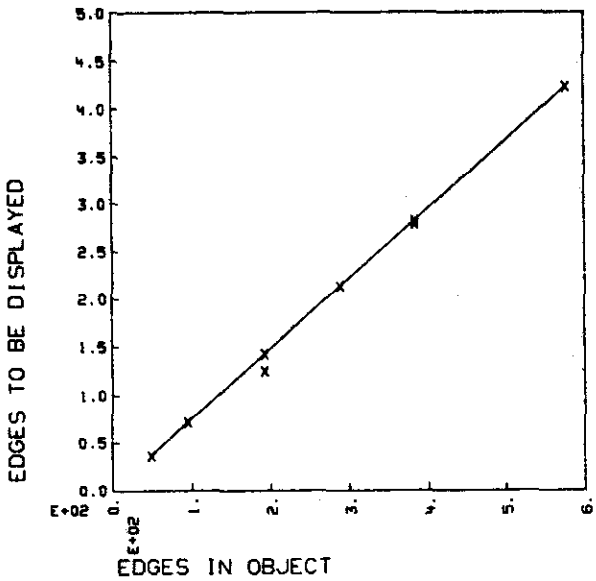
Following the summary graphs are examples of the data collected for several of the data sets. The most noteworthy item is the dependency of the variables on the viewing position. This is not surprising in an algorithm that is highly data sensitive.

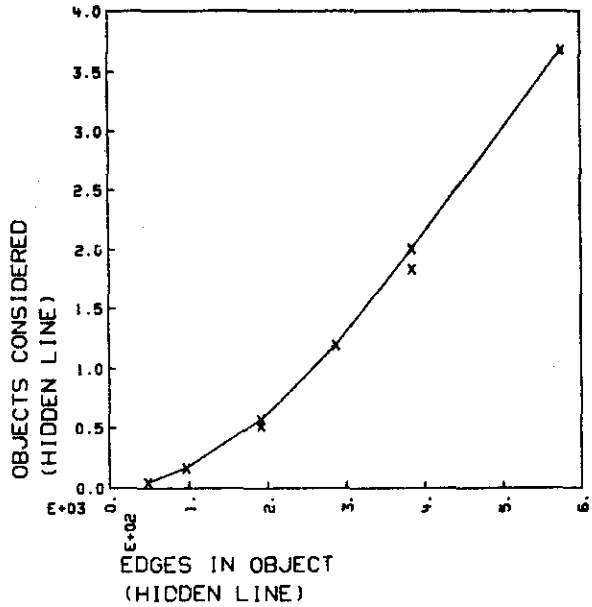

OBJECTS CONSIDERED
(INVISIBILITY)
EDGES IN OBJECT
(INVISIBILITY)
E+02

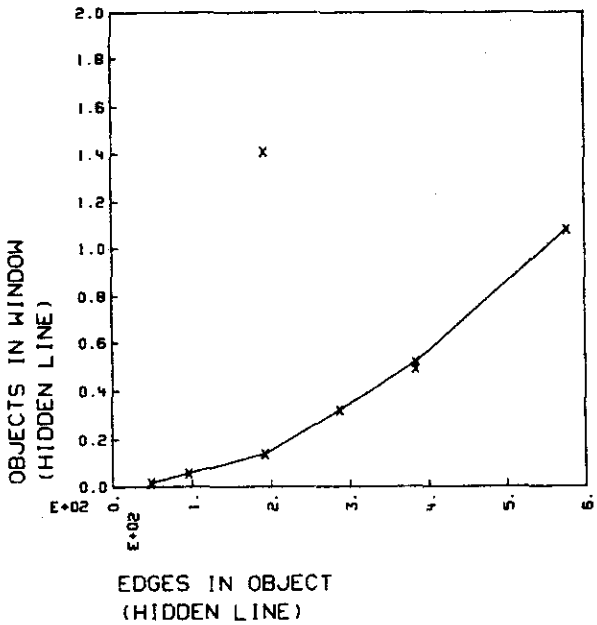

OBJECTS IN WINDOW
(INVISIBILITY)
EDGES IN OBJECT
(INVISIBILITY)
E+01
E+02

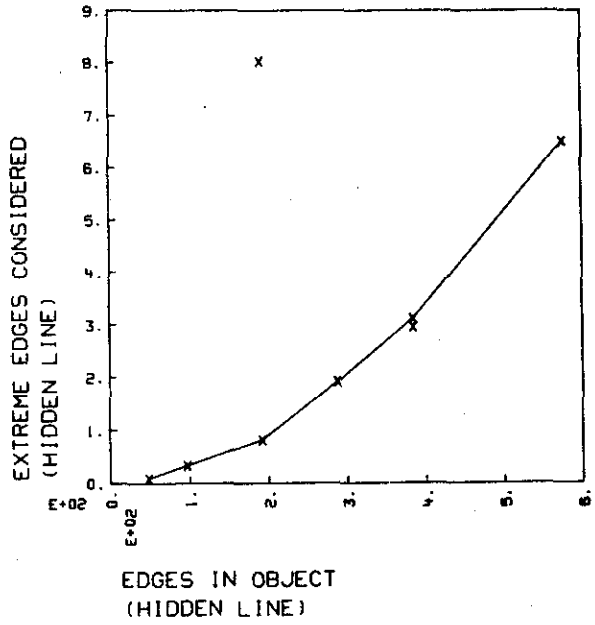

EXTREME EDGES CONSIDERED
(INVISIBILITY)
EDGES IN OBJECT
(INVISIBILITY)
E+01
E+02

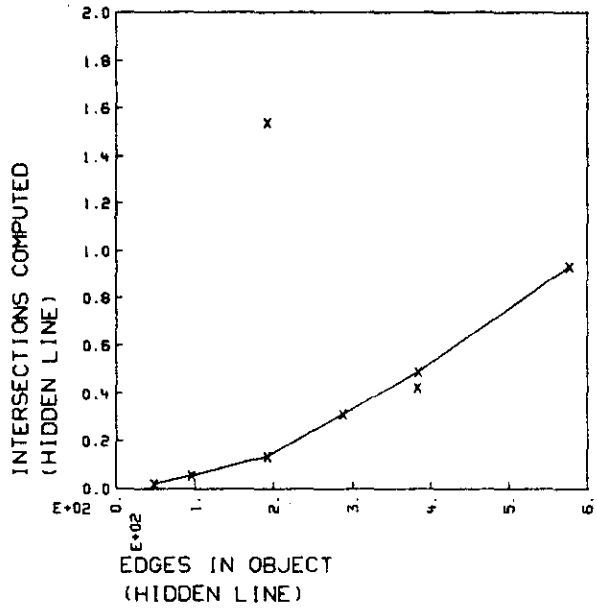

INTERSECTIONS COMPUTED
(INVISIBILITY)
EDGES IN OBJECT
(INVISIBILITY)
E+01
E+02

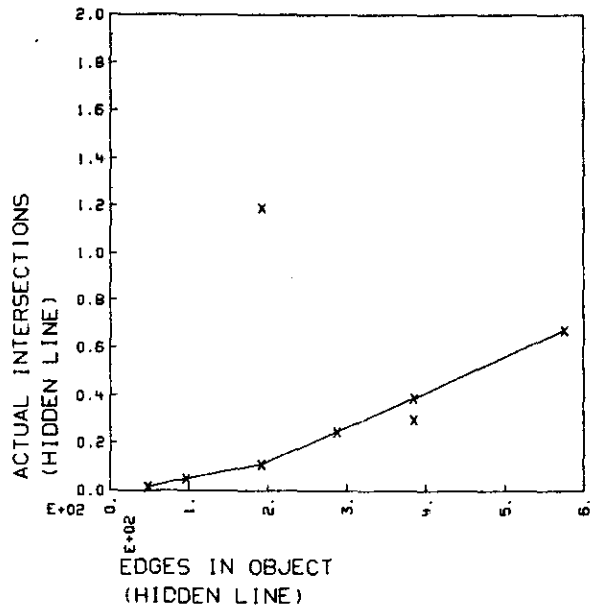

EDGES TO BE DISPLAYED
EDGES IN OBJECT
E+02

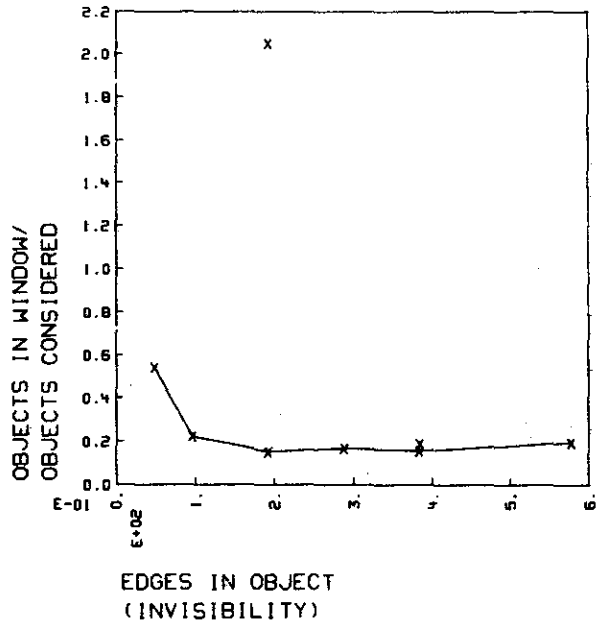

OBJECTS CONSIDERED
(HIDDEN LINE)
EDGES IN OBJECT
(HIDDEN LINE)
E+03
E+02

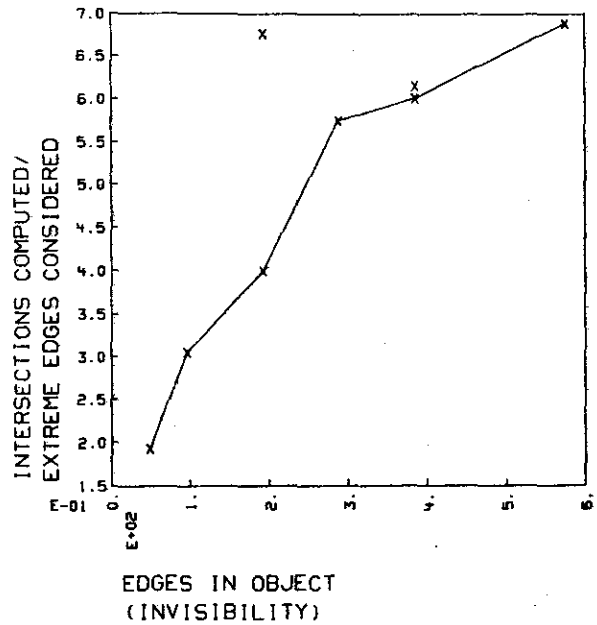

OBJECTS IN WINDOW
(HIDDEN LINE)
EDGES IN OBJECT
(HIDDEN LINE)
E+02

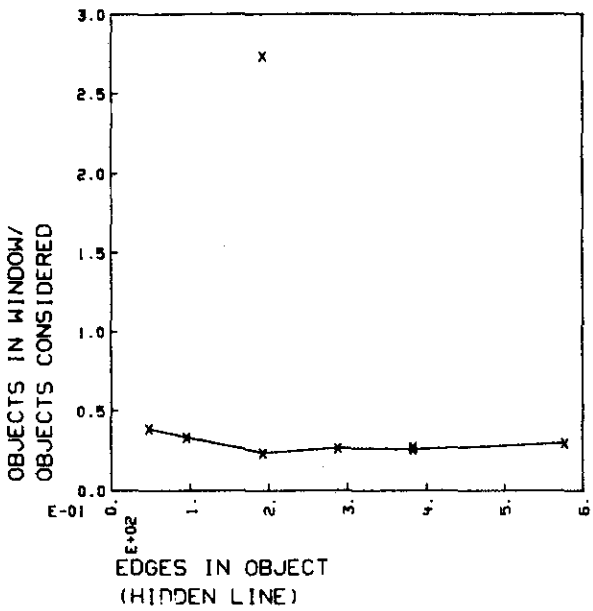

EXTREME EDGES CONSIDERED
(HIDDEN LINE)
EDGES IN OBJECT
(HIDDEN LINE)
E+02

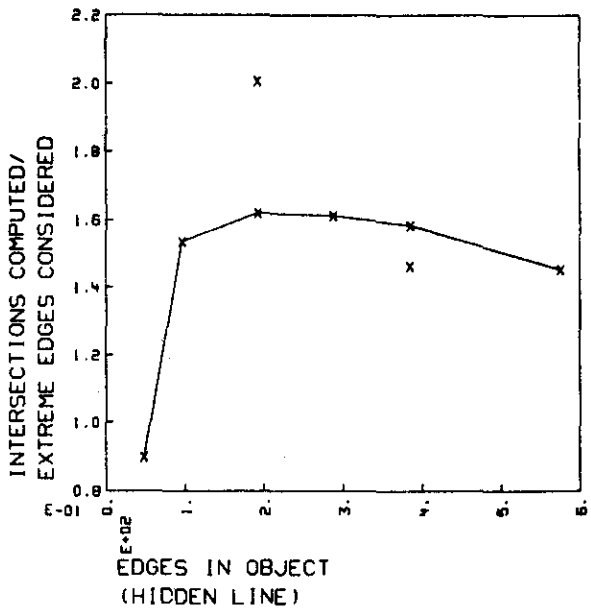

INTERSECTIONS COMPUTED
(HIDDEN LINE)
EDGES IN OBJECT
(HIDDEN LINE)

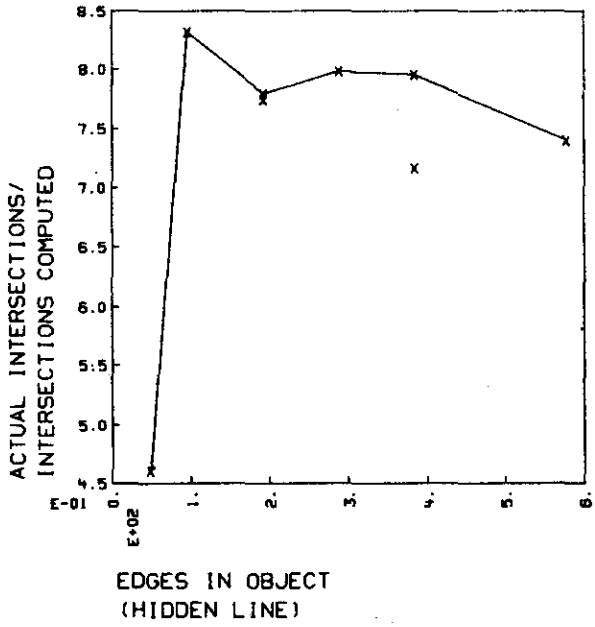

ACTUAL INTERSECTIONS
(HIDDEN LINE)
EDGES IN OBJECT
(HIDDEN LINE)

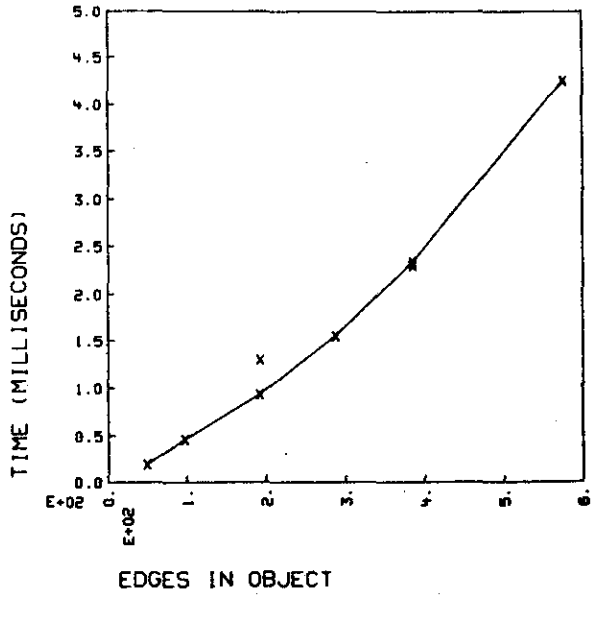

OBJECTS IN WINDOW/
OBJECTS CONSIDERED
EDGES IN OBJECT
(INVISIBILITY)

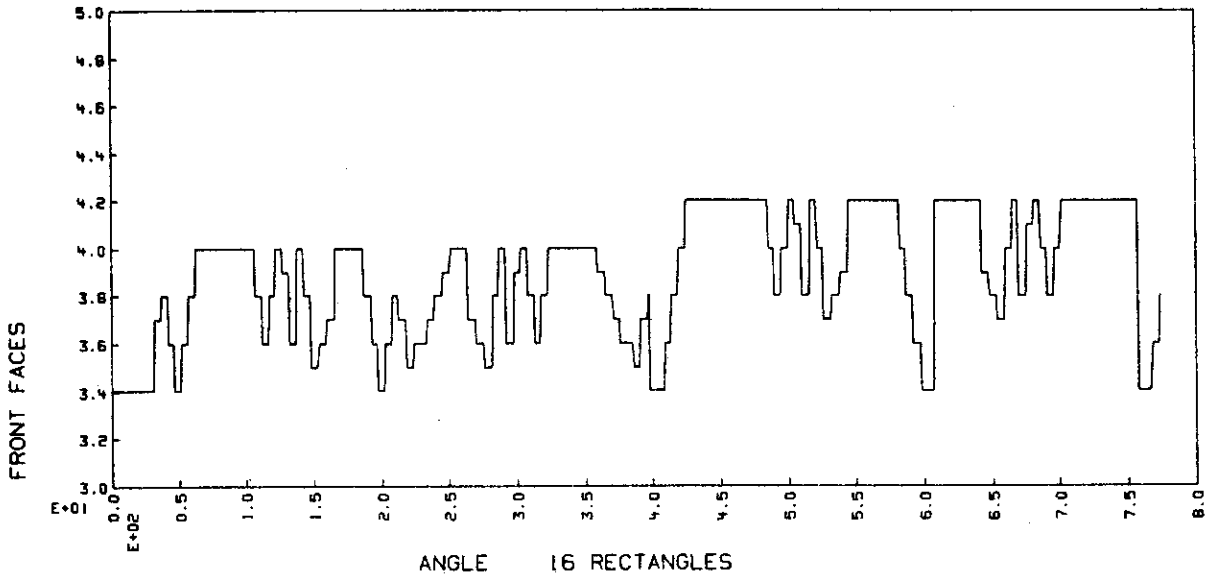

INTERSECTIONS COMPUTED/
EXTREME EDGES CONSIDERED
EDGES IN OBJECT
(INVISIBILITY)

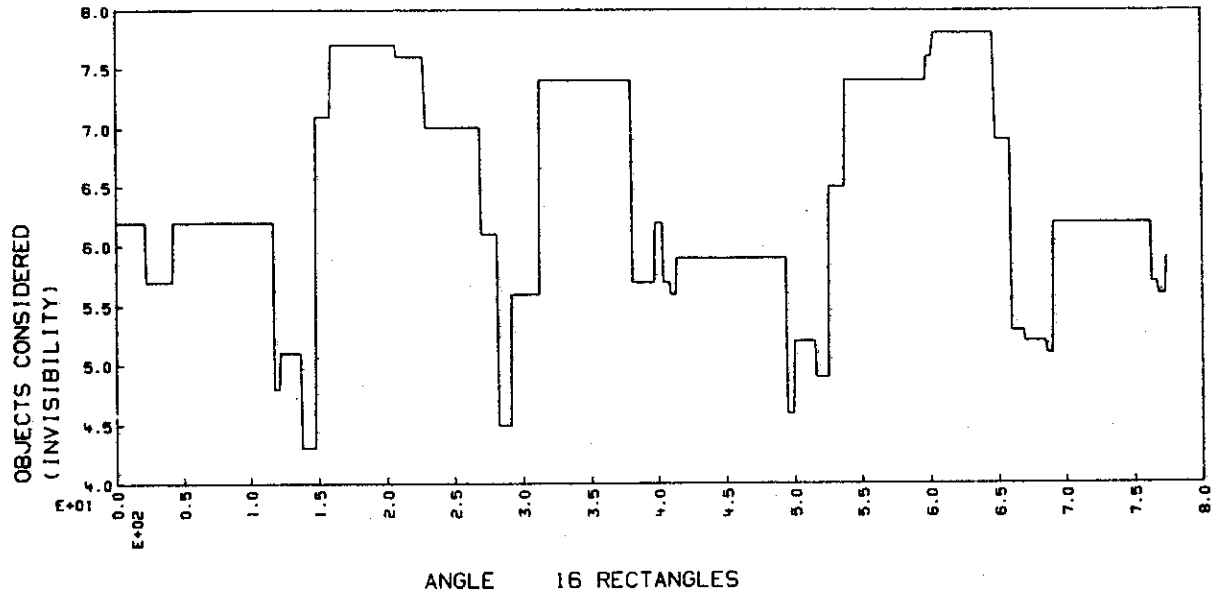

OBJECTS IN WINDOW/
OBJECTS CONSIDERED
3.0
2.5
2.0
1.5
1.0
0.5
0.0
E-01
0.
1.
2.
3.
4.
5.
6.
E+02
EDGES IN OBJECT
(HIDDEN LINE)

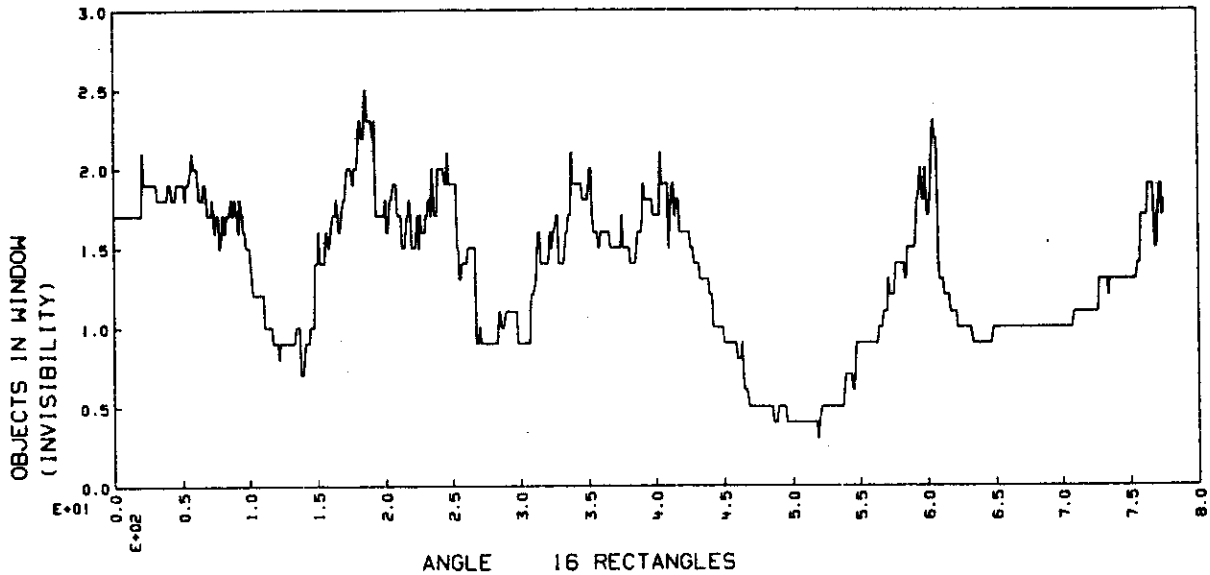

INTERSECTIONS COMPUTED/
EXTREME EDGES CONSIDERED
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
E-01
0.
1.
2.
3.
4.
5.
6.
E+02
EDGES IN OBJECT
(HIDDEN LINE)

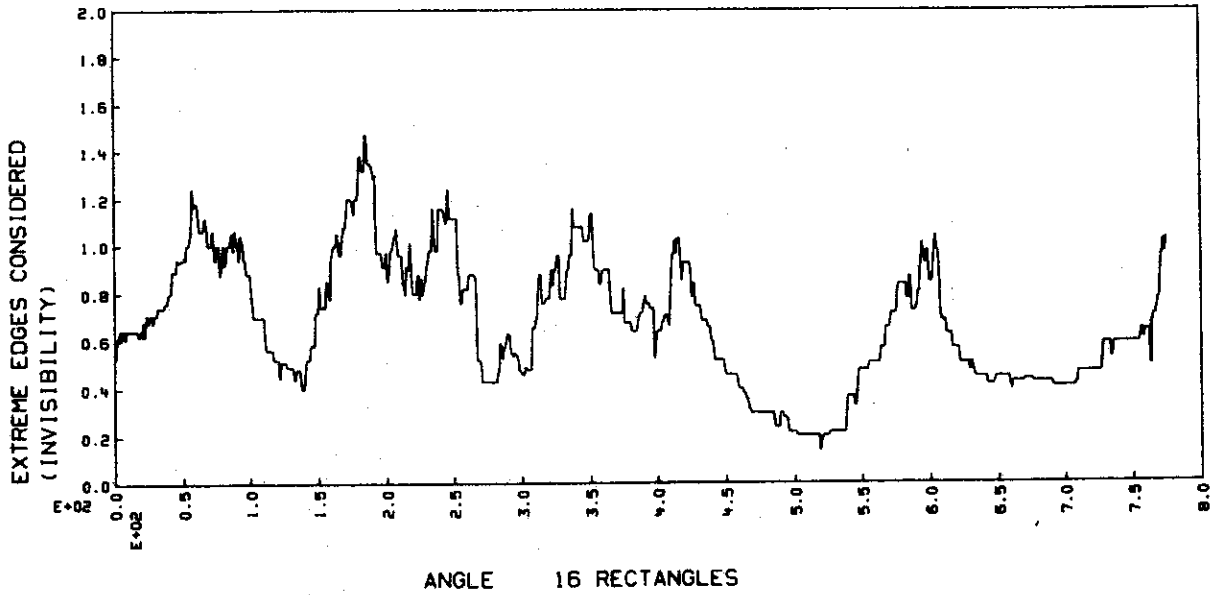

ACTUAL INTERSECTIONS/
INTERSECTIONS COMPUTED
8.5
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
E-01
0.
1.
2.
3.
4.
5.
6.
E+02
EDGES IN OBJECT
(HIDDEN LINE)

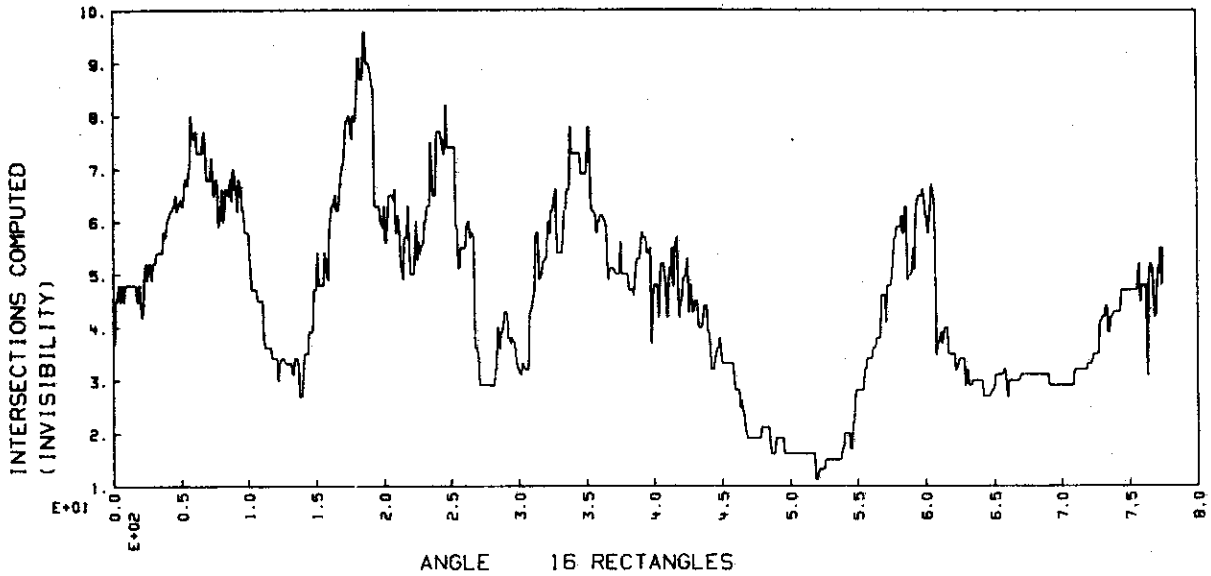

TIME (MILLISECONDS)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
E+02
0.
1.
2.
3.
4.
5.
6.
E+02
EDGES IN OBJECT

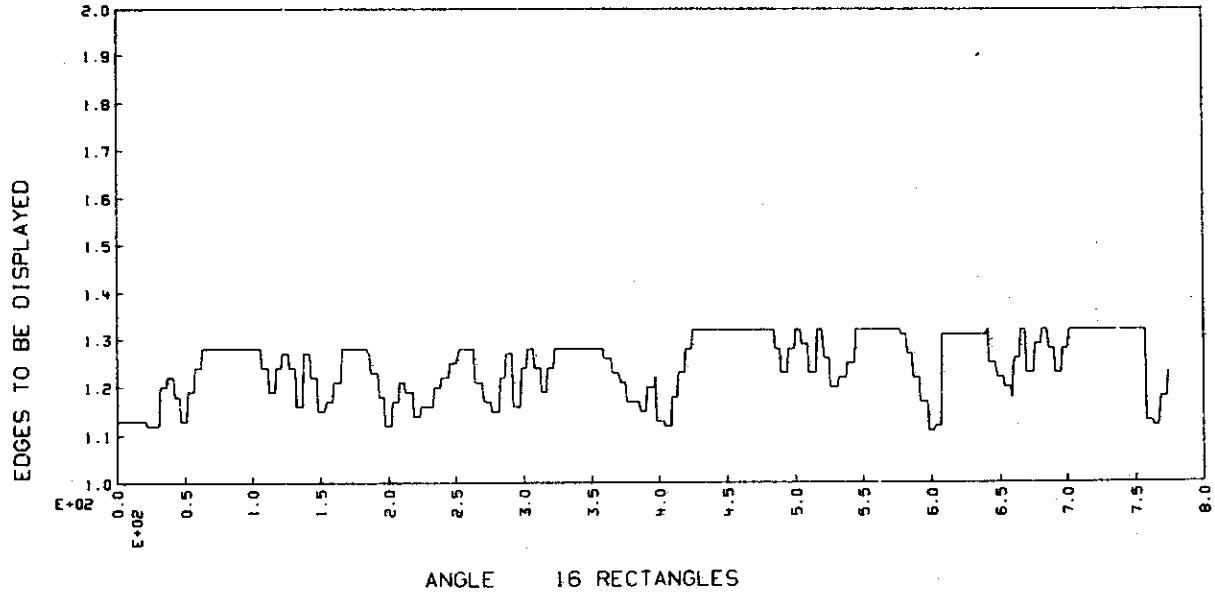

FRONT FACES
5.0
4.8
4.6
4.4
4.2
4.0
3.8
3.6
3.4
3.2
3.0
E+01
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
ANGLE 16 RECTANGLES

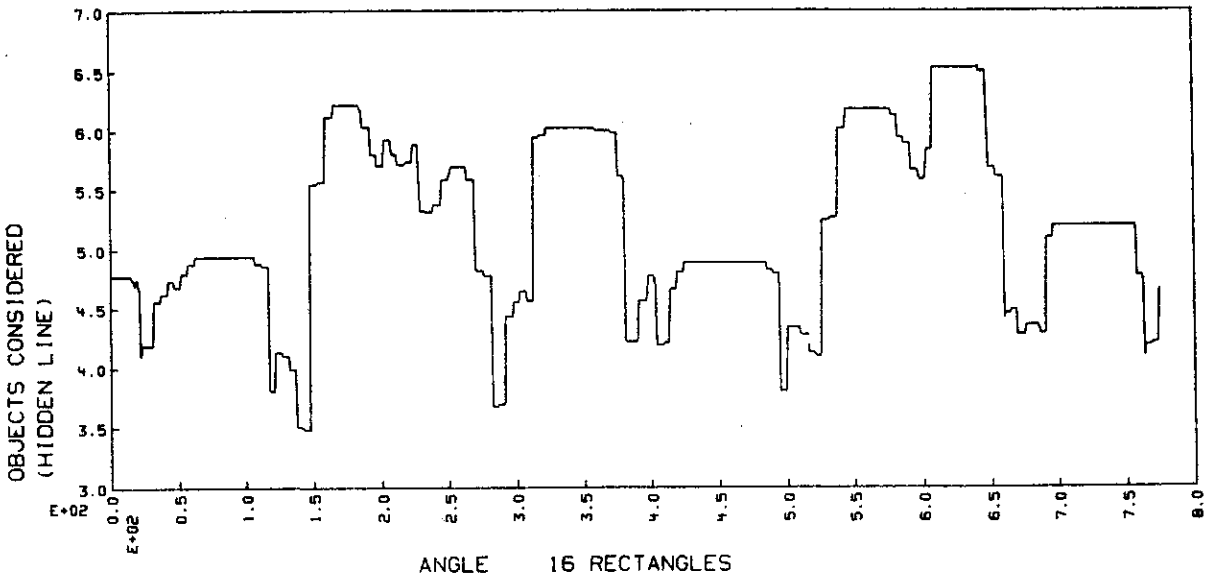

OBJECTS CONSIDERED
(INVISIBILITY)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
E+01
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
ANGLE 16 RECTANGLES

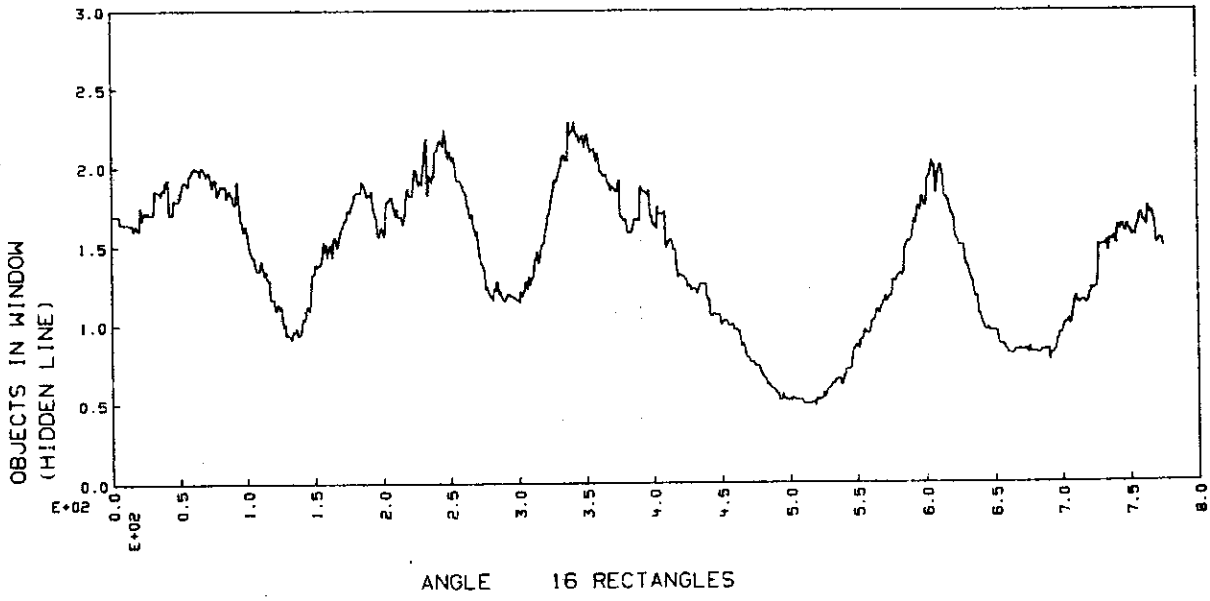

OBJECTS IN WINDOW
(INVISIBILITY)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
E+01
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
ANGLE 16 RECTANGLES

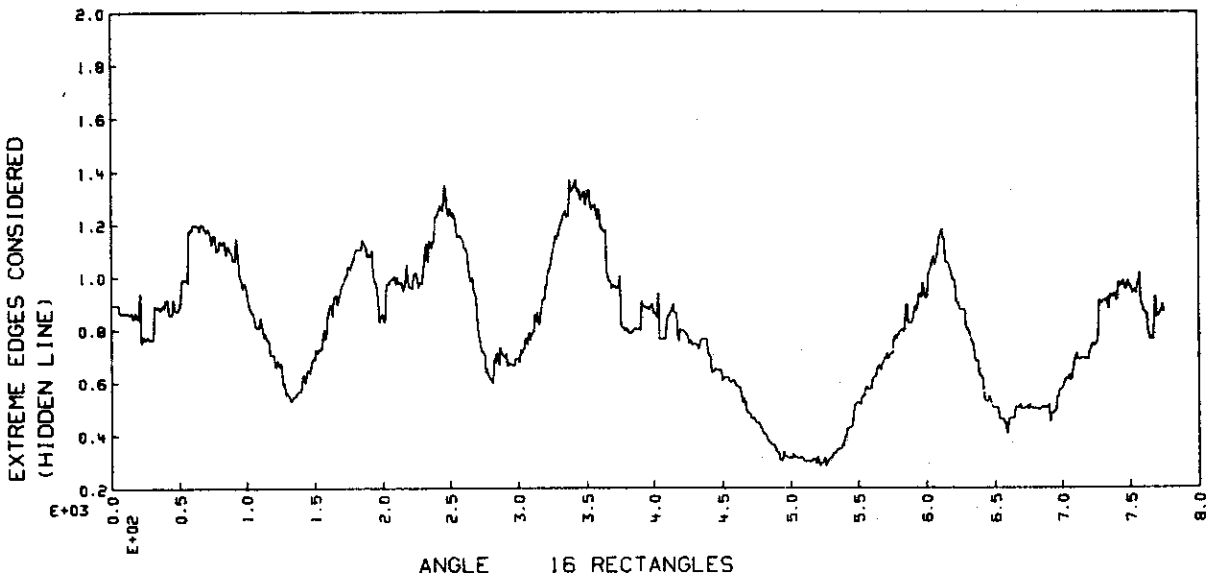

EXTREME EDGES CONSIDERED
(INVISIBILITY)
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
E+02
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
ANGLE 16 RECTANGLES

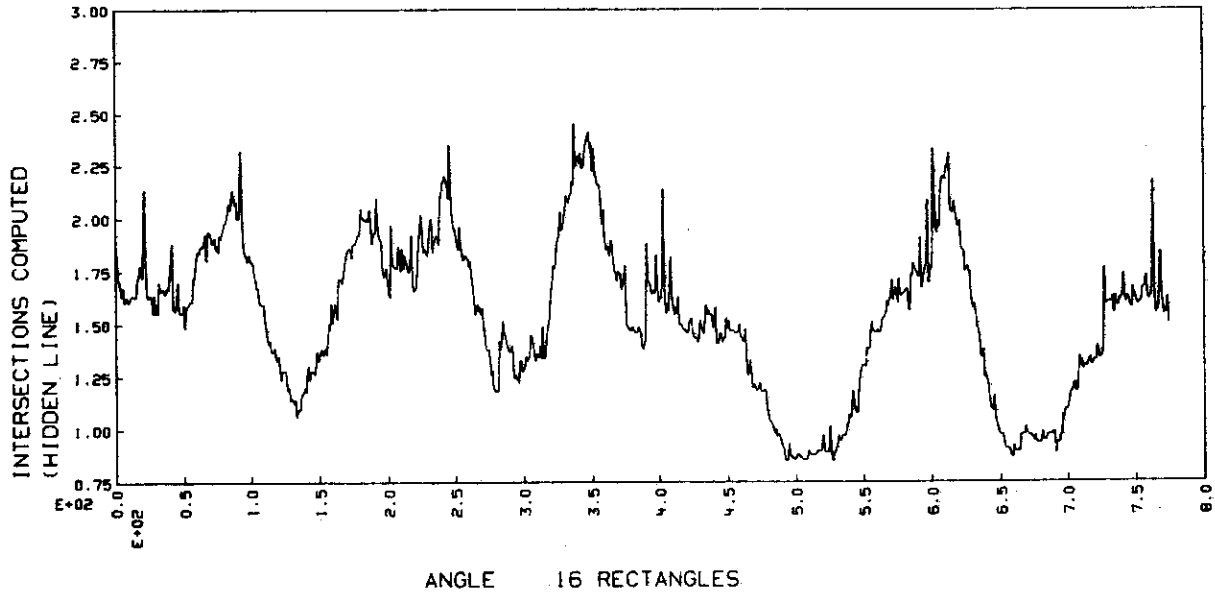

INTERSECTIONS COMPUTED
(INVISIBILITY)
10.
9.
8.
7.
6.
5.
4.
3.
2.
1.
E+01
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
ANGLE 16 RECTANGLES

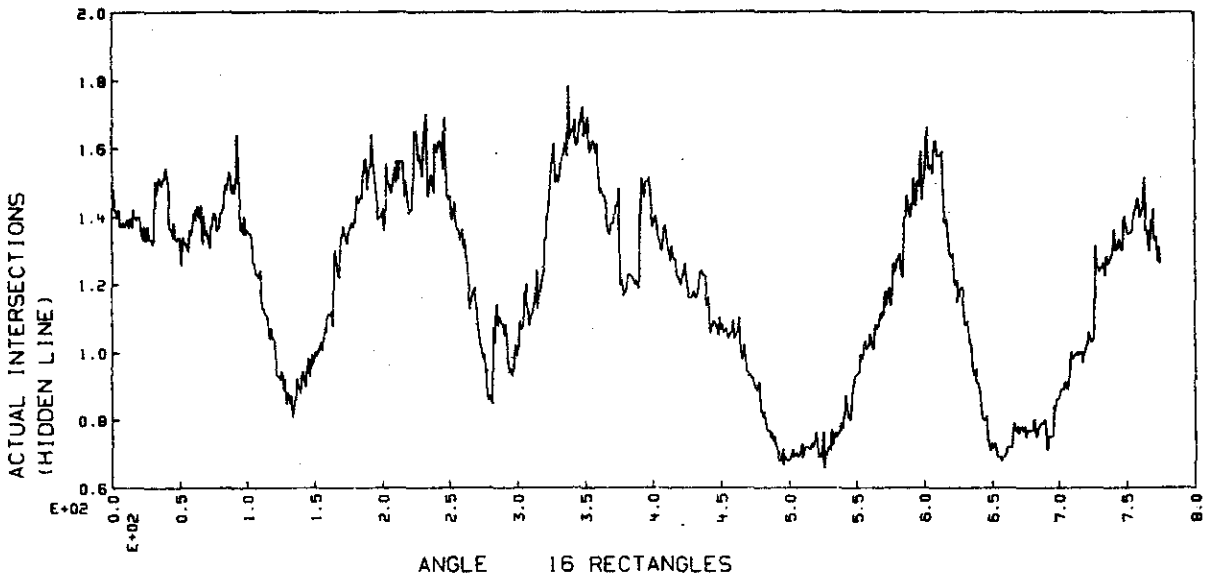

EDGES TO BE DISPLAYED
2.0
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
E+02
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
ANGLE 16 RECTANGLES

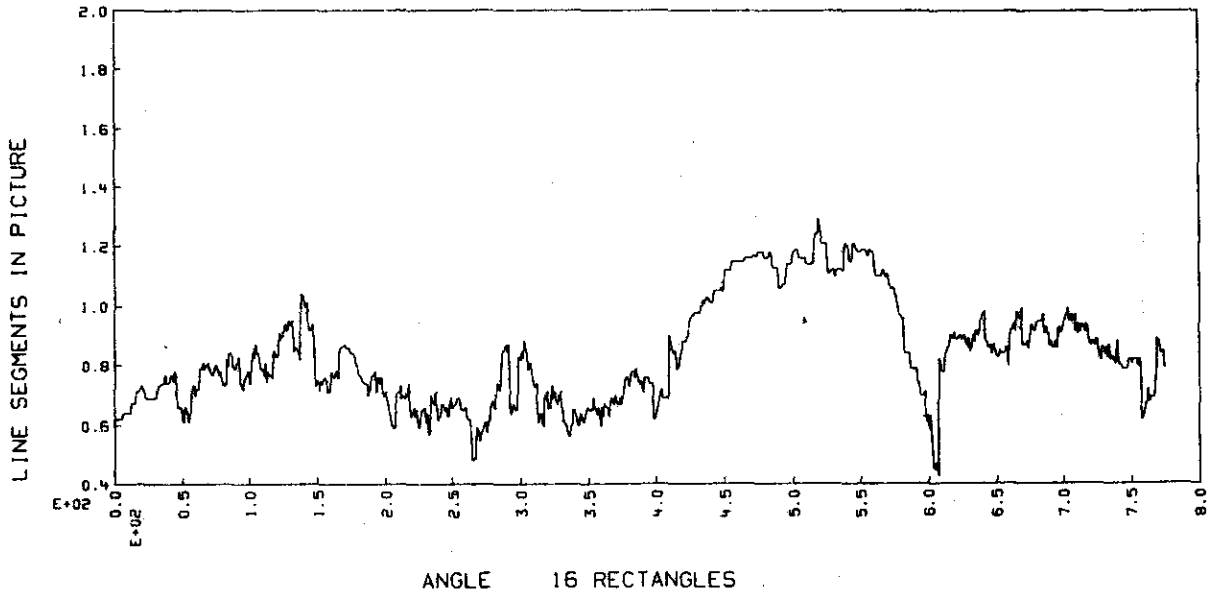

OBJECTS CONSIDERED
(HIDDEN LINE)
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
E+02
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
ANGLE 16 RECTANGLES

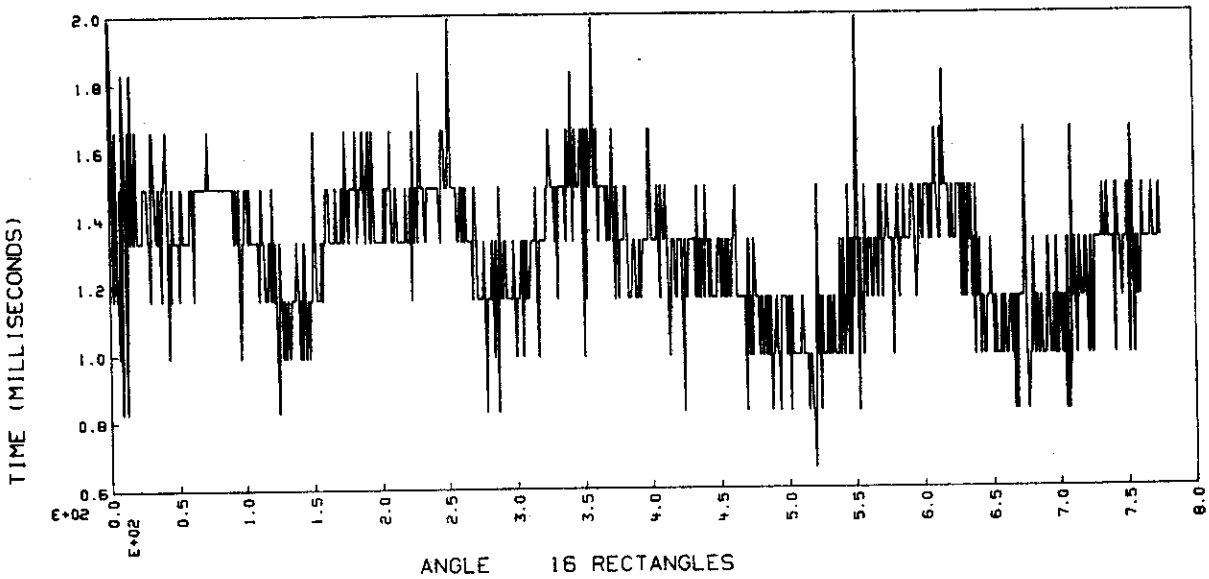

OBJECTS IN WINDOW
(HIDDEN LINE)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
E+02
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
ANGLE 16 RECTANGLES

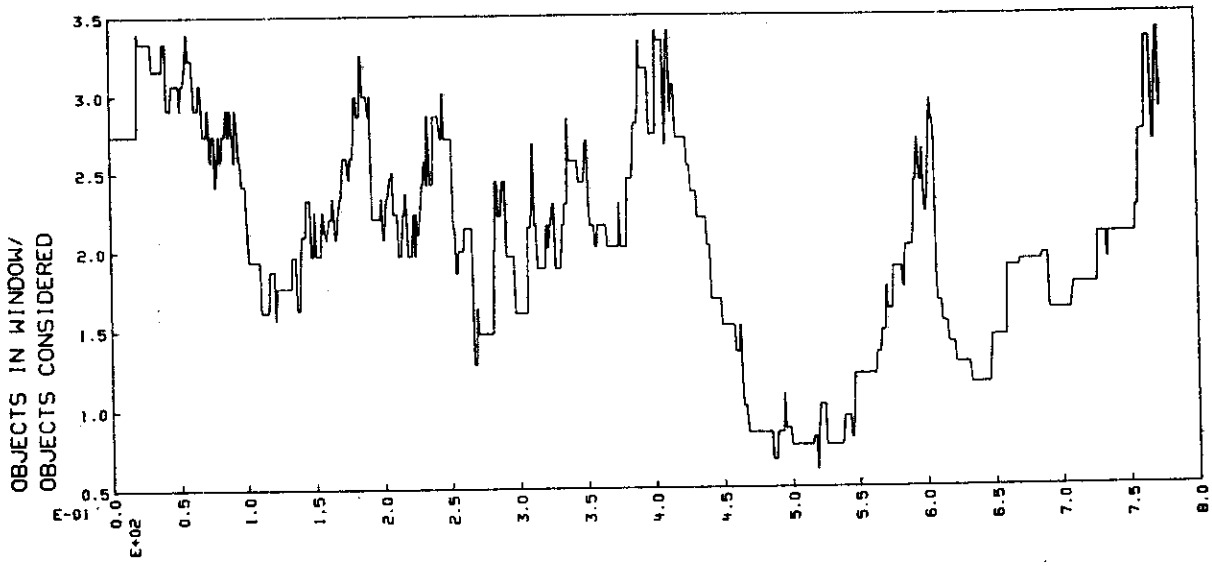

EXTREME EDGES CONSIDERED
(HIDDEN LINE)
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
E+03
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
ANGLE 16 RECTANGLES

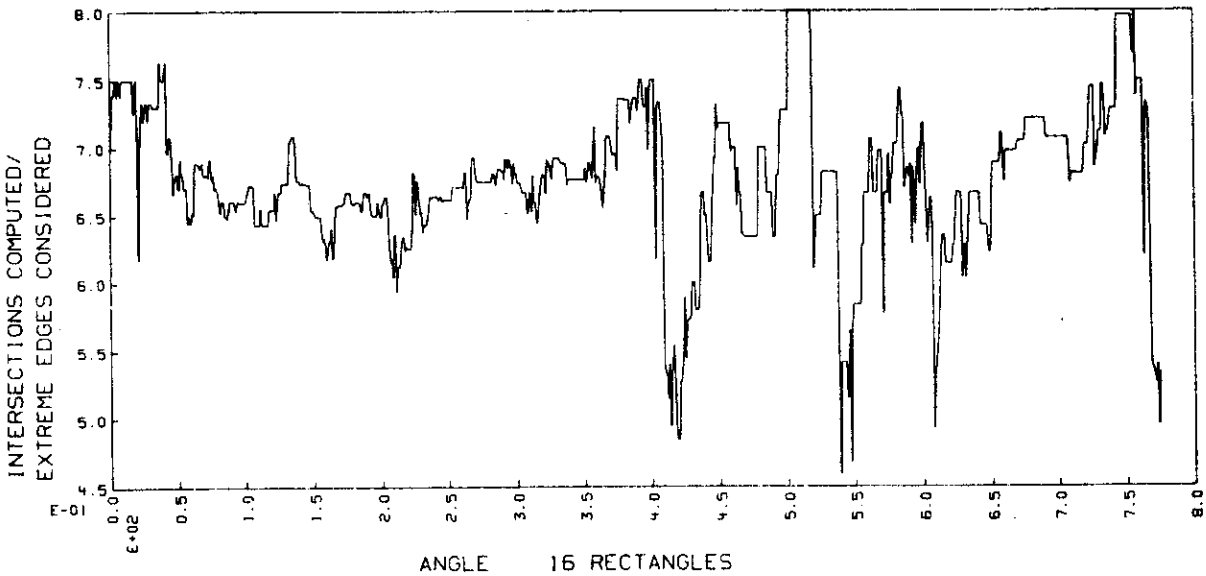

INTERSECTIONS COMPUTED
(HIDDEN LINE)
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
E+02
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
ANGLE 16 RECTANGLES


ACTUAL INTERSECTIONS
(HIDDEN LINE)
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
E+02
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
ANGLE 16 RECTANGLES

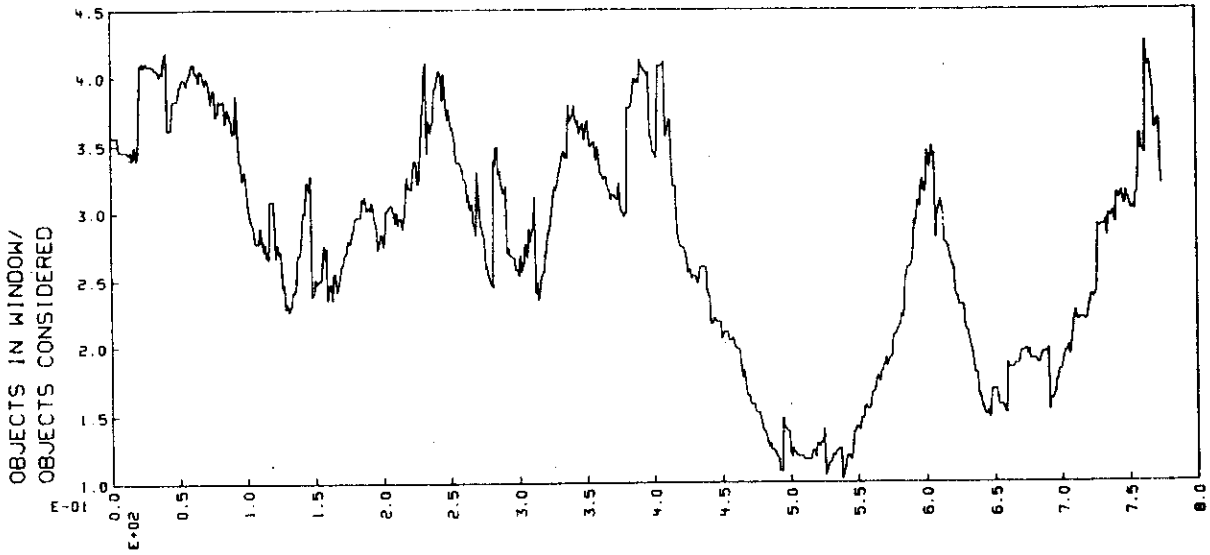

LINE SEGMENTS IN PICTURE
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
E+02
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
ANGLE 16 RECTANGLES


TIME (MILLISECONDS)
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
E+02
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
ANGLE 16 RECTANGLES

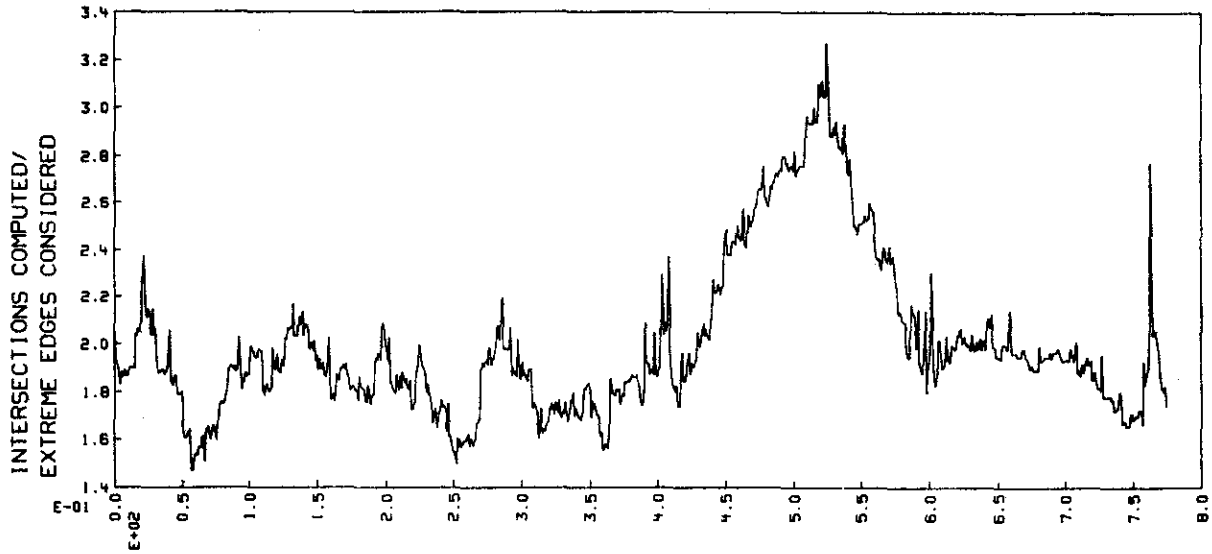

OBJECTS IN WINDOW/
OBJECTS CONSIDERED
3.5
3.0
2.5
2.0
1.5
1.0
0.5
E-01
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
ANGLE 16 RECTANGLES
(INVISIBILITY)



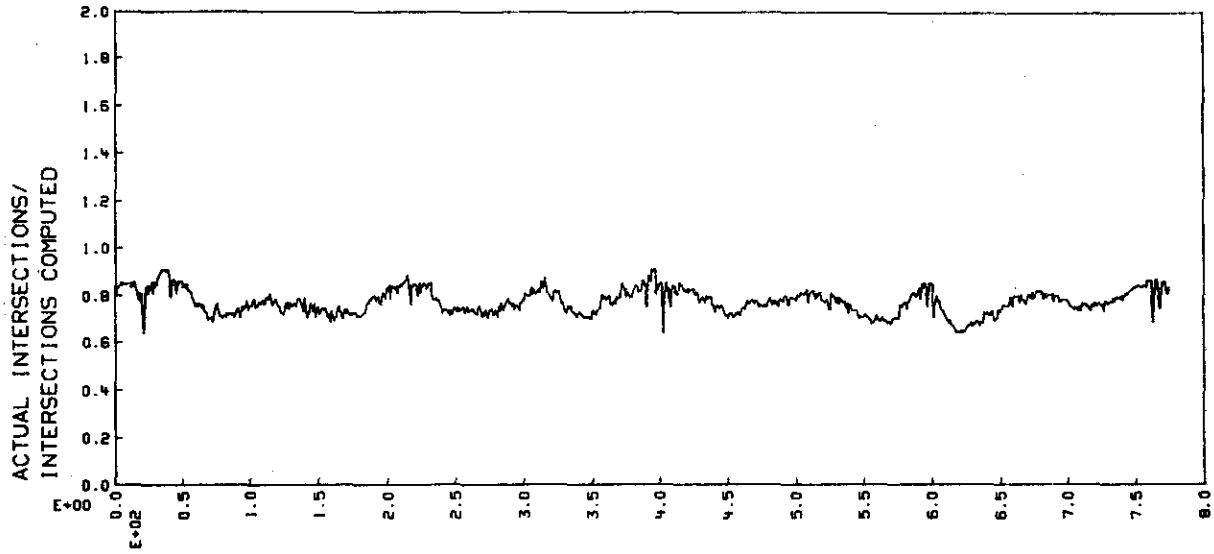
ANGLE 16 RECTANGLES
(INVISIBILITY)



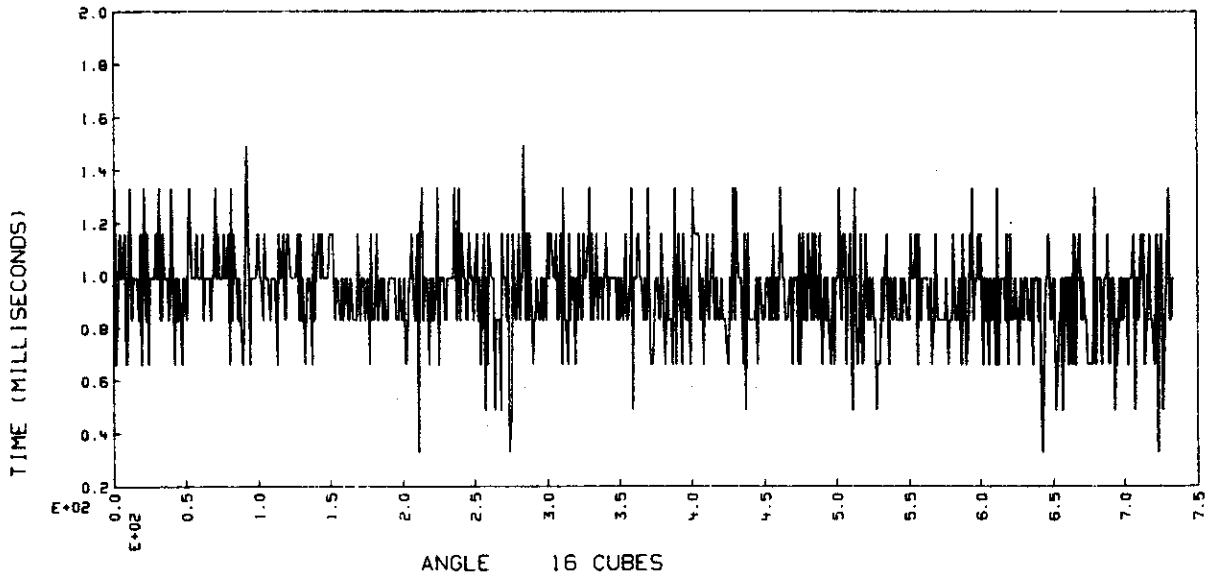
ANGLE 16 RECTANGLES
(HIDDEN LINE)

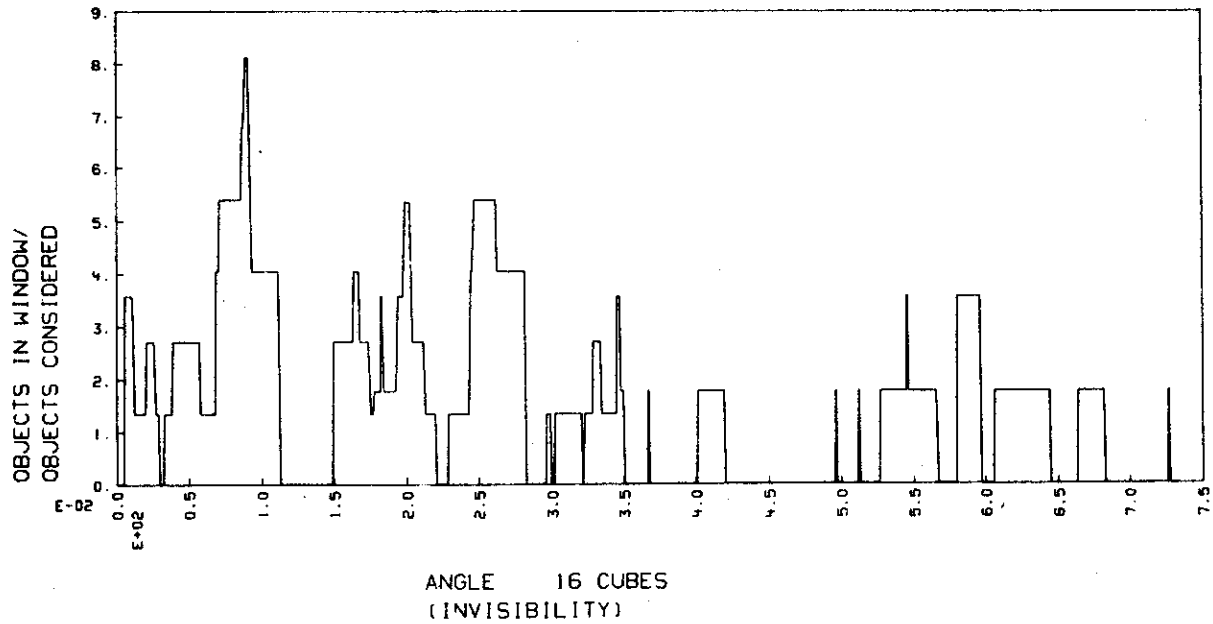


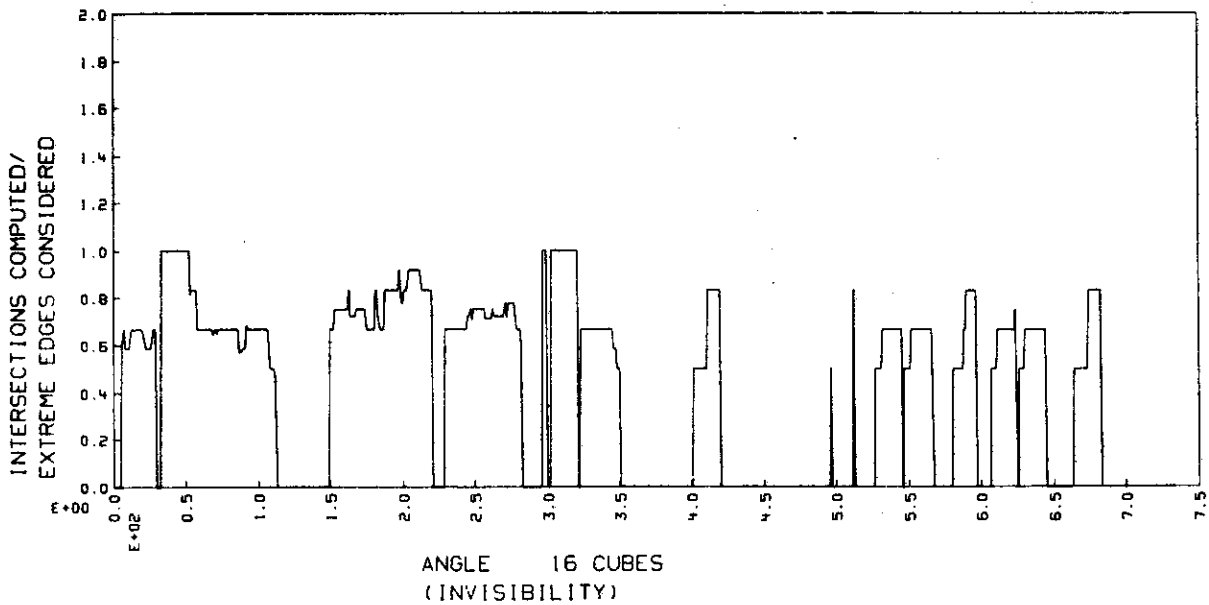
ANGLE 16 RECTANGLES
(HIDDEN LINE)

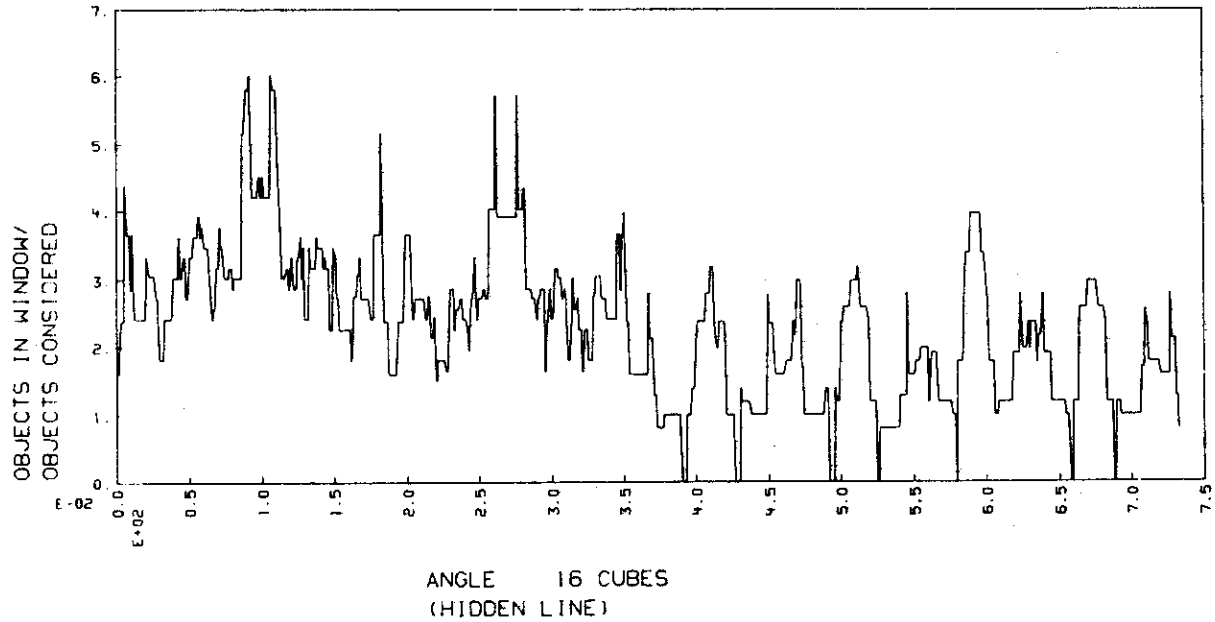


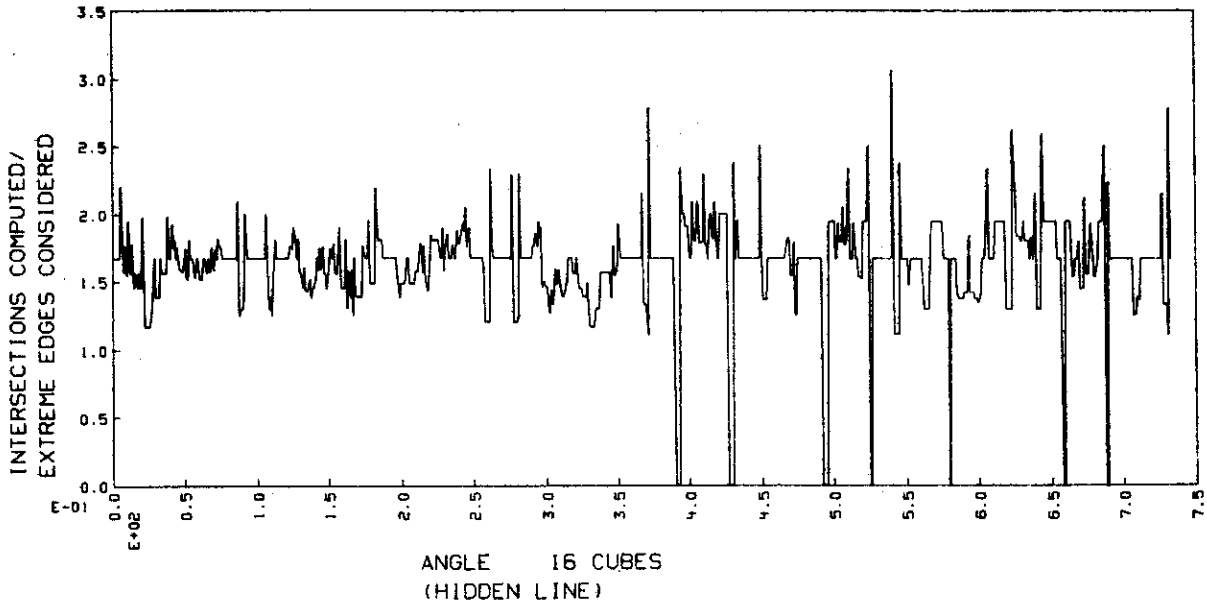
ANGLE 16 RECTANGLES
(HIDDEN LINE)

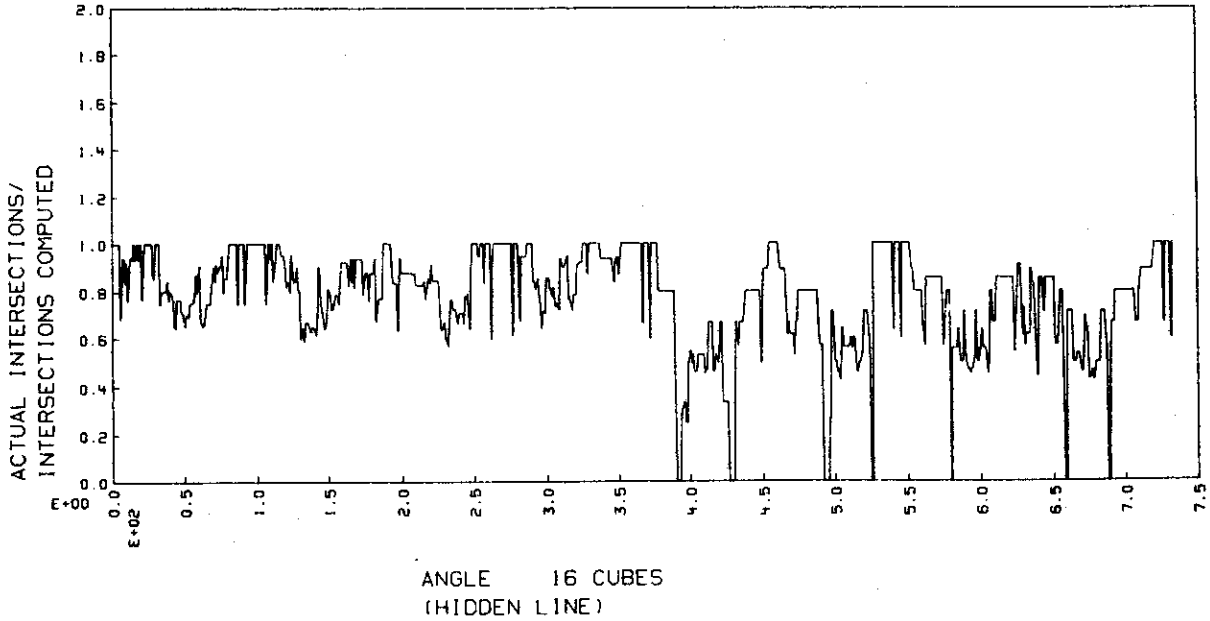

TIME (MILLISECONDS)
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
E+02
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
E+02
ANGLE 16 CUBES

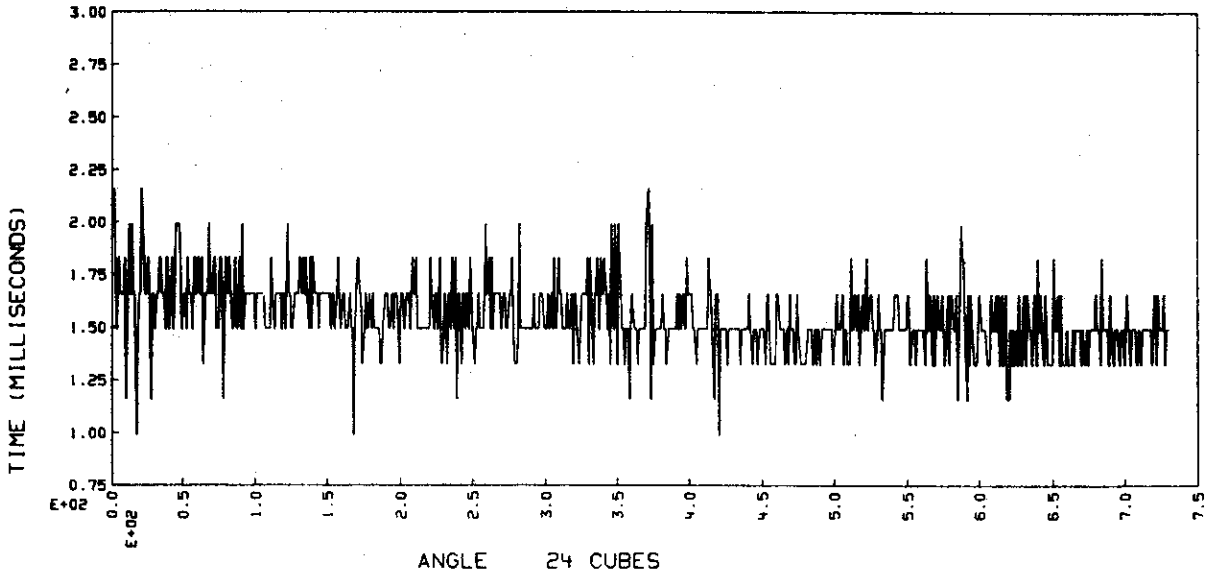

OBJECTS IN WINDOW/
OBJECTS CONSIDERED
9.
8.
7.
6.
5.
4.
3.
2.
1.
0.
E-02
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
E+02
ANGLE 16 CUBES
(INVISIBILITY)

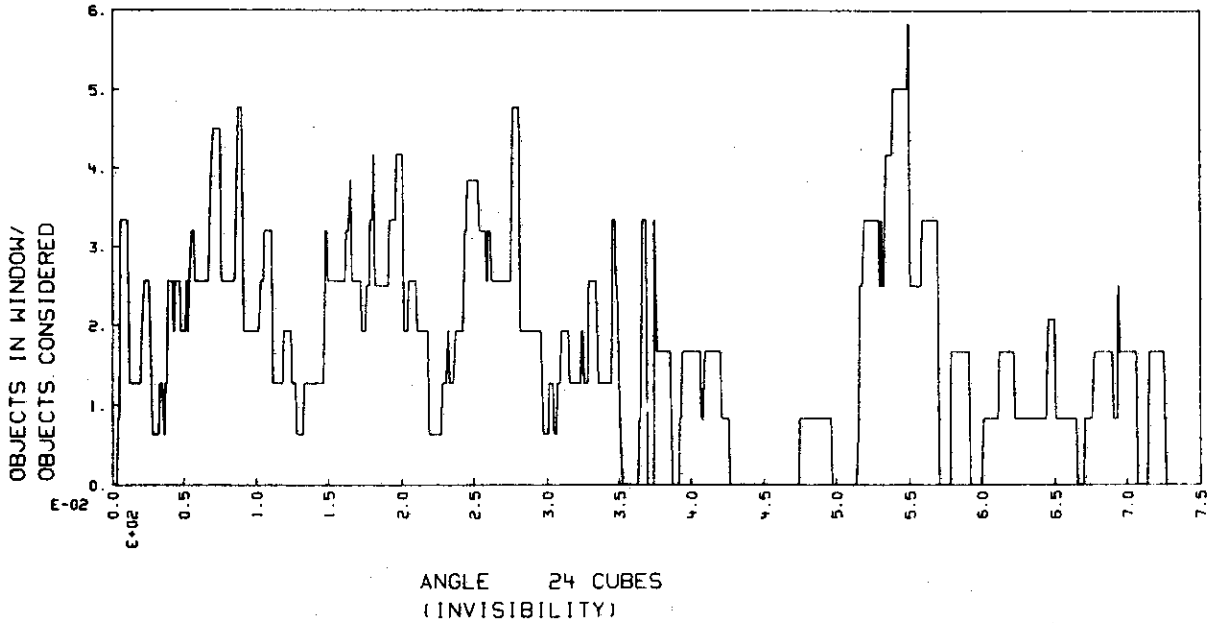

INTERSECTIONS COMPUTED/
EXTREME EDGES CONSIDERED
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
E+00
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
ANGLE 16 CUBES
(INVISIBILITY)

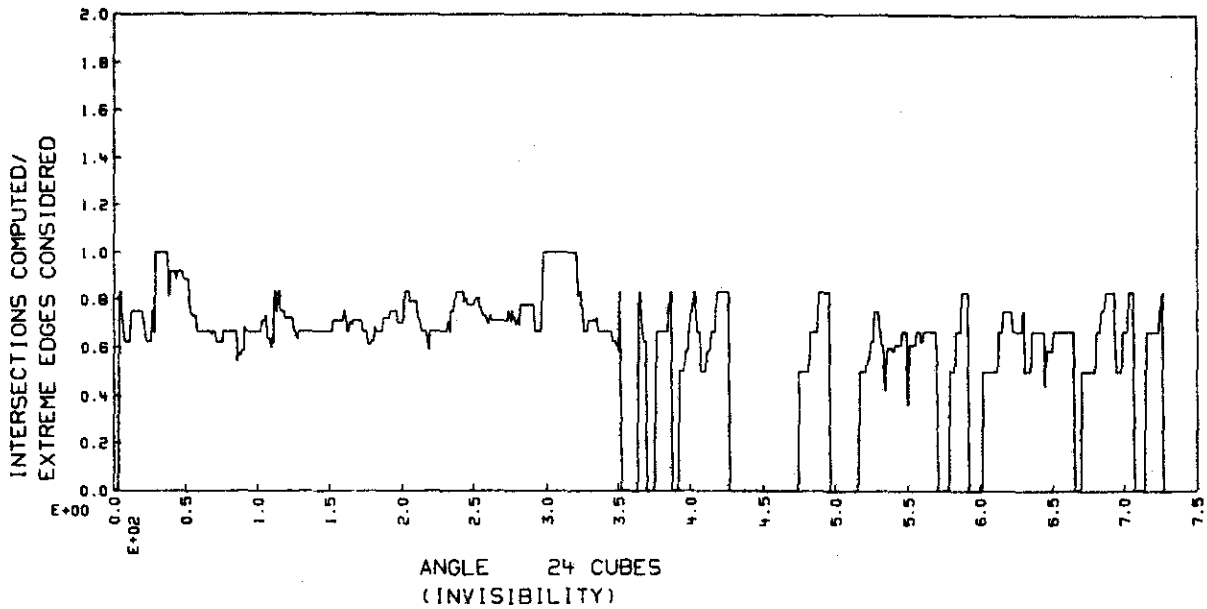

OBJECTS IN WINDOW/
OBJECTS CONSIDERED
7.
6.
5.
4.
3.
2.
1.
0.
E-02
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
ANGLE 16 CUBES
(HIDDEN LINE)

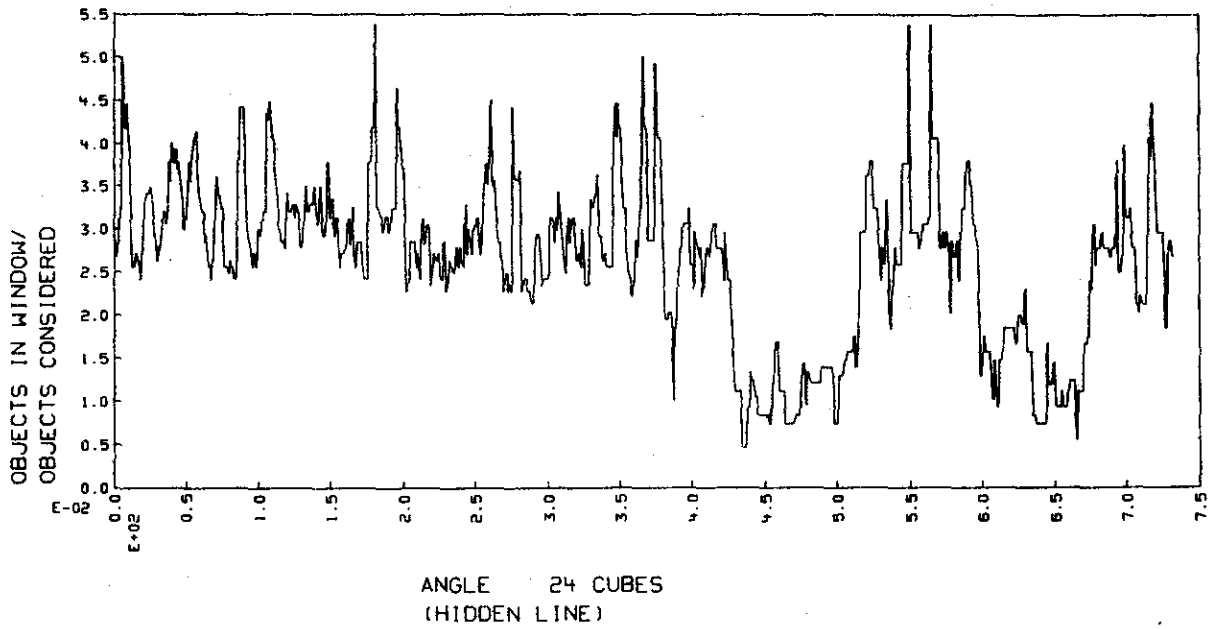

INTERSECTIONS COMPUTED/
EXTREME EDGES CONSIDERED
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
E-01
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
ANGLE 16 CUBES
(HIDDEN LINE)

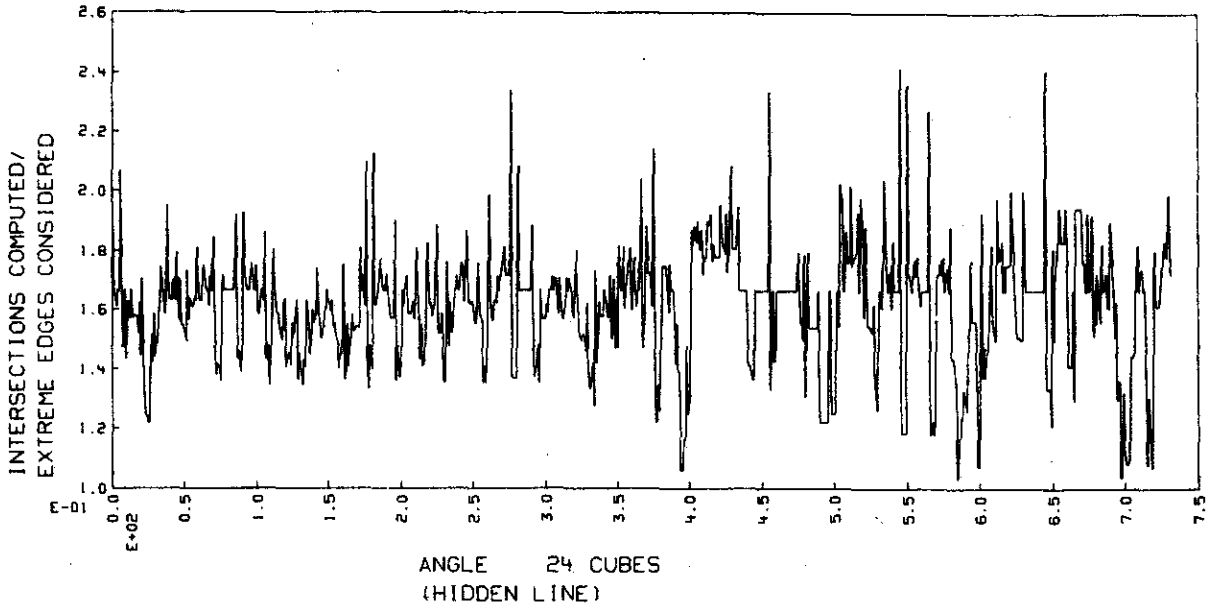

ACTUAL INTERSECTIONS/
INTERSECTIONS COMPUTED
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
E+00
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
ANGLE 16 CUBES
(HIDDEN LINE)

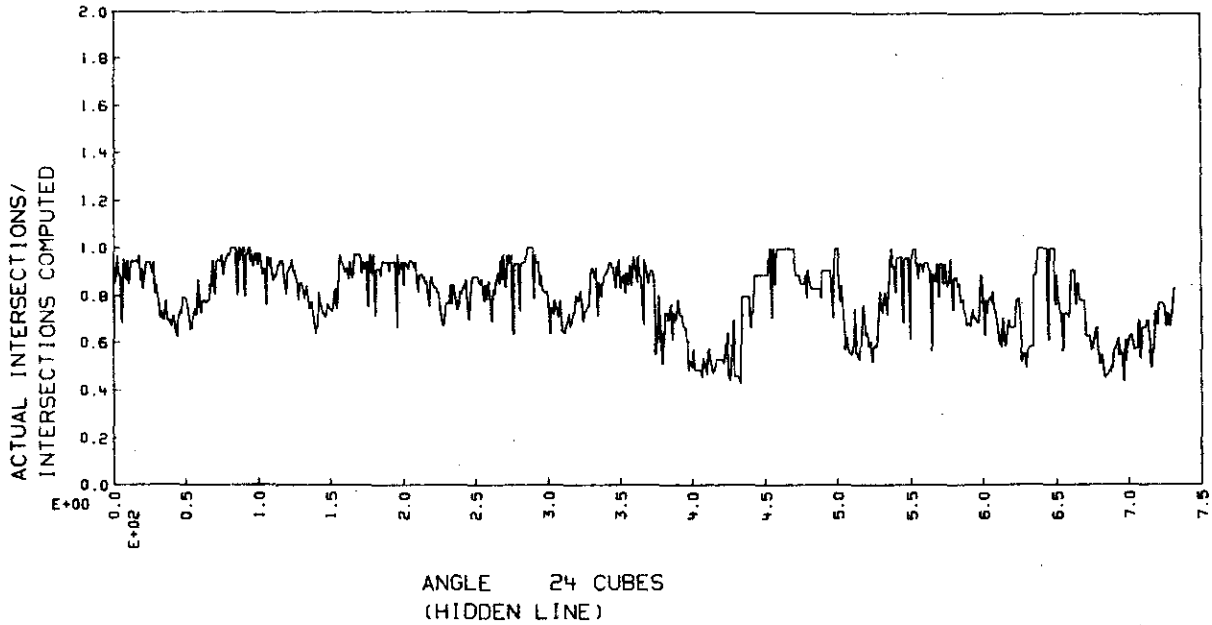

TIME (MILLISECONDS)
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
E+02
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
E+02
ANGLE 24 CUBES

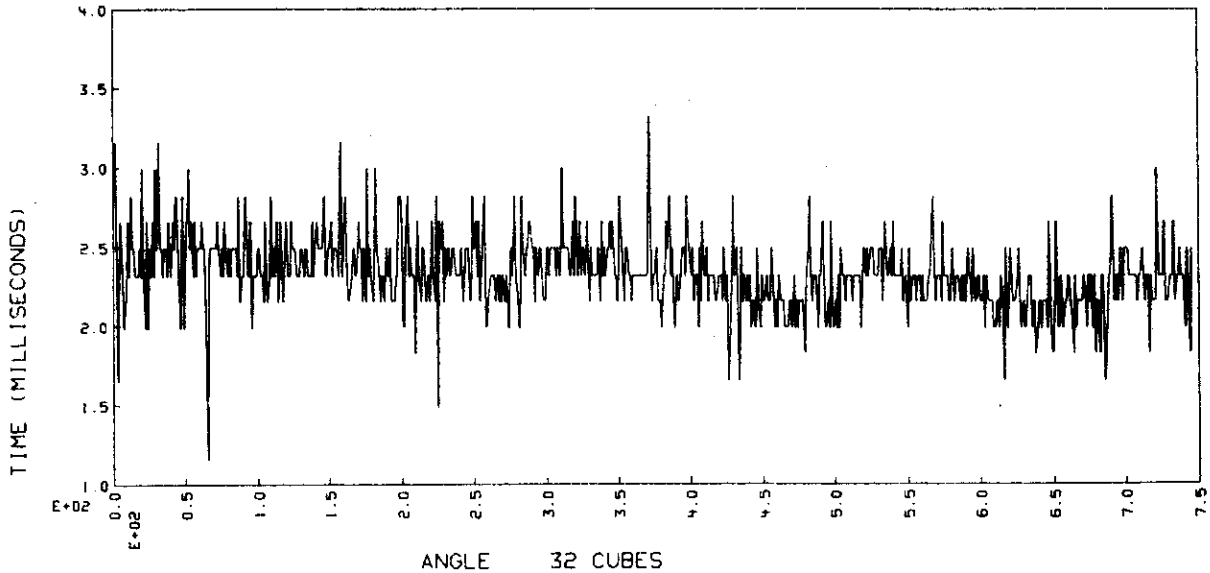

OBJECTS IN WINDOW/
OBJECTS CONSIDERED
6.
5.
4.
3.
2.
1.
0.
E-02
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
E+02
ANGLE 24 CUBES
(INVISIBILITY)

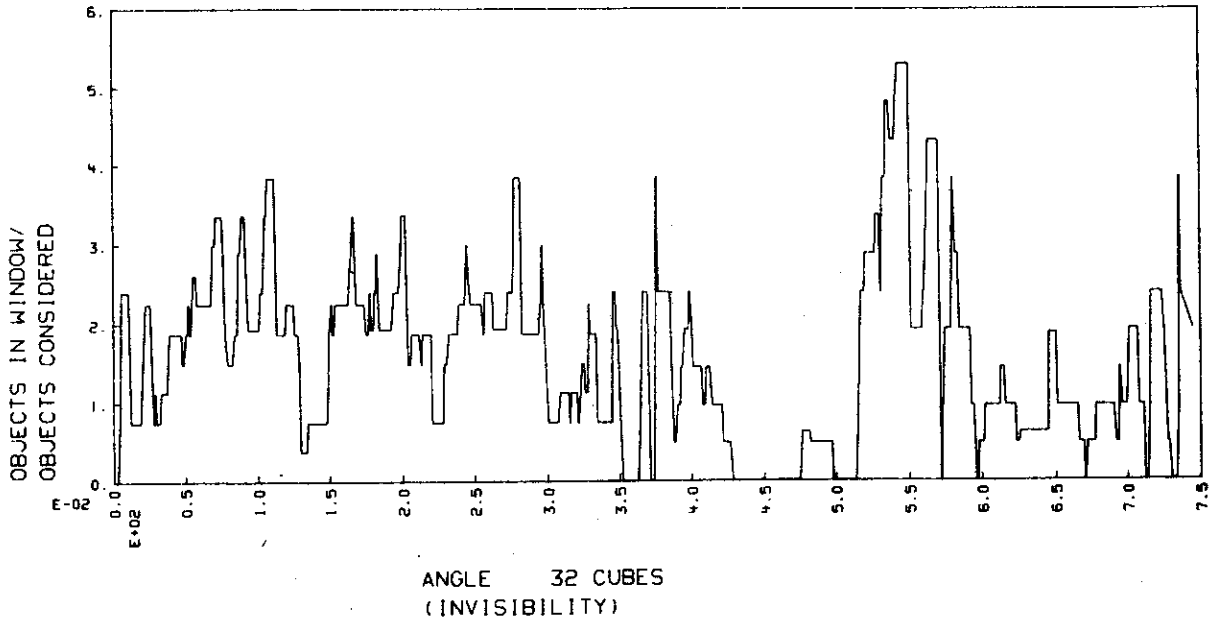

INTERSECTIONS COMPUTED/
EXTREME EDGES CONSIDERED
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
E+00
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
E+02
ANGLE 24 CUBES
(INVISIBILITY)

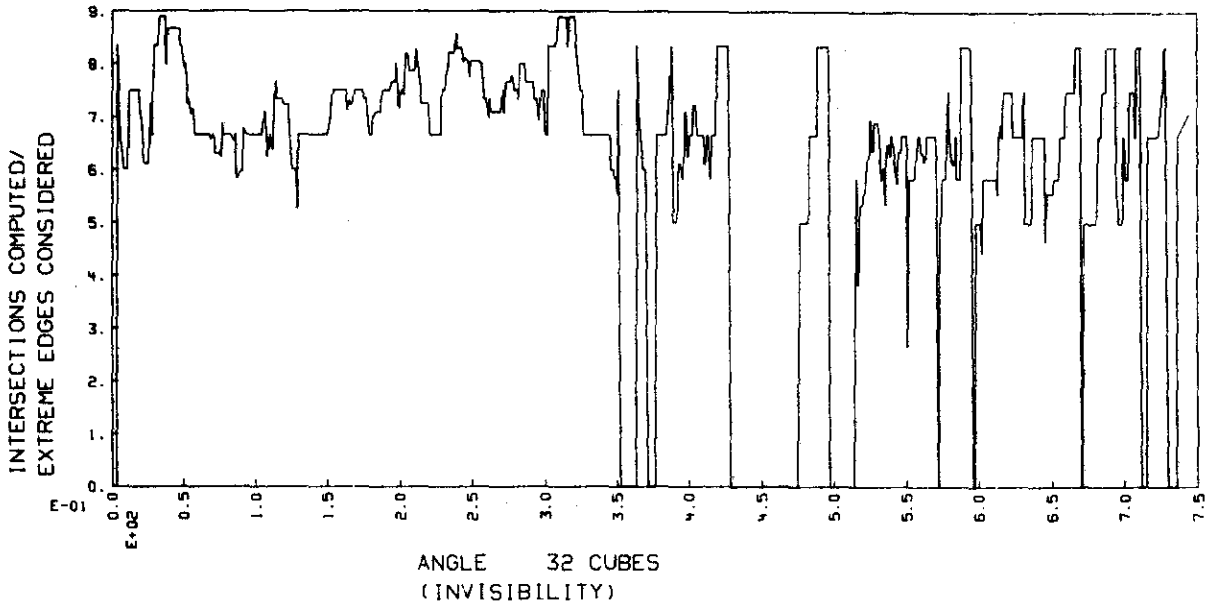

OBJECTS IN WINDOW/
OBJECTS CONSIDERED
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
E-02
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
E+02
ANGLE 24 CUBES
(HIDDEN LINE)

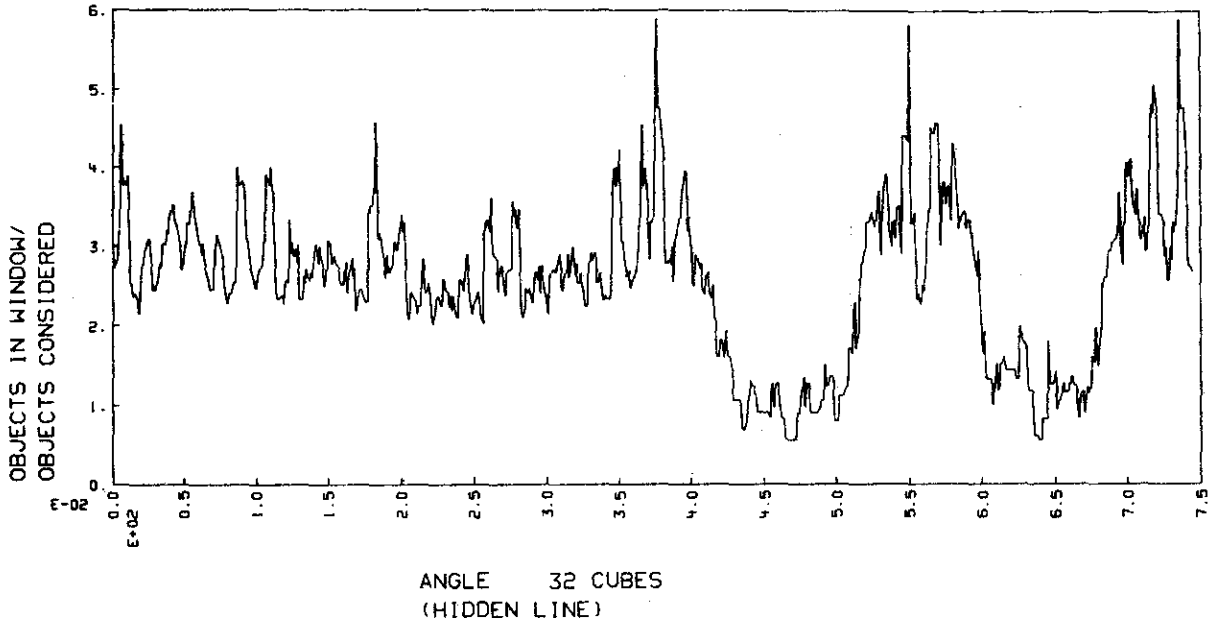

INTERSECTIONS COMPUTED/
EXTREME EDGES CONSIDERED
2.6
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
E-01
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
E+02
ANGLE 24 CUBES
(HIDDEN LINE)

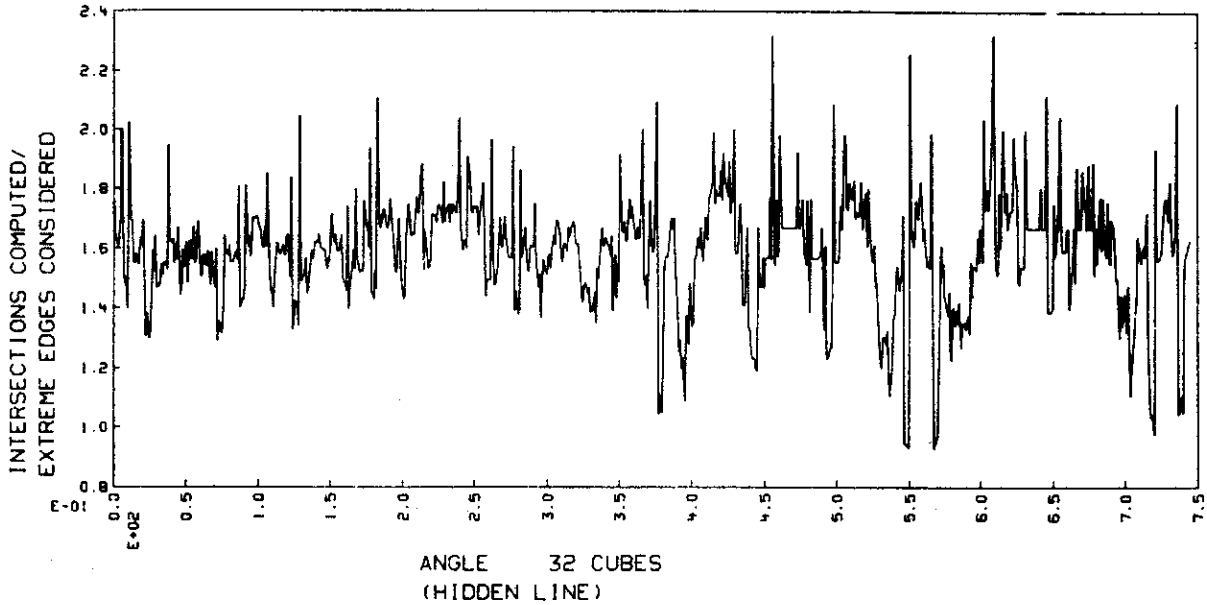

ACTUAL INTERSECTIONS/
INTERSECTIONS COMPUTED
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
E+00
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
E+02
ANGLE 24 CUBES
(HIDDEN LINE)

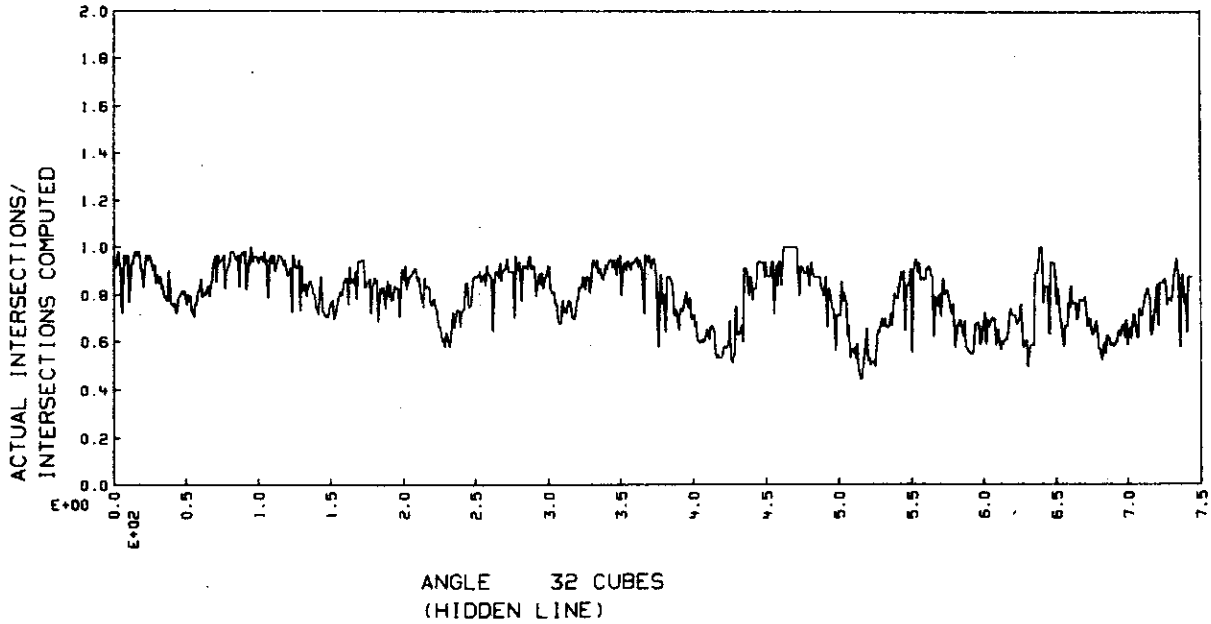

TIME (MILLISECONDS)
4.0
3.5
3.0
2.5
2.0
1.5
1.0
E+02
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
E+02
ANGLE 32 CUBES

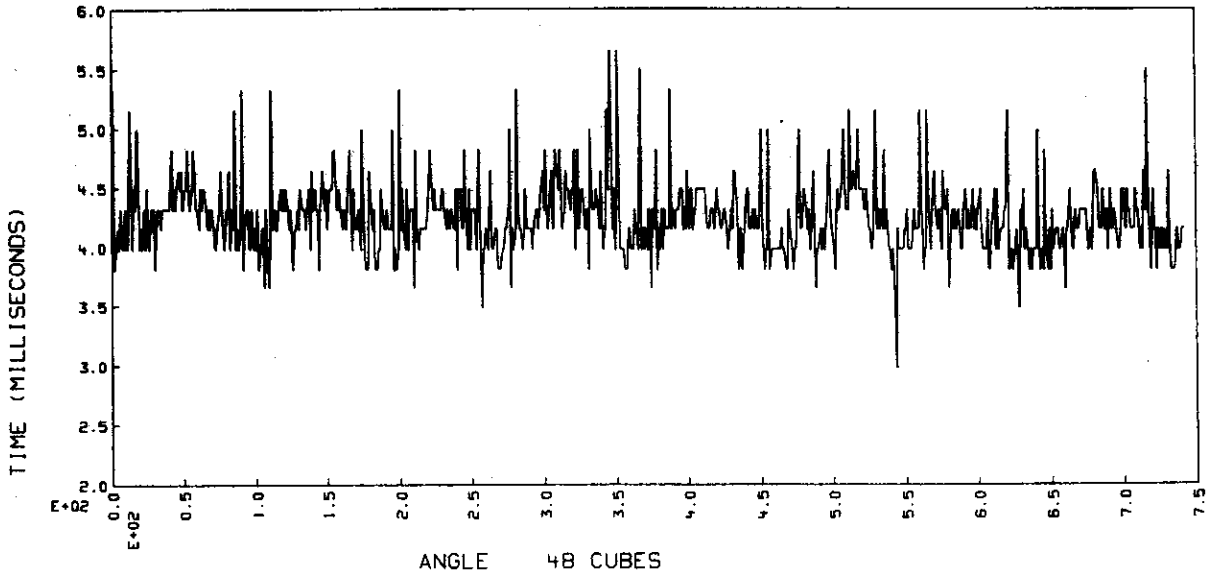

OBJECTS IN WINDOW/
OBJECTS CONSIDERED
6.
5.
4.
3.
2.
1.
0.
E-02
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
E+02
ANGLE 32 CUBES
(INVISIBILITY)

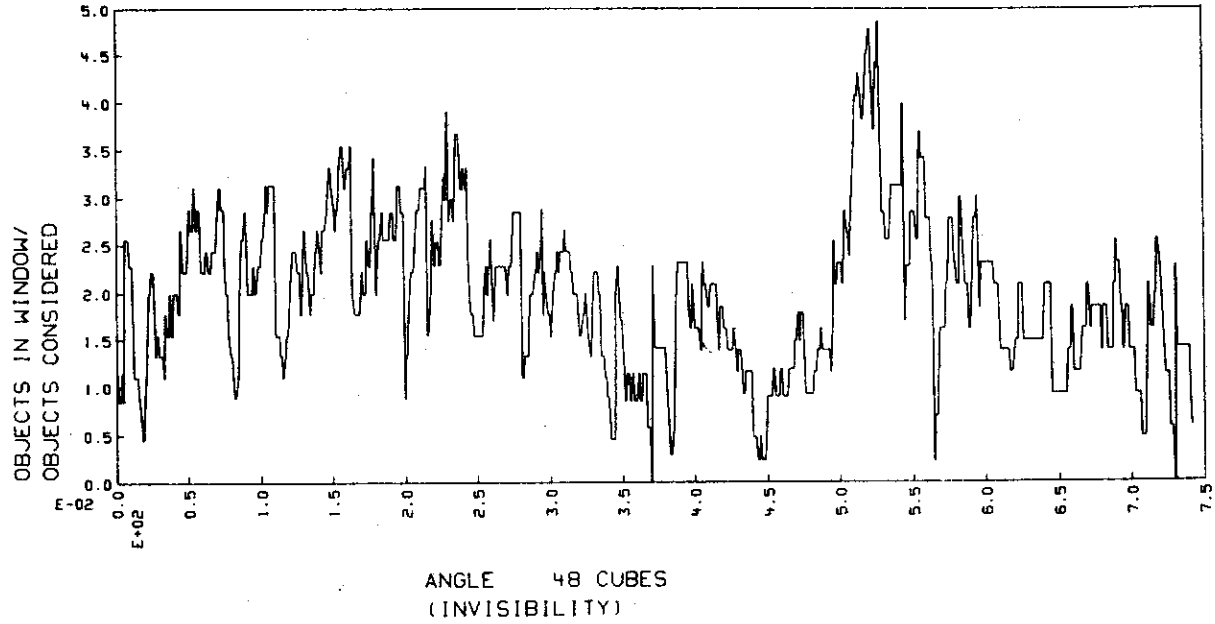

INTERSECTIONS COMPUTED/
EXTREME EDGES CONSIDERED
E-01
E+02
ANGLE 32 CUBES
(INVISIBILITY)

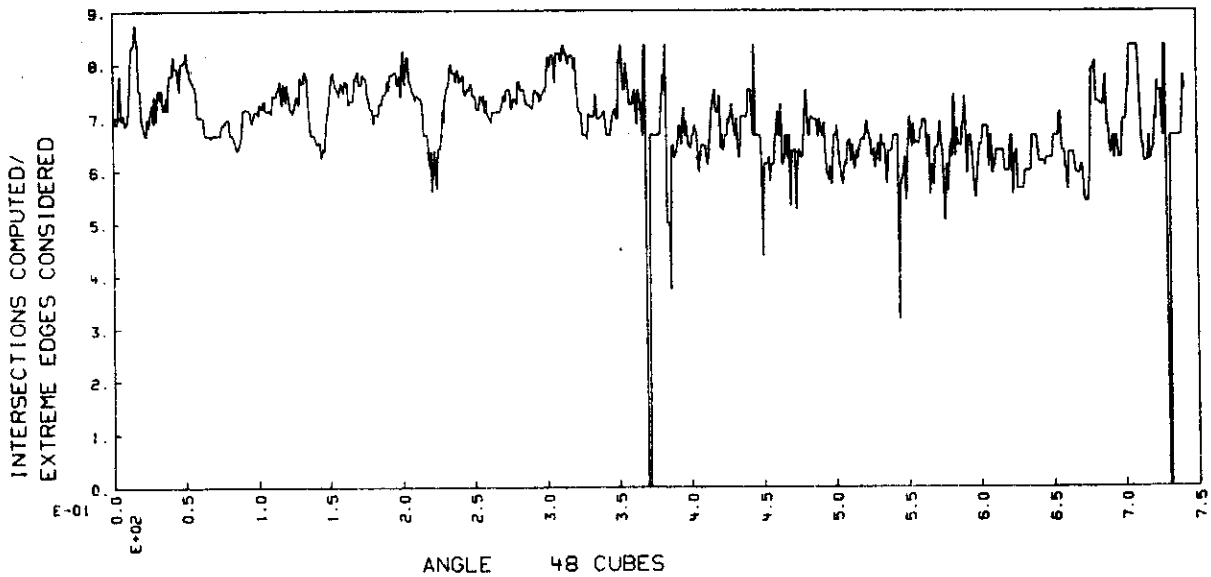

OBJECTS IN WINDOW/
OBJECTS CONSIDERED
E-02
E+02
ANGLE 32 CUBES
(HIDDEN LINE)

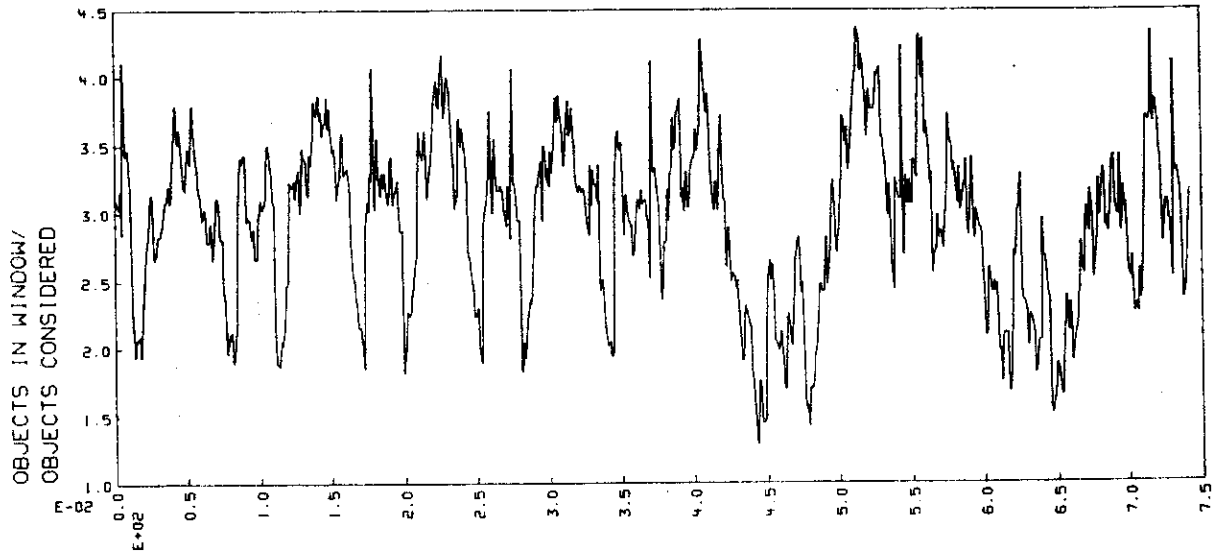

INTERSECTIONS COMPUTED/
EXTREME EDGES CONSIDERED
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
E-01
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
E+02
ANGLE 32 CUBES
(HIDDEN LINE)

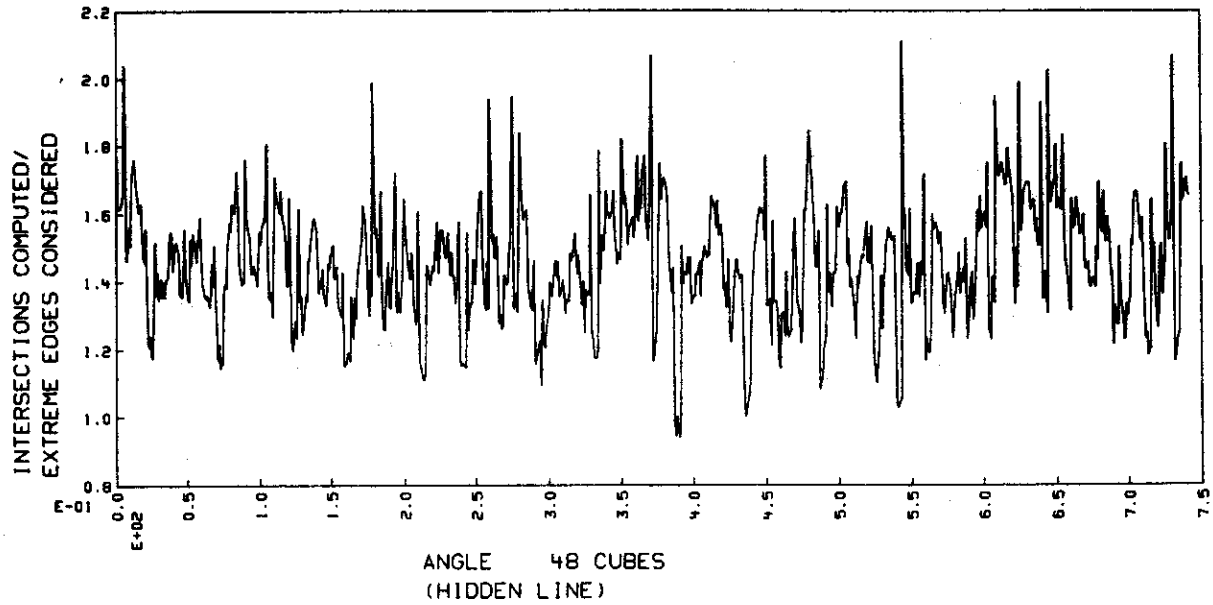

ACTUAL INTERSECTIONS/
INTERSECTIONS COMPUTED
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
E+00
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
E+02
ANGLE 32 CUBES
(HIDDEN LINE)

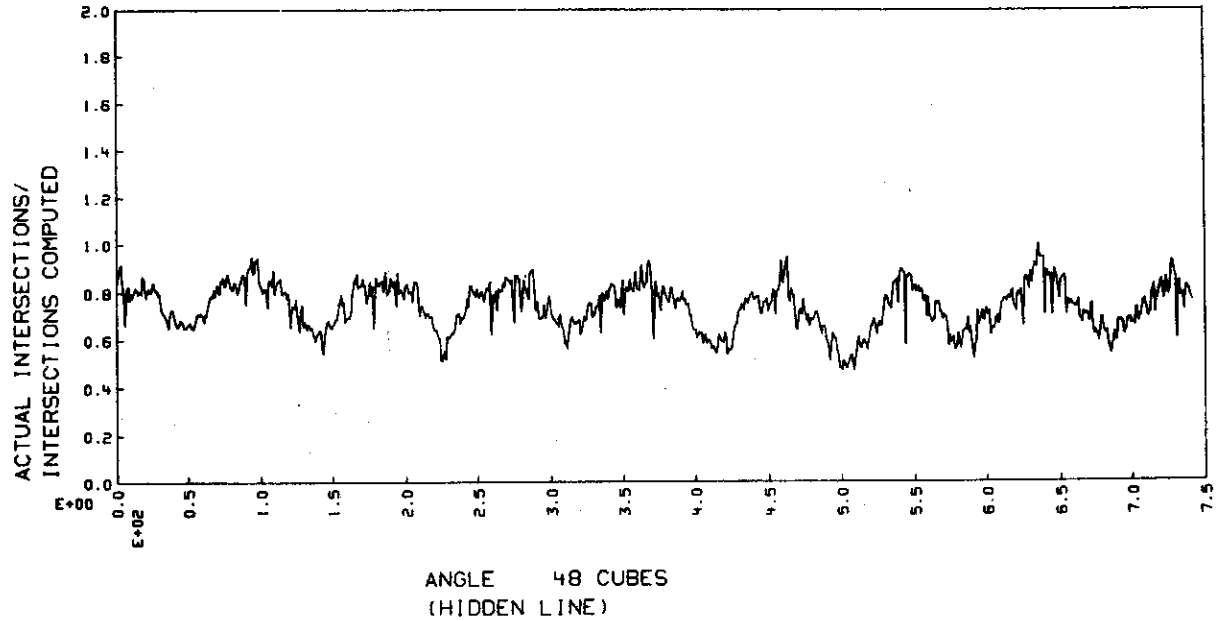

TIME (MILLISECONDS)
6.0
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
E+02
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
E+02
ANGLE 48 CUBES

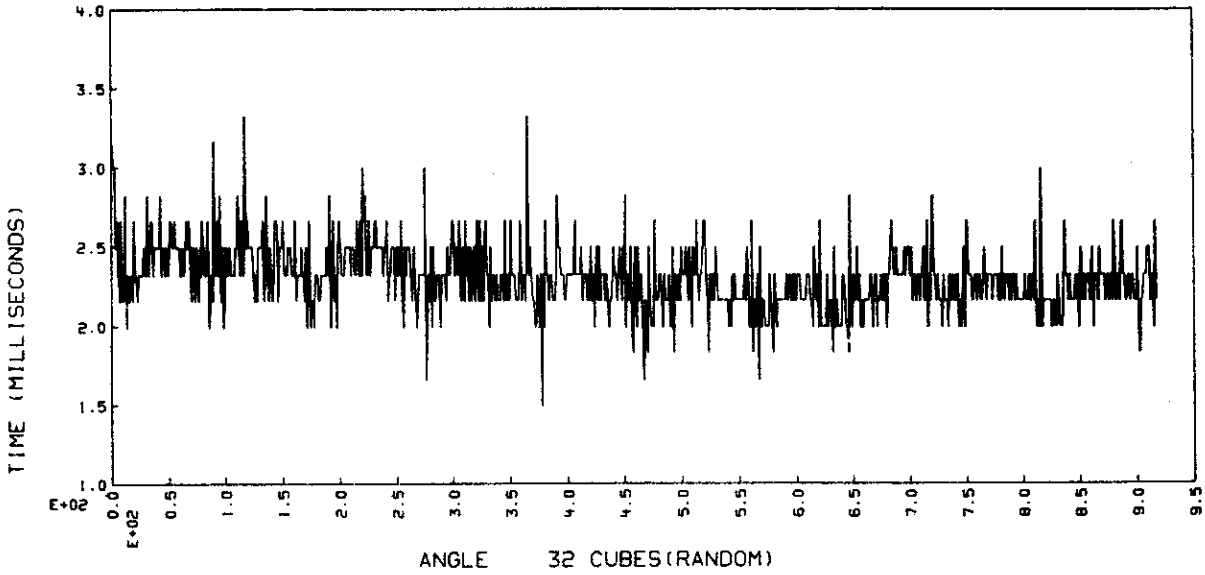

OBJECTS IN WINDOW/
OBJECTS CONSIDERED
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
E-02
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
E+02
ANGLE 48 CUBES
(INVISIBILITY)

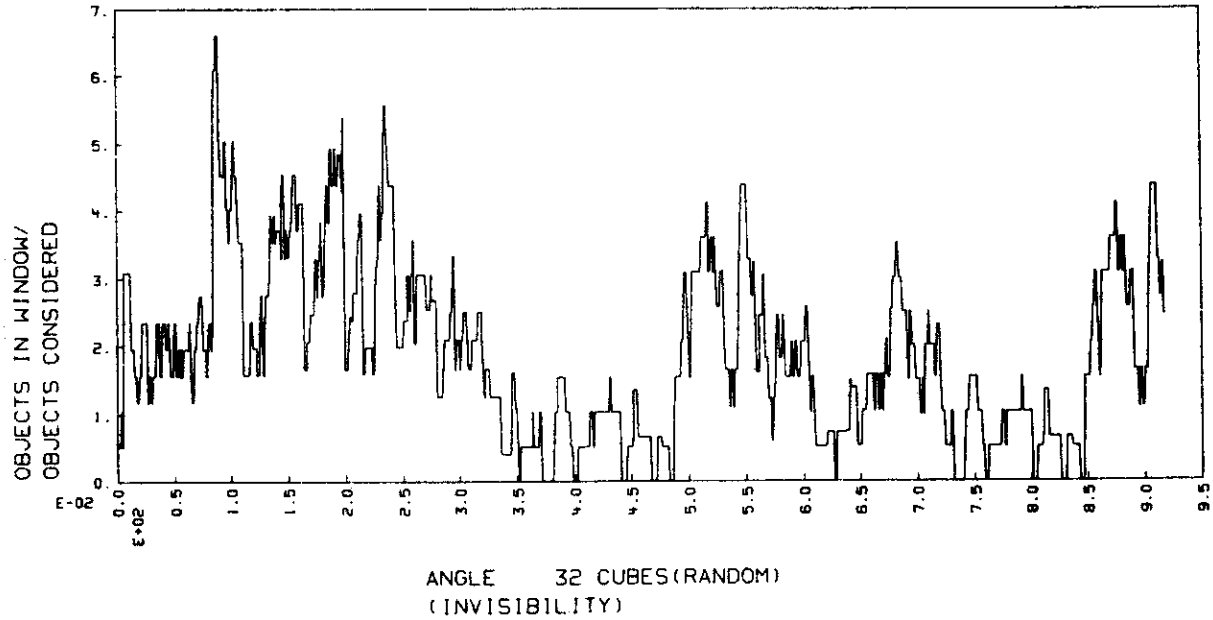

INTERSECTIONS COMPUTED/
EXTREME EDGES CONSIDERED
E-01
E+02
ANGLE 48 CUBES
(INVISIBILITY)

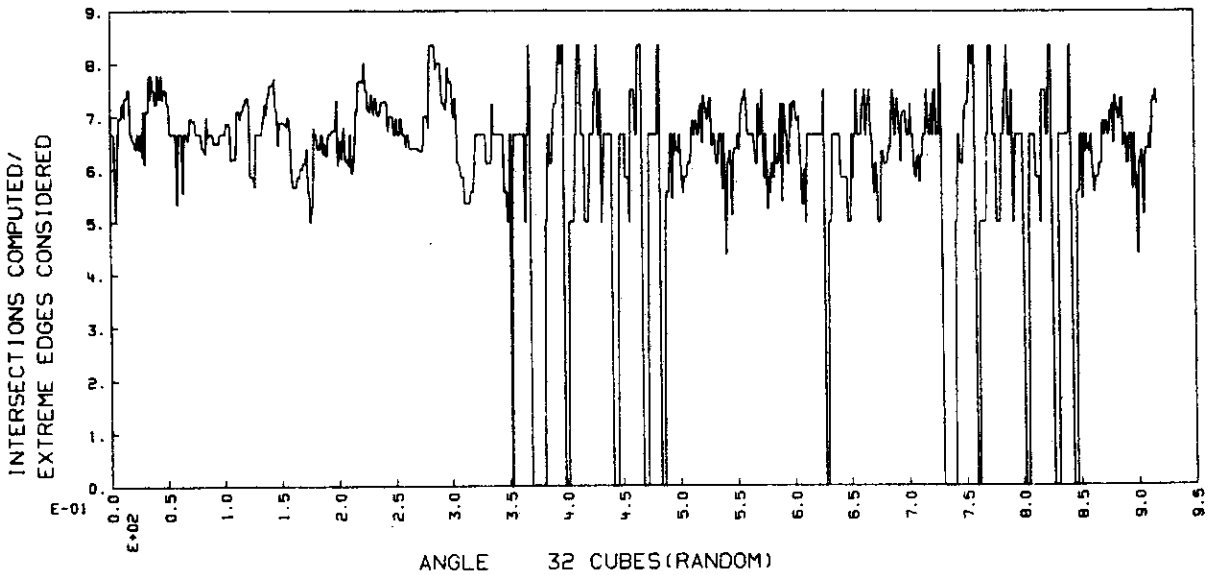

OBJECTS IN WINDOW/
OBJECTS CONSIDERED
E-02
E+02
ANGLE 48 CUBES
(HIDDEN LINE)

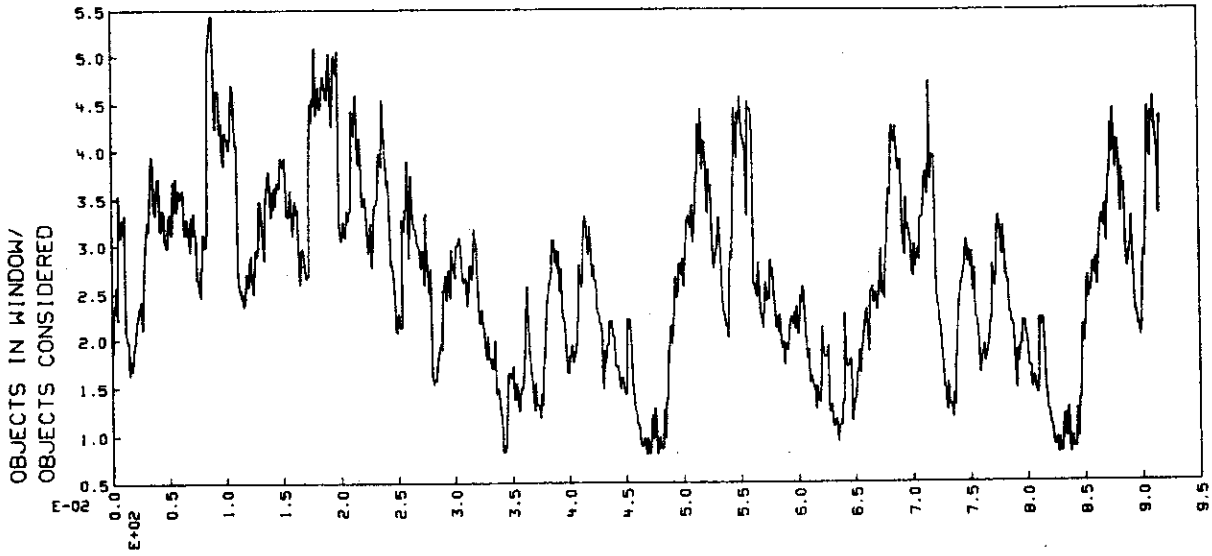

INTERSECTIONS COMPUTED/
EXTREME EDGES CONSIDERED
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
E-01
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
ANGLE 48 CUBES
(HIDDEN LINE)

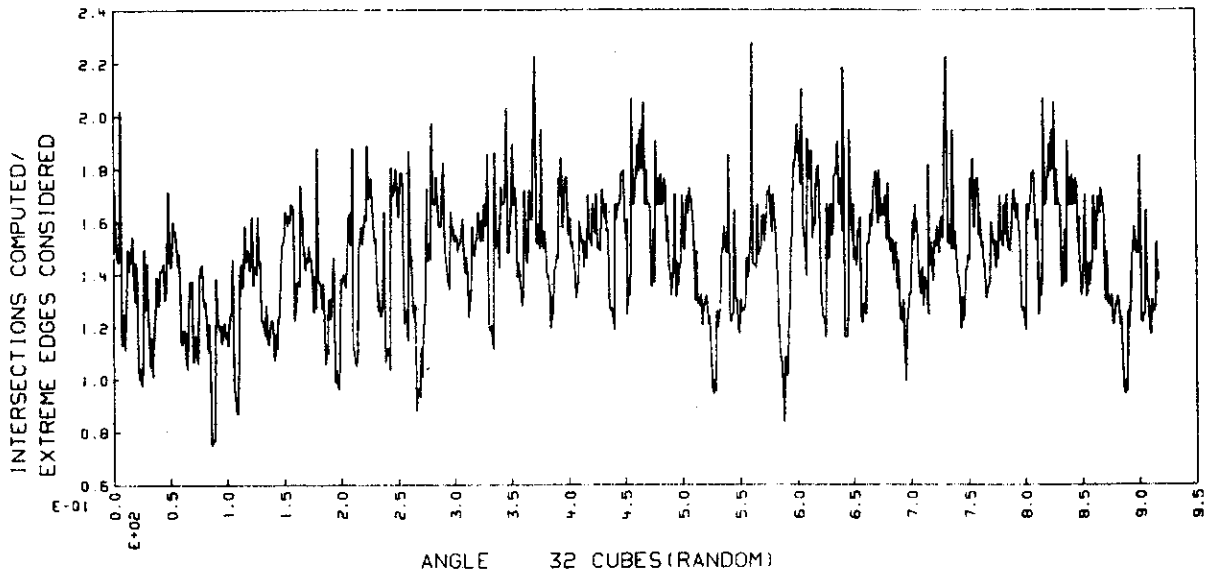

ACTUAL INTERSECTIONS/
INTERSECTIONS COMPUTED
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
E+00
0.0
E+02
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
ANGLE 48 CUBES
(HIDDEN LINE)


TIME (MILLISECONDS)
ANGLE 32 CUBES(RANDOM)

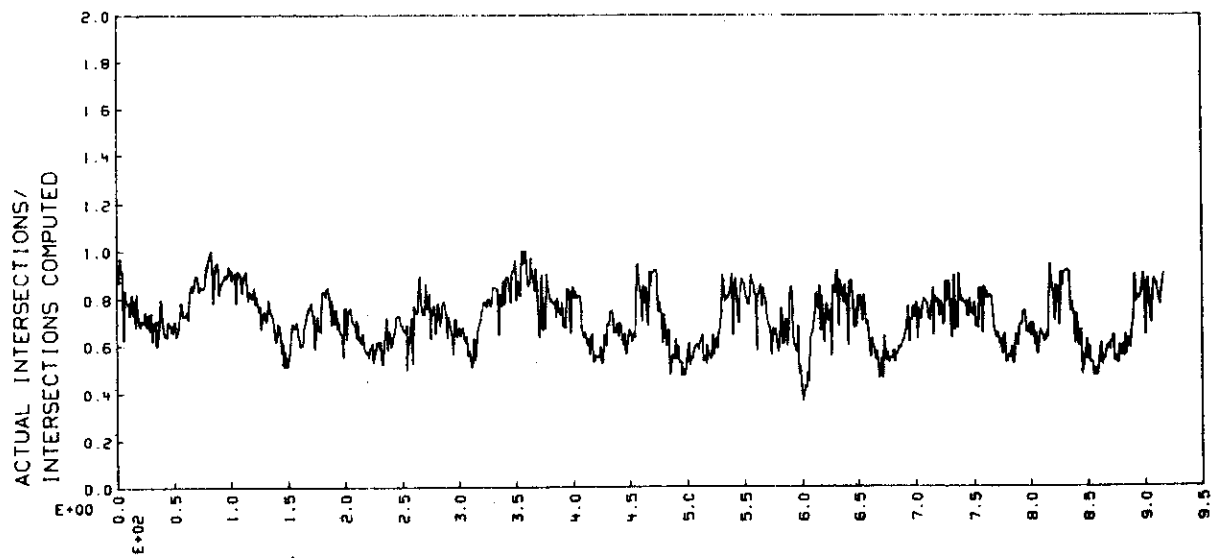

OBJECTS IN WINDOW/
OBJECTS CONSIDERED
ANGLE 32 CUBES(RANDOM)
(INVISIBILITY)


INTERSECTIONS COMPUTED/
EXTREME EDGES CONSIDERED
ANGLE 32 CUBES(RANDOM)
(INVISIBILITY)


OBJECTS IN WINDOW/
OBJECTS CONSIDERED
ANGLE 32 CUBES(RANDOM)
(HIDDEN LINE)



ANGLE 32 CUBES(RANDOM)
(HIDDEN LINE)



ANGLE 32 CUBES(RANDOM)
(HIDDEN LINE)