1986

Separation Plane Priority in Computer Image Generation

Keith A. Stump
University of Central Florida

SEPARATION PLANE PRIORITY IN COMPUTER IMAGE GENERATION

BY

KEITH A. STUMP
B.S.E., University of Central Florida, 1983

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Engineering
in the Graduate Studies Program of the College of Engineering
University of Central Florida
Orlando, Florida

Spring Term

1986

ABSTRACT

Flight simulators are devices in which air crews can be trained without the use of actual aircraft. Potentially dangerous maneuvers, such as air-to-air refueling, and destructive exercises, such as evasive action from weaponry or aerial dogfights, can be practiced repeatedly with no risk to pilot or crew. Flight simulators are cost effective since the fuel costs associated with training pilots in actual aircraft can be excessive. Flight simulators offer an alternate training method with reduced cost.

The task of a visual flight simulator is to present the trainee with scenes representative of those that would be seen if the actual mission being trained for were flown. Scenes produced by a Computer Image Generation device must be of sufficient content, fidelity, resolution, brightness and field of view to allow the trainees to improve their skills. If one of these factors falls below the threshold of acceptability, the training value of the device is diminished, if not lost altogether.

One of the most challenging problems in Computer Image Generation is the removal of hidden parts from images of solid objects. In real life, the opaque material of these objects obstructs the light rays from hidden parts and prevents us from seeing them. In the computer generation of an image no such automatic elimination takes place. Instead, all parts of every object, including parts that should be hidden are

displayed. In order to remove these parts and create a more realistic image, a hidden-line or hidden-surface algorithm must be applied to the set of objects. When more than a single object is in the scene another problem arises; which of the objects block the view of the others. This is an occultation problem.

This paper presents a "separation plane" priority algorithm used in Computer Image Generation to solve this occultation problem. The algorithm uses a binary search technique to generate a "listable set"; a set of planes that yield proper object priority for any viewpoint in the data base.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

INTRODUCTION

The use of simulation for operator training and engineering research has become widely accepted in connection with aircraft, spacecraft, tanks, automobiles and ships. In many applications, simulation is an economic necessity. Traditionally, operators of large equipment or special-purpose vehicles had to be trained using actual equipment. With the trend of increased vehicle complexity, this tended to be extremely costly in fuel, operating costs and equipment costs. The cost of operating a given simulator is far less than the corresponding cost of operating the vehicle that it simulates for equivalent training effectiveness.

In other cases, simulation may be the only way to accomplish a particular task. The training of astronauts to land on the moon or the evaluation of the dynamics of a new airplane design before it is built would be impossible without simulation. In most cases, intangible benefits add greatly to the acceptance of simulators. In particular, the ability to conduct intensive emergency procedure training in a simulator without endangering the operator, the instructor, the vehicle, or innocent bystanders allows more effective training than could be accomplished in the actual vehicle, regardless of economic considerations.

Today's highly sophisticated simulators use real-time computer-generated imagery for visuals and numerical processes that rival that of the most sophisticated hardware. These Computer Image Generation Systems (CIG) simulate the visual environment external to a vehicle as

it would be seen by an observer inside the vehicle looking out through a window. As the trainer maneuvers his vehicle these views change so as to maintain an accurate impression of the vehicle's motion. The environment is represented by the combined use of unbounded plane surfaces and solid objects. The unbounded surfaces are described by a nested hierarchy of stylized texture patterns, and the solid objects, which are convex polyhedra, are described numerically in terms of their size, shape, location and color.

The CIG requires inputs describing the attitude and position of the observer with respect to the environment. Based on these inputs and the stored numerical model of the environment, the CIG computes the perspective transformation of the environmental model onto a bounded plane that represents the observer's window. The transformation is presented to the observer as synthetic video by means of a raster-scan color display. The image must be updated at a rate such that continuous, jitter-free motion is achieved. This rate is usually greater than 30 frames or updates per second (Steiner 1985; Schachter 1983).

The particular uses to which the CIG is applied place heavy emphasis on dynamic fidelity, flexibility in scene content, and proper perception of solid objects. These three requirements, within the context of available digital devices, dictate the particular approach and hardware organization used in the CIG to generate perspective images of solid objects. Special purpose computers must be designed to handle the large computational requirements and high data rates needed. A typical simulator can cost anywhere from $1 million to $100 million (Steiner 1985).

To simplify the graphics algorithms as much as possible, surfaces are modeled with polygons. Higher order parametric equations which describe a surface are much more expensive in terms of computer time than linear equations that describe polygons.

Solid-object display is the most analytical branch of computer graphics and over the years has inspired some of the most original research work. This paper will discuss one concept that provides proper occultation (the determination of which objects conceal one another) when more than one solid object is present in the environment. Chapter II discusses occultation in more detail and discusses the algorithm used most widely in Computer Image Generation. Chapter III discusses the algorithm this writer developed to calculate the minimum number of separation planes used in the occultation technique. Chapter IV gives examples of a program's use of the algorithm and Chapter V discusses conclusions made.

OCCULTATION

If the environment contains more than one object, provisions must be made in the image generator to allow objects to occult one another properly. In conventional computer graphics terminology, this is a hidden-line or hidden-surface problem. The reader will notice, however, that there is a significant difference between the hidden-line problem and the occultation problem.

When the determination is made as to which parts of the object will be concealed, as they will be if they face away from the observer or are obscured by other parts of the object; this is called the hidden-surface problem and is one of the classic problems of computer graphics (Newman and Sproull 1979). When it must be determined which parts of many different objects will be seen from various viewpoints, this is an occultation problem.

In the early 1960s, when the first hidden-line algorithm was developed, displays were exclusively line-drawing devices. The earliest such algorithms, of which Roberts' was the most prominent, were extremely slow (Newman and Sproull 1979). When raster displays became available, attention shifted to hidden-surface removal and many techniques, both hardware and software, were developed.

In general, visibility problems can be attacked either in the object space or in the image space. A solution in the object space focuses on geometric relations among the parts of the objects in order to decide what is visible and what is not. A solution in the image

space traverses the area of the picture and examines the projections as they occur (Pavilidis 1982).

An object-space algorithm performs geometric calculations with as much precision as possible, usually the precision available in floating-point hardware of the computer. Since the precision of the solution is much greater than that of a display device, the image can be displayed enlarged many times without losing accuracy. By contrast, image-space algorithms perform calculations with only enough precision to match the resolution of the display screen used to present the image. These algorithms simply calculate an intensity for each of the 250,000 or 1 million distinct dots on the screen (Newman and Sproull 1979).

In an excellent survey paper by Sutherland, Sproull, and Schumacker, the authors discuss and categorize ten hidden-surface algorithms developed through 1972 (Foley and Van Dam 1982). Additional algorithms have been developed since 1972. Despite the existence of many algorithms, there is no single answer to the hidden-surface problem and no best algorithm. Many of the differences between algorithms stem from different requirements; the algorithms operate on different kinds of scene models, generate different forms of output, or cater for images of different complexities.

Priority Methods

The objective here is not to discuss hidden-surface algorithms, but rather give an overview of the visibility algorithms used for CIG simulation. Most CIG algorithms use priority methods, which are probably the most frequently used of the available hidden-surface algorithms (Giloi

1978). The selection of this method is based on the high computational rate required for visual simulation.

The idea of the priority algorithm is to arrange all polygons in the scene in priority order based on their depth. Polygons nearer the viewpoint will have higher priority than those far away. After priority has been determined, polyhedrons are scan-converted one at a time into a frame buffer, starting with the polyhedrons of lowest priority. This procedure will generate a correct hidden-surface view provided the priority order is properly computed. To explain the determination of the priority order we begin the discussion with a single simple polyhedron.

A single wire-frame polyhedron can be drawn very simply if hidden lines are not to be eliminated; that is all polygonal faces are simply drawn in any order. This, however, does not provide an adequate depth cue and ambiguities can arise. As shown in Figure 1, different perceptions of the same object can be made. This is not adequate for simulation application, which requires extreme realism.

A daylight flight simulator that uses computer-generated images of the view from the cockpit must generate very realistic pictures. Pilots seem to depend on subtle visual cues for depth perception, such as skid marks on a runway. The relative depth of objects in a scene is readily apparent if the lines that are hidden from view by opaque objects are removed from the image. This technique requires considerable computation but is nevertheless required for producing finished pictures of a scene.

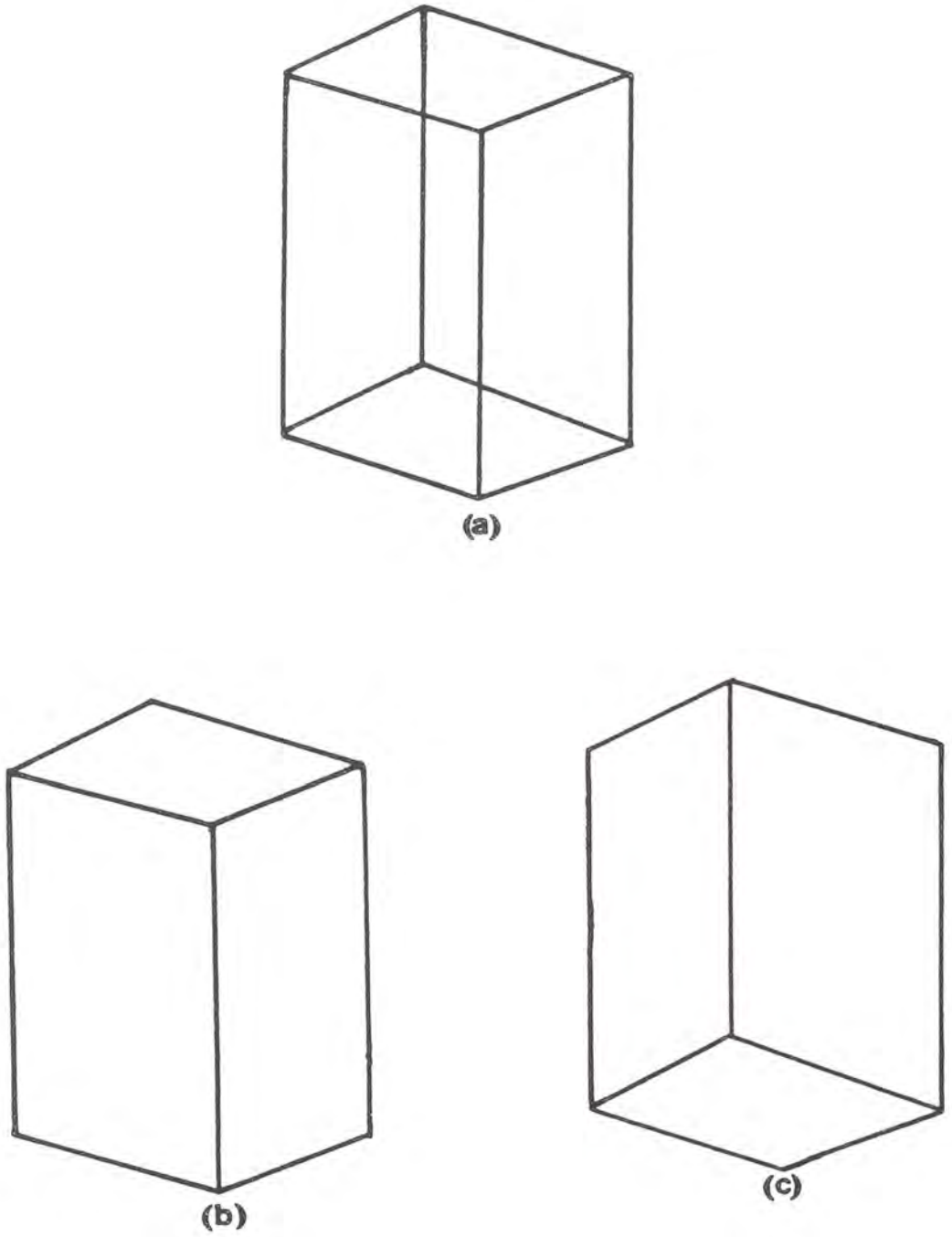

Figure 1. Ambiguities Can Arise from Wire-Frame Drawings. The Object (a) Can Be Either (b) or (c).

Back-Facing

One way of eliminating hidden-surfaces is to identify back faces; polygons that cannot possibly be visible because they lie on the side of an object facing away from the viewpoint. If back faces are to be eliminated, face normals must first be computed. Then, the angle between a face normal and the line of sight is inspected. If this angle is obtuse, the face is hidden from the observer's view and can be deleted. This technique, called back-face elimination, a simple use of object coherence, typically halves the number of polygons that need consideration during the generation of a hidden-surface image.

In some cases, eliminating back faces solves the hidden-surface problem immediately. If the scene consists of exactly one convex polyhedron, the elimination of back faces eliminates all the hidden surfaces. A cube, for example, can be displayed realistically by simply removing back faces. This is an extreme example of object coherence that makes processing convex polyhedra particulary easy.

When groups of polyhedra are to be drawn, a much more formidable problem immediately arises; determining which portions of each polyhedron are occulted by other polyhedra. Much effort has been expended on this problem in recent years and several solutions now exist (Bennett 1983). Back-facing faces of all polyhedra can be identified with the simple face normal test, so that if the data base is composed of only convex polyhedra, which it is, the occultation problem is only that of determining which polyhedra are in front of which others. A method devised by Schumacker allows a large part of the work of hidden-surface elimination to be done off-line and incorporated into the data base (Schachter 1983).

Separation Planes

The priority order is determined by a set of invisible "separating planes," which form pockets containing the objects. An example is shown in Figure 2. First, a single major plane is passed through the collection of polyhedra, dividing it in two, without cutting through any polyhedron. (It may be necessary to predivide a polyhedron in two to satisfy this last requirement.) In each of the resulting halves of the space, a plane is drawn dividing the polyhedra into two groups. This process is repeated until each pocket has only one polyhedron. This structure is stored as a binary tree, like the one shown in Figure 3, with the major plane at the top, and the polyhedra, as leaves, at the bottom. The coefficients for the equation of each plane are stored with the record for this plane. In real time the eyepoint is known, and this structure is operated upon to determine on which side of each plane the eye is on. The polyhedra on the same side of the plane as the viewpoint will have the highest priority. A tree-search algorithm yields a list of polyhedra, ordered outward from the observer's eye. An additional example can be seen in Figure 4 where separating planes X and Y divide space into four regions. If the viewpoint is in region A, the polyhedra priority order is 1, 2, 3, 4; in region D, it is 3, 4, 1, 2, etc. Once separation planes have been located, the priority calculation is straight forward.

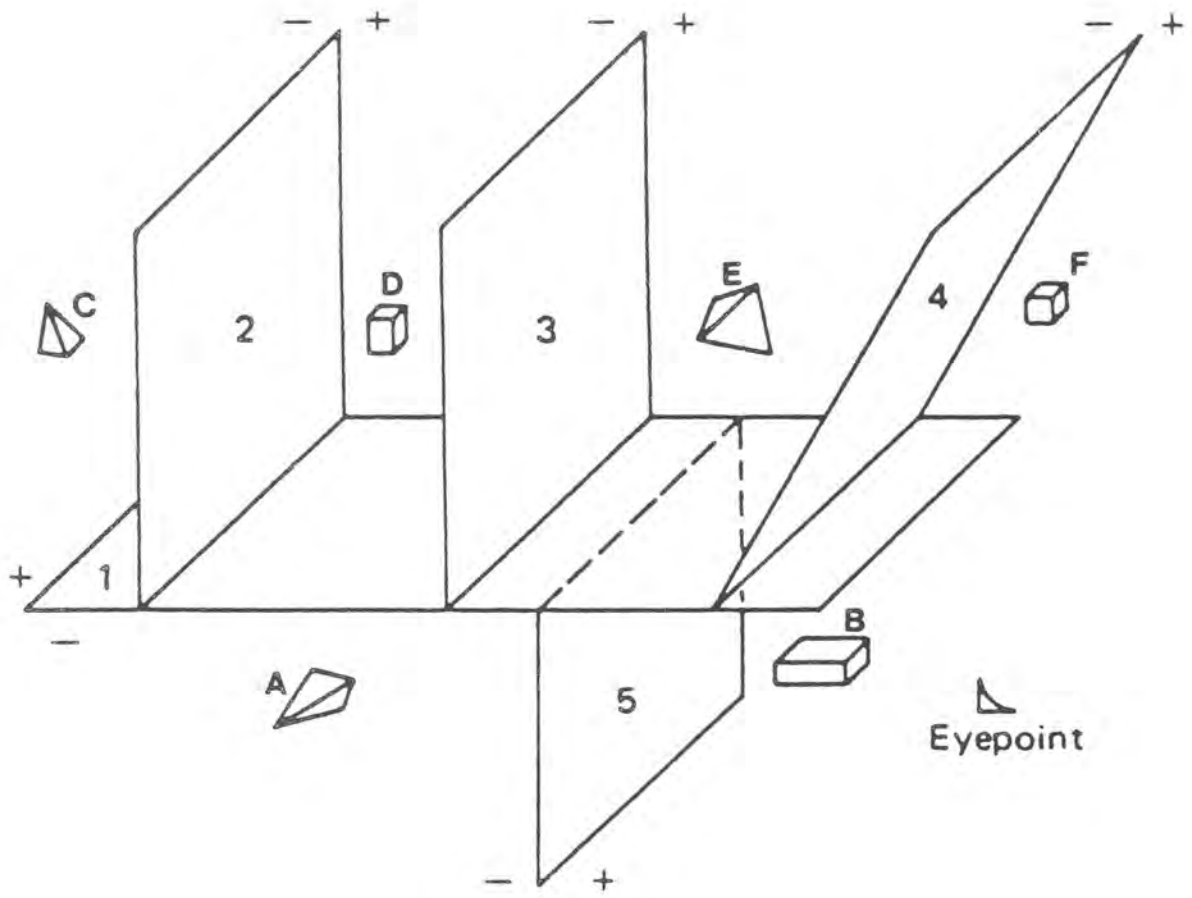

Figure 2. Separating Planes.

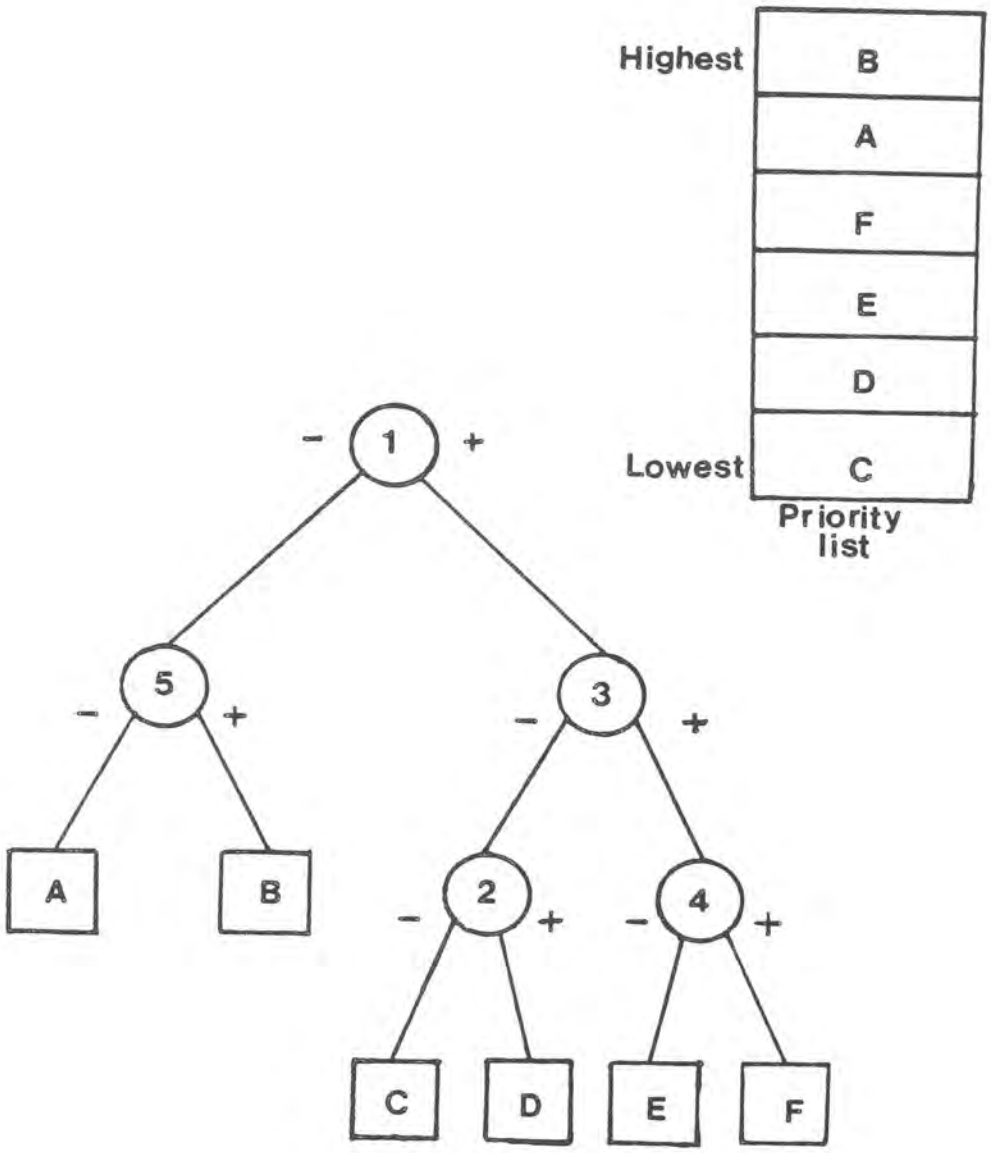

Figure 3. Separating Plane Tree. Numbers Denote Separating Planes and Letters Denote Objects.

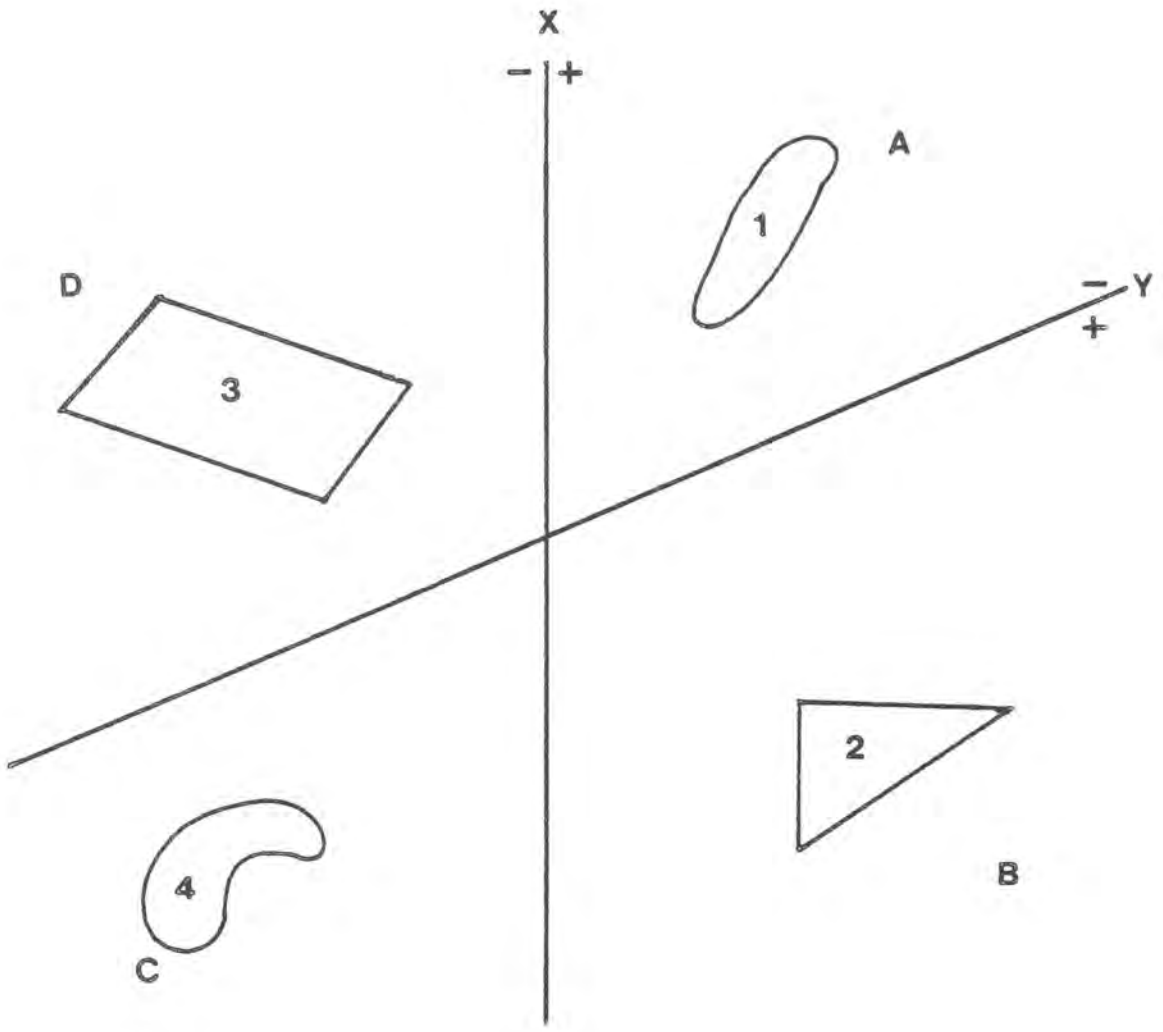

Figure 4. Separating Planes. Two Separating Planes, X and Y, are Used to Determine the Priority Order of Objects from the Viewpoint Location.

Clusters

This method of calculating priority can be further enhanced by building the scene with a collection of parts localized well enough in space to allow planes to separate them. This is the principle of "clusters" proposed by Schumacker in which the order of faces within a cluster relative to the eye is unchanged, no matter where the eye is, and only the order of clusters themselves must be determined in real time (Schachter 1983). This algorithm is sometimes called the priority list approach. For this algorithm, during data-base development, entire groups of faces or clusters are identified for which the occulted order within a group is totally independent of eye position. This priority order is stored in the data-base for each cluster along with the planes separating the clusters.

Priority Processor And Resolver

Generating images in real time requires an efficient algorithm, in addition to special-purpose hardware to implement it. The earliest equipment of this sort was built by General Electric for NASA's Manned Spacecraft Center in 1968 (Newman and Sproull 1979). The hardware was based on a priority algorithm that concurrently scan converted a large number of polygons, selecting the one with the highest priority for display. This technique is still used today.

A priority processor constructs a list containing a unique priority number for each face, based upon viewpoint, active data lists, separation planes and relative priority data obtained from the data base (Schachter 1983). The stages of operation are as follows. Viewpoint and moving model positions are received at the start of each raster

period. Using these data, the priority processor establishes the relative priorities of the coordinate systems, producing a top-level priority list containing a single entry for each coordinate system. Then, for each entry in the top-level priority list, the priority processor constructs a priority list with a single entry for each model of each coordinate system. (A model is defined by General Electric as any collection of object faces for which relative priorities have been determined and stored during off-line during data-base development.) Next, the priority processor creates a separate priority list for each active model. These lists have a single entry for each object of a model. In the final stage, the individual object priority lists are combined in model priority list order to obtain a complete active-object priority list. The priority numbers of ground plane faces are then combined with the priority numbers of object faces to form the final list. (Objects always have priority over the ground surface.)

Final hidden-surface calculations are performed by a priority resolver which receives the priority list and an ordered edge list. Priority resolution is essentially a convex-object-oriented function. When the left element of an object is received for a channel, the object identification is entered in a list. When the right edge of an object is received, the object identification is removed. Thus, at any moment, the priority list contains the identities of all objects pierced by a ray from the viewpoint, through the raster element being computed, into the 3-D environment. The priority resolver uses the available priority information to determine which of these pierced faces is closest to the viewpoint (Schachter 1983).

LISTABILITY

General Electric visual systems are based almost entirely on the use of separation planes to resolve priority and form an ordered list of faces in priority order. When a set of planes can correctly yield this priority for any viewpoint in the data base, the set is called a "listable set." There is an anomaly, however, that occurs quite frequently which is referred to as a "non-listable set." This means that for some viewpoint the data provided with the separation planes is not sufficient to form the priority list. In these cases, operator intervention is required, which can be quite costly in time and money. Unfortunately, this is usually the norm rather than the exception.

A new algorithm was required that generated the required planes; one that would always form a listable set. This is what was developed, but before proceeding directly into the new algorithm an overview of the situation will be presented. In this presentation an example of a non-listable set will also be given.

For purposes of explanation, it is helpful to first define a coordinate system as shown in Figure 5. Figure 5 shows a viewpoint origin, an environment origin and a model origin. The observer's line of sight, or boresight, is along the u axis. Models refer to a single object or object cluster.

The coordinate system at the observer's eye will be referred to as the viewpoint coordinate system. It is a left-handed system. The

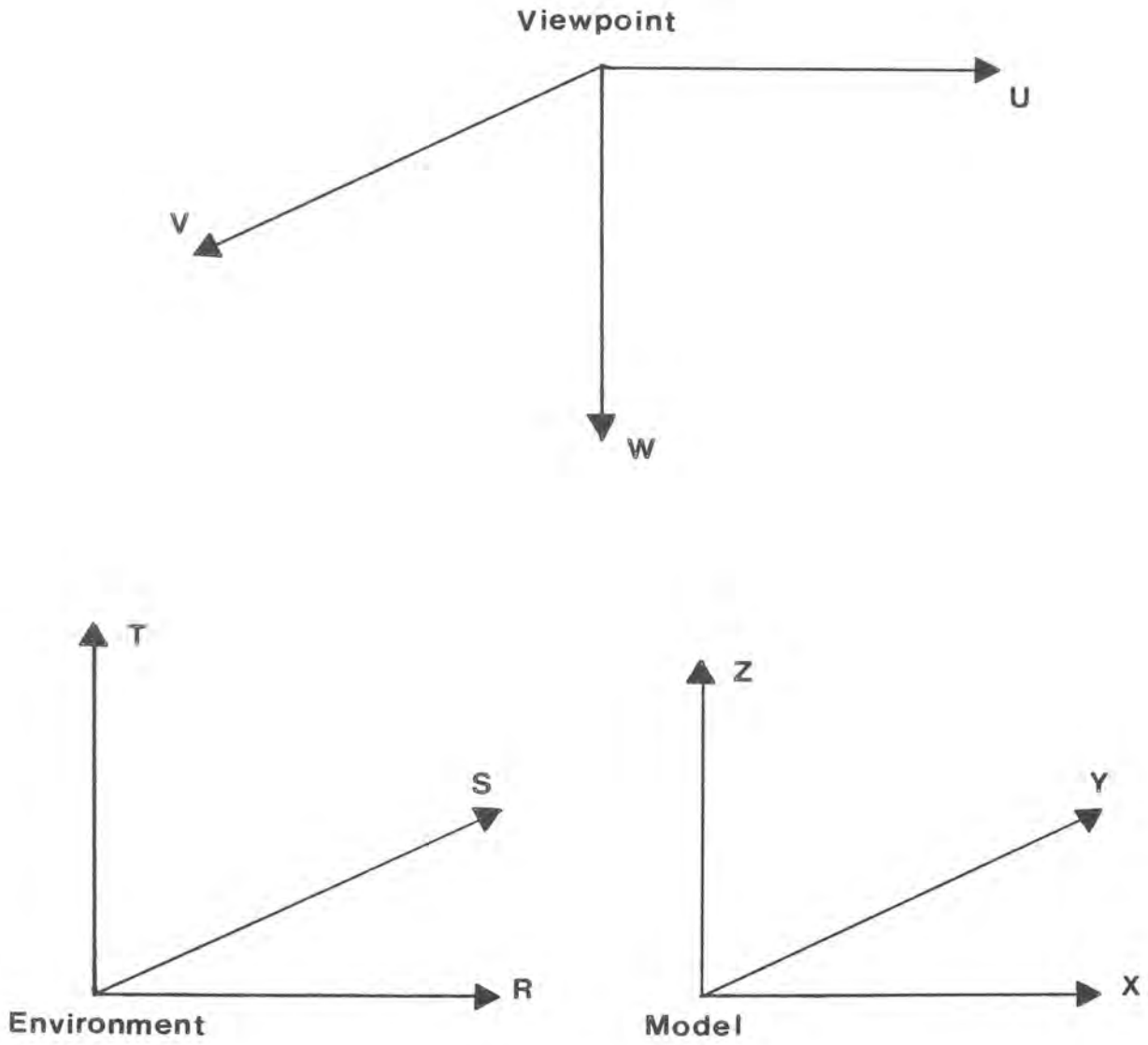

Figure 5. Coordinate System.

coordinate system in which the scene is modeled will be referred to as the environment coordinate system. A transformation matrix can be used to transform all points defined in the environment coordinate system to points defined in the viewpoint coordinate system (Newman and Sproull 1979). As the observer and correspondingly the viewpoint coordinate system change position relative to the environment coordinate system, the transformation matrix is calculated and used to transfer all points (and additional vector information) from the environment coordinate system into the viewpoint coordinate system.

The third coordinate system used is the model coordinate system. The model coordinate system is a right-handed system and is parallel to the environment coordinate system. This coordinate system is what the planes of the objects are based on. This is so the models can be placed at different locations in the environment system without a recalculation of the object planes. A simple translation matrix can be used to make the planes relative to the environmental origin. Since any point in the model coordinate system can be quickly converted to the viewpoint coordinate system, the separation planes are described along with objects in the model coordinate system. Therefore, this paper will be concerned only with the model coordinate system.

Instead of using x,y,z locations to describe the separation planes, the CIG hardware uses the unit normal of the plane and the perpendicular distance, called directed distance, along the normal to the origin. Included with each plane is a list of objects that are resolved by the plane. In order to label the sides of the separation plane, the normal is directed towards the "TRUE" side, the other being labeled the "FALSE"

side. So for the trivial case shown in Figure 6, the condition resolved is

$$\overline{N}_s = 1,0,0, \tag{1}$$

$$d_s = +4.0 \tag{2}$$

and

$$\text{TRUE/FALSE} = A/B, \tag{3}$$

where N_s is the unit normal of the plane surface and d_s is the directed distance.

In order to calculate the separation plane unit normal, it is necessary to refer to vector algebra. First, note that the vector cross-product of two vectors yields a vector perpendicular to the two vectors. The direction of the resulting vector is obtained by using the right-hand or left-hand rule. In this case the right hand rule applies since the model coordinate system is a right-handed system.

Three consecutive edges of an element define two vectors if the two edge node coordinates are subtracted from one another. The geometric interpretation is that both vectors define a plane which is parallel to the separation plane and passing through the origin. Therefore any vector perpendicular to this plane is also perpendicular to the separation plane, which will be the separation planes normal.

Referring to Figure 6, the coordinates of nodes 1, 2 and 3 are (x_1, y_1, z_1), (x_2, y_2, z_2) and (x_3, y_3, z_3), respectively. The two vectors parallel to the plane will be,

$$\overline{a} = \text{node 1 - node 2}, \tag{4}$$

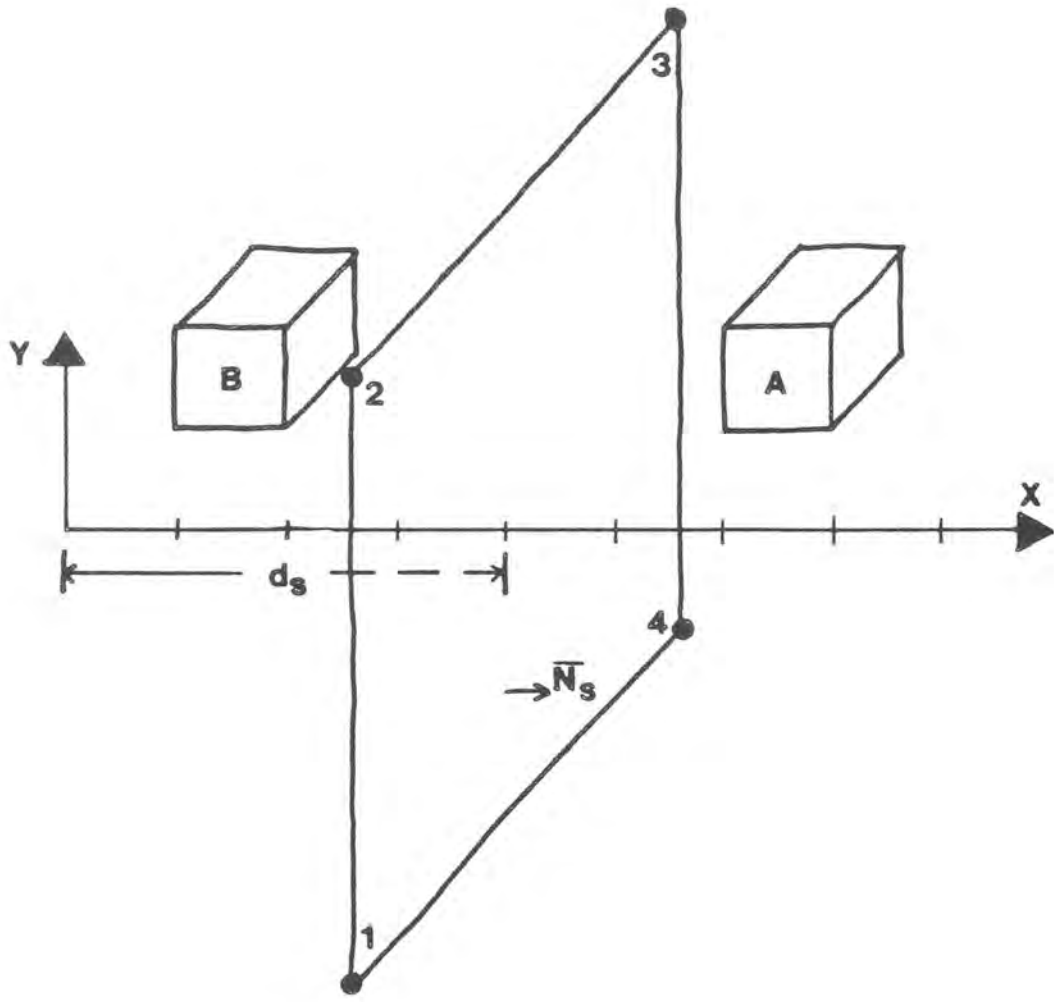

Figure 6. Two Object Model.

$$= (x_1-x_2,\ y_1-y_2,\ z_1-z_2), \tag{5}$$

$$= (ax,\ ay,\ az), \tag{6}$$

and

$$\overline{b} = \text{node 3 - node 2}, \tag{7}$$

$$= x_3-x_2,\ y_3-y_2,\ z_3-z_2, \tag{8}$$

$$= (bx,\ by,\ bz). \tag{9}$$

Now the two vector cross-product to obtain the normal will be,

$$\overline{N} = \overline{a} \times \overline{b} = (Xn,\ Yn,\ Zn), \tag{10}$$

where

$$Xn = ay * bz - az * by\ , \tag{11}$$

$$Yn = az * bx - ax * bz\ , \tag{12}$$

and

$$Zn = ax * by - ay * bx\ . \tag{13}$$

The unit normal is this normal divided by its magnitude, where the magnitude of a vector is defined as the length of the vector, expressed mathematically as,

$$/\overline{N}/ = \text{SQRT}\ (Xn^2 + Yn^2 + Zn^2). \tag{14}$$

The unit normal of a separation plane is therefore

$$\overline{N}_s = \frac{\overline{N}}{/\ \overline{N}\ /}. \tag{15}$$

During visual generation the on-line priority processor must determine the relationship of the viewpoint to each of the separation planes. Clearly, if the viewpoint is on the "TRUE" side of the plane, then object A has priority over B in Figure 6. Suppose the viewpoint is located at

$$\overline{R}_p = (7,\ 1,\ 10),$$

then the directed distance to the viewpoint is

$$\overline{R}_p \cdot \overline{N}_s = +7 \quad .$$

The viewpoint is further from the origin than the plane, along N_s, so it is on the "TRUE" side. This is expressed with the mathematical equations,

$$\text{if } \overline{R}_p \cdot \overline{N}_s - d_s \geq 0 \text{ then TRUE}$$

else

$$\text{if } \overline{R}_p \cdot \overline{N}_s - d_s < 0 \text{ then FALSE.}$$

The "TRUE/FALSE" indicates whether the "TRUE" objects or "FALSE" objects have priority. For the case shown, object A has priority over object B.

For real models, there are considerably more than two objects to resolve. The separation planes are selected such that each pair of objects is resolvable. For N objects there will be

$$\frac{N(N-1)}{2}$$

object pairs.

For example, suppose there is a four object model with objects (A, B, C, D) then associated with the separation plane data there must be six object pairs in the TRUE/FALSE data. They are as follows,

A / B
A / C
A / D
B / C
B / D
C / D.

The priority processor uses this information as follows:

(a) For each separation, determine the viewpoint "TRUE" or "FALSE" location;

(b) A counter is maintained for each object which is incremented by one if the object is on the

opposite side of the separation plane than the viewpoint. Only the objects that are listed in the TRUE/FALSE data for this separation plane are incremented. Since every object pair is used, the counter reflects the number of objects with priority over this object;

(c) The output priority list is obtained from the counters. The highest priority object has no objects with priority over it so it would have a count of zero;

(d) One constraint which relates to listability is that each object has a unique count.

For four objects the counters would read (0, 1, 2, 3) for a total of six, the number of object pairs. A count of (1, 1, 1, 3) is not allowed, even if the objects are not in priority conflict. This is a non-listable set and would result in an incorrect priority order for some viewpoints.

Non-Listable Set

Although the case shown is not typical of the configurations found in normal data bases, it does demonstrate the non-listable set. Figure 7 shows a four object set. The object pairs for the separation planes are as follows:

PLANE	TRUE	FALSE
1	C	B
	D	A
2	B	D
	C	A
3	B	A
	D	C .

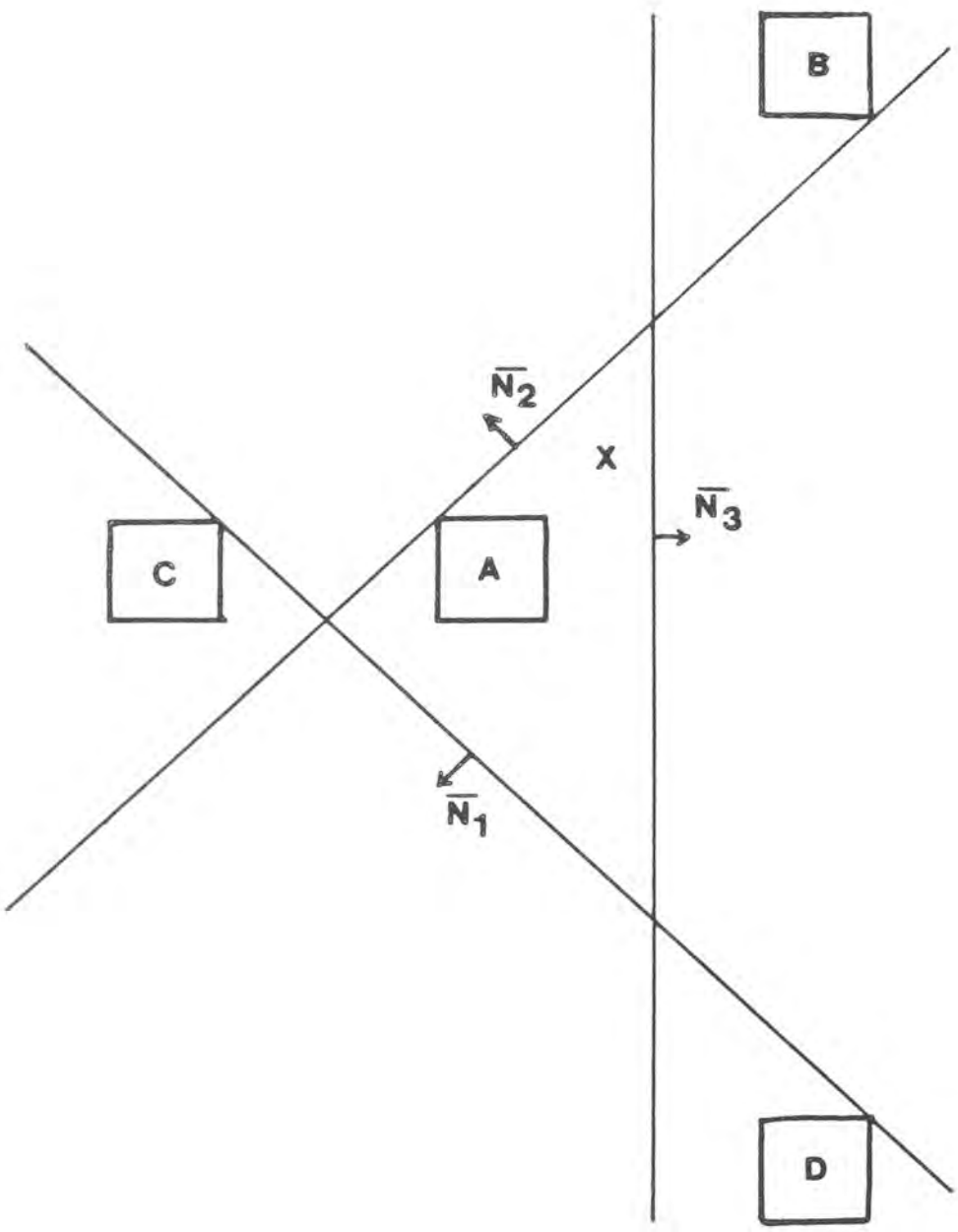

Figure 7. Four Object Model.

The viewpont location is shown with a cross and is on the "FALSE" side of all three separation planes. Using the counters, the first plane yields the counts

OBJECT	COUNT
A	0
B	0
C	1
D	1 .

The second separation plane adds to the counters to yield

Object	Count
A	0
B	1
C	2
D	1 .

The final plane yields the result.

Object	Count
A	0
B	2
C	2
D	2 .

Since each object count is not unique, this is a non-listable set. Close examination of the data will show how the problem occurred. Plane 1 yields B over C. Plane 3 yields C over D. This implies B over D, but Plane 2 yields D over B. In listability this condition is called a "circuit."

Notice, using Figure 7, the same separation planes but different TRUE/FALSE assignments yield the listable set

PLANE	TRUE	FALSE
1	C	A
	C	B
2	D	A
	D	B
3	B	A
	D	C .

Given the object pairs for the same viewpoint, the counts would be

Object	Count
A	0
B	1
C	2
D	3 .

This is a listable set. The output priority list would be A over B over C over D.

There are several off-line programs that calculate the required separation planes. The programs are very complex and unfortunately will not always find a listable set, depending on whether the objects contain circuits. If all subsets are free of circuits then the entire set is listable.

Binary Search Algorithm

A listability algorithm was needed which would generate a model's TRUE/FALSE list that would be valid from all viewpoints. The algorithm developed uses a binary search form of logic. A listing of the program is given in the Appendix. The following is a description of the algorithm.

First, a list of candidate separation planes is required. This list is obtained by using, as an initial guess, the faces of the objects to be ordered. In most cases this is a sufficient set but there are occurrences where no listable set exists in this data. Therefore, a provision has been added where the operator can also add possible candidate separation planes.

Each object is tagged with plus for a "TRUE" side object and minus for a "FALSE" side object. For the example shown in Figure 7 the data would be (using letters instead of numbers for objects):

Plane	Objects
1	-A -B +C +D
2	-A +B +C -D
3	-A +B -C +D.

From this form the TRUE/FALSE pairs are easily constructed. There are a large number of planes and TRUE/FALSE condidates available at this point and in most cases a sizable number of usable correct listable sets. Any listable set will work, so the purpose is to identify one; preferably the one with the least number of separation planes.

The algorithm attempts to subdivide the set down into successive disjoint sets. The first step, given "N" objects, is to identify those planes containing all "N" objects, be they "TRUE" or "FALSE" or a mixture. These planes, and thus far there have been several, are the candidates for the first plane to be used.

The criteria for selecting one is to form a count of the number of plus objects (N_p) and a count of the number of minus objects (N_m). Given that no separation plane divides an object, it will always hold that

$$N = N_p + N_m.$$

Therefore, for each plane candidate compute the relative difference

$$N_c = /N_p - N_m/.$$

The selected plane is the one with the minimum N_c. This resolves the problem into two sets, one which has the "TRUE" objects from above and

the other the "FALSE." For each subset, again find the separation planes containing those objects and test for the best plane. This continues until every object occurs in a subset by itself; this is then a listable set.

Using the case shown in Figure 7 as an example, the TRUE/FALSE tagging is:

Plane	Objects
1	-A -B +C +D
2	-A +B +C -D
3	-A +B -C +D.

For plane 1 the counts would be

$$N_p = 2,\ N_m = 2, \text{ and } N_c = 0.$$

For plane 2 the counts would be

$$N_p = 2,\ N_m = 2, \text{ and } N_c = 0.$$

Finally, for plane 3 the counts would again be

$$N_p = 2,\ N_m = 2 \text{ and } N_c = 0.$$

Since all N_c are zero, each plane is equally as good, and the first will be chosen. This plane yields the TRUE/FALSE pairs:

C/A, C/B, D/A, and D/B.

This plane also forms the two sets;

set 1 = Object A, B

and

set 2 = Object C, D.

Testing the planes yields that both Plane 2 and 3 qualify. After using plane 1 it is deleted from further use.

For plane 2, set 1 yields

$$N_p = 1\ N_m = 1 \text{ and } N_c = 0.$$

For plane 3, set 1 yields

$$N_p = 1,\ N_m = 1 \text{ and } N_c = 0.$$

Therefore, use plane 2 with the TRUE/FALSE pair B/A.

The same test using set number 2 yields plane 2 again with the TRUE/FALSE pair C/D. The final TRUE/FALSE pairs are:

plane 1 C/A, C/B, D/A, D/B

and

plane 2 B/A, C/D.

This is listable and all four objects, in a subset by themselves, are listable.

Limitations

Up to now no formal mention has been made about what happens if the separation planes intersect one of the objects of the model. The quick answer would be that this would generate a non-listable set. This is not always the result.

In order for the algorithm to work there must always be at least one plane that will separate the initial set into two complete groups. Otherwise, proper binary determination cannot be made. After this first initial cut, this requirement then moves to the subsets. For each subset there again must be a separation plane that will divide the subset into two complete groups. If at the same time this plane intersects one of the objects of another subset, no harm is done. The only area of interest is the subset currently being worked on.

In order to handle the situation of a plane intersecting one of the objects, a zero is added to the list of possible outcomes. Each object is thus tagged plus for a "TRUE" side object, minus for a "FALSE" side object and zero for an intersected object.

BINARY SEARCH PROGRAM

A program was written to test the effectiveness of the binary search algorithm developed. Many test cases were investigated, but only three will be presented here.

Each test case consists of a model made up of varying amounts and arrangements of cube objects like the one shown in Figure 8. The vertices of the object were ordered so that all face normals would point away from the faces of the objects.

Test Case One

The first case presented is a model consisting of only two objects. The orientation of the objects is shown in Figure 9. For this case, a solution of a single separation plane should be obtained. Since the algorithm uses the object faces to generate the candidate separation planes, only two possible solutions exist. One is a separation plane at x=10 and the other is a separation plane at x=20. These are the only two separation planes that will divide the model into two subsets, in which case, each object will be in its own subset. Since each of these two planes are equally as good, the program uses the first one found. The output of the program is shown in Figure 10. The unit normal, directed distance, and TRUE/FALSE pair for the separation plane are given. This information would generate the plane shown in Figure 11. This separation plane divides the area that the viewpoint can be in into two regions; one on the true side of the plane, and one on the false

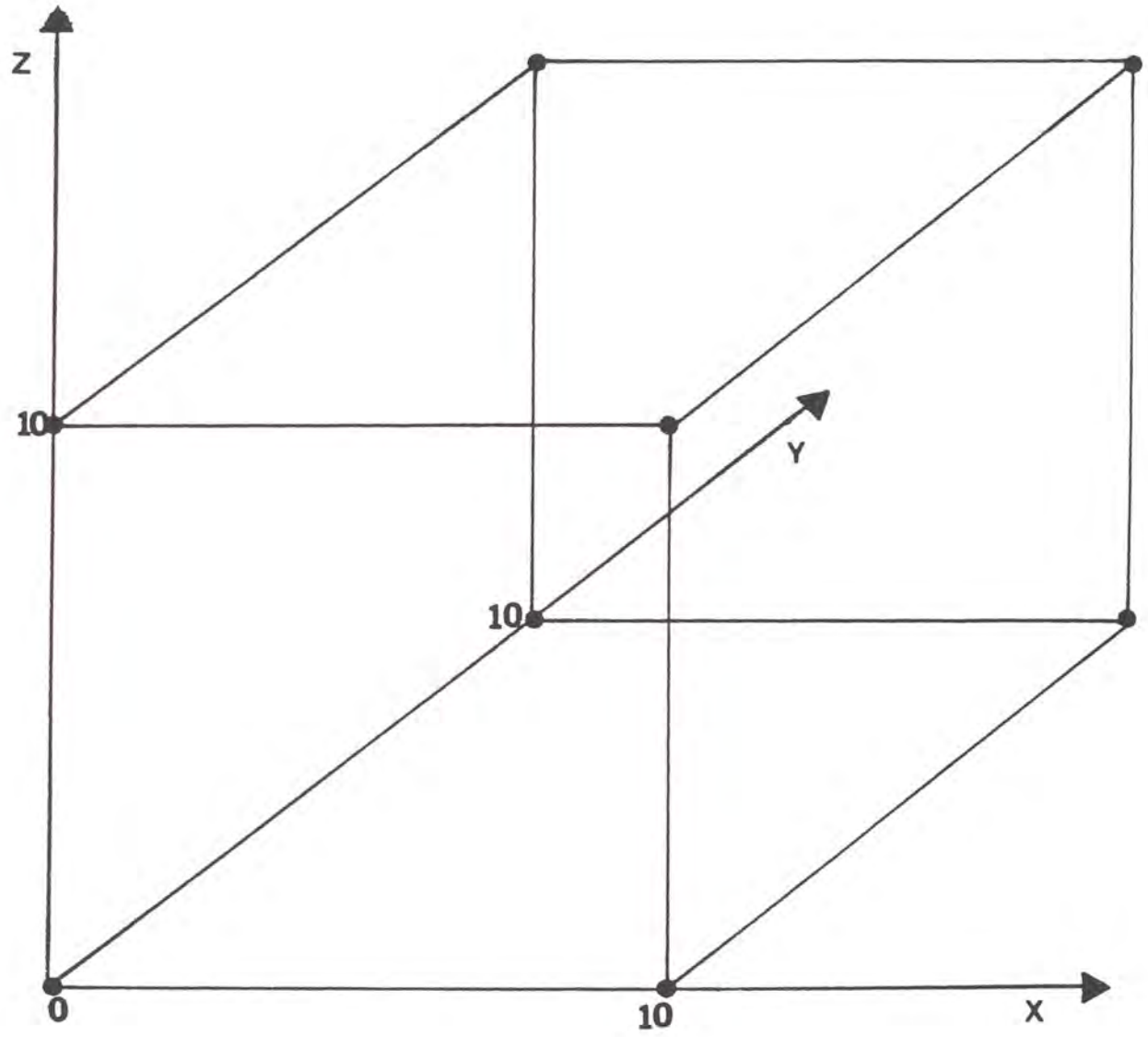

Figure 8. Single Cube Object.

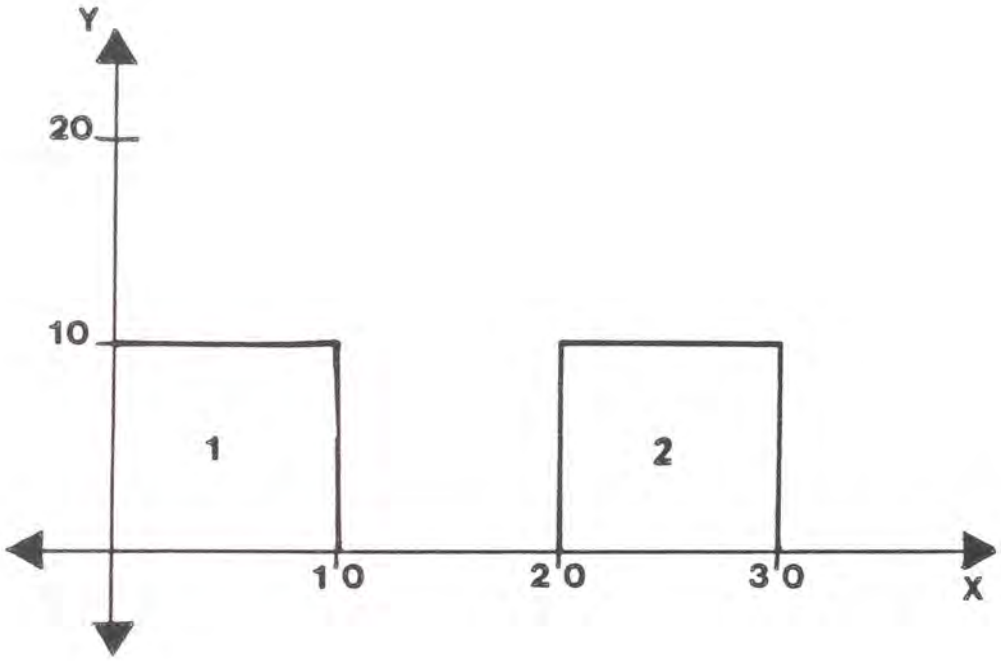

Figure 9. Two Object Model Test Case.

```
NUMER OF SEPARATION PLANES REQUIRED =  1

FOR PLANE NUMBER  1
THE UNIT NORMAL IS:  1.0  0.0  0.0
AND THE DIRECTED DISTANCE IS: 10.0
THE TRUE/FALSE PAIRS FOR THIS PLANE ARE:     TRUE      FALSE
                                            ------    -------
                                              2         1
```

Figure 10. Program Output for Two Object Model Test Case.

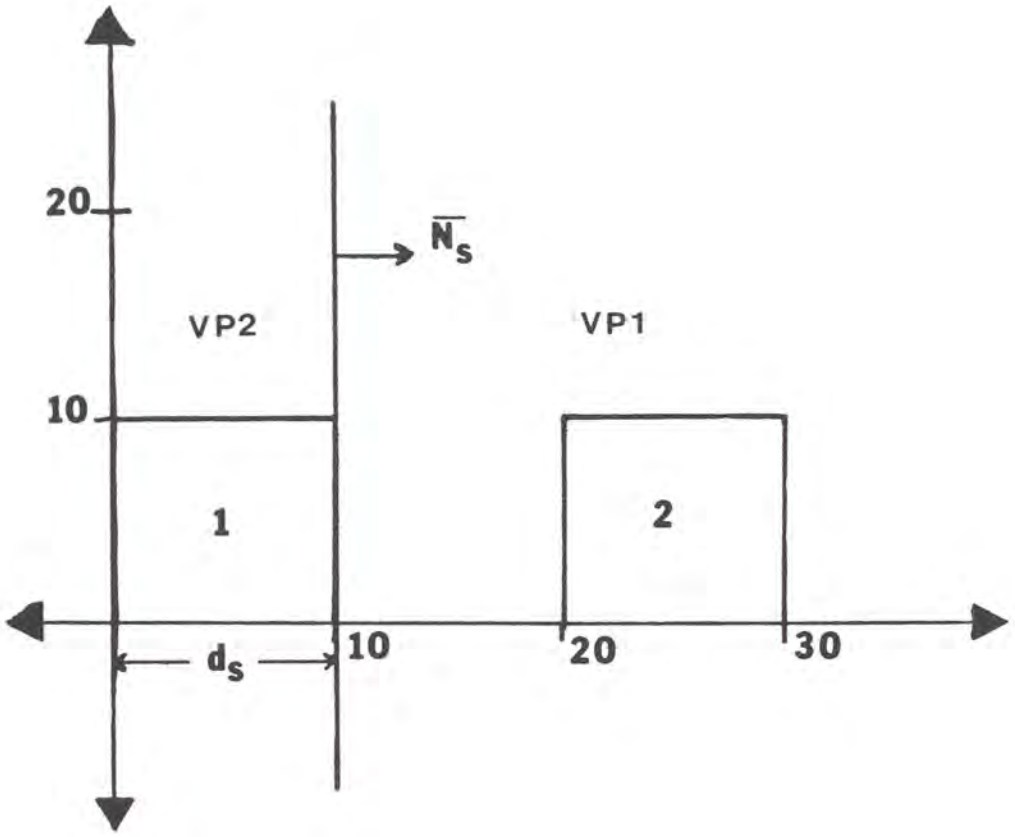

Figure 11. Two Object Model with Separation Plane.

side. These are labeled VP1 and VP2, respectively, in Figure 11. For VP1 the object counts would be (1, 0). For VP2 the object counts would be (0, 1). Since each object has its own unique count for all possible viewpoints, a listable set has been obtained.

It may have been noted that no mention has been made as to what occurs when the viewpoint is along one of the separation planes. This is because for these instances, no priority problem between the objects can exist, and the viewpoint is assumed to be on the true side of the plane.

Test Case Two

The second test case consists of a model containing four objects as shown in Figure 12. This is the test case of Figure 7. In addition to the candidate planes of the object faces, three additional candidate planes have been added. These planes are shown as the dashed lines in Figure 12.

The output of the program for this test case is shown in Figure 13. Only two separation planes are required to yield a listable set. The two planes required are shown in Figure 14. These two planes divide the possible areas of viewpoint location into four regions. These regions are labeled VP1, VP2, VP3 and VP4 in Figure 14. The counts for these viewpoints are:

VIEWPOINT REGION	OBJECT 1	2	3	4
VP1	2	3	0	1
VP2	3	2	1	0
VP3	1	0	3	2
VP4	0	1	2	3.

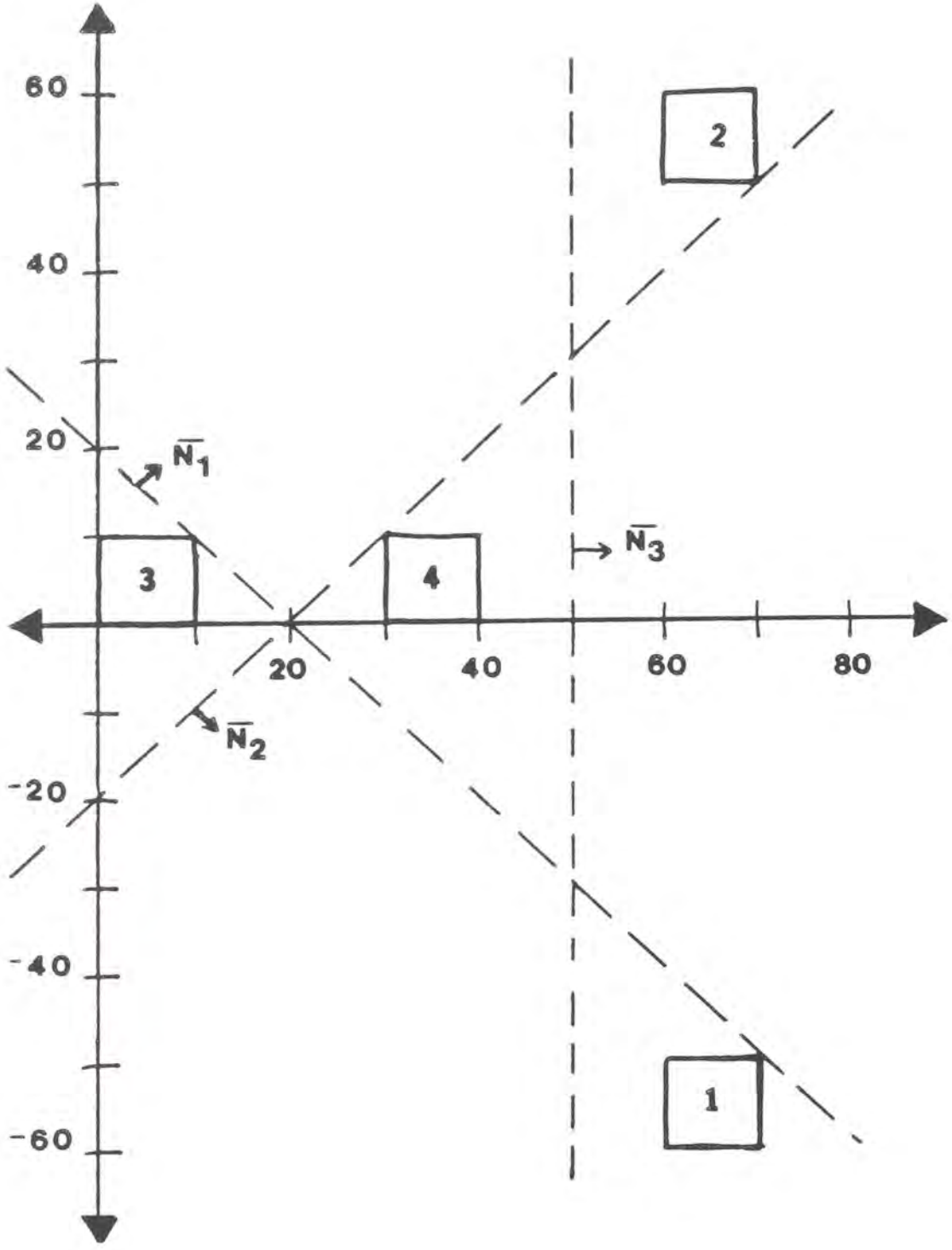

Figure 12. Four Object Model Test Case.

```
NUMER OF SEPARATION PLANES REQUIRED =  2

FOR PLANE NUMBER  1
THE UNIT NORMAL IS: -1.0  0.0  0.0
AND THE DIRECTED DISTANCE IS:-60.0
THE TRUE/FALSE PAIRS FOR THIS PLANE ARE:
```

TRUE	FALSE
3	1
3	2
4	1
4	2

```
FOR PLANE NUMBER  2
THE UNIT NORMAL IS:  0.7  0.7  0.0
AND THE DIRECTED DISTANCE IS: 14.1
THE TRUE/FALSE PAIRS FOR THIS PLANE ARE:
```

TRUE	FALSE
4	3
2	1

Figure 13. Program Output for Four Object Model Test Case.

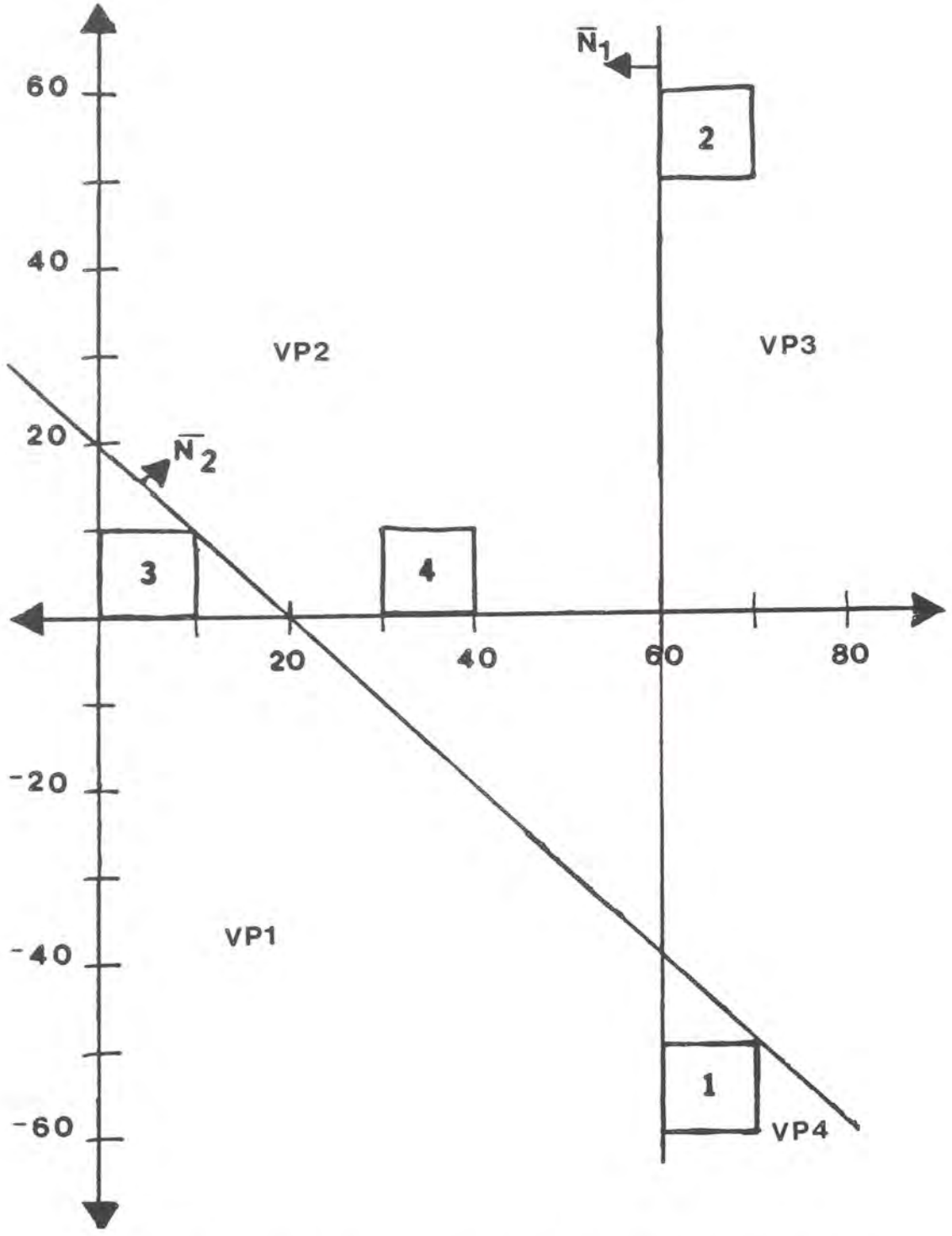

Figure 14. Four Object Model with Separation Planes.

Since each object has a unique count for each possibility, a listable set has been obtained.

The two planes were obtained as follows. For the first separation there exist five (5) possible candidates out of the twenty-seven (27) given. Since each candidate is as good as another, the first plane found is used. The next step is to divide each of the remaining subsets until each object of the model is in its own subset. The first subset dealt with contains objects 1 and 2. There are six possible ways to divide these two objects. These planes are shown in Figure 15. Here the selection of the correct plane is crucial in order to minimize the final number of required planes. A wrong choice here and three planes may be required as opposed to two.

The decision as to which plane to use is made on information obtained earlier when all the objects were compared. The plane that divides the most objects will be used as the subset separation plane. There exist two planes that are equal in this regard, plane 5 and plane 6, so the first one is arbitrarily chosen.

The next step of the algorithm is to divide the last subset, objects 3 and 4. Note, however, that these two objects have already been divided by the selection of the second plane. Therefore, only two planes are required.

Test Case Three

The final test case consists of separation planes that intersect some of the objects. For this case a model containing five objects is used. The orientation of the objects is shown in Figure 16.

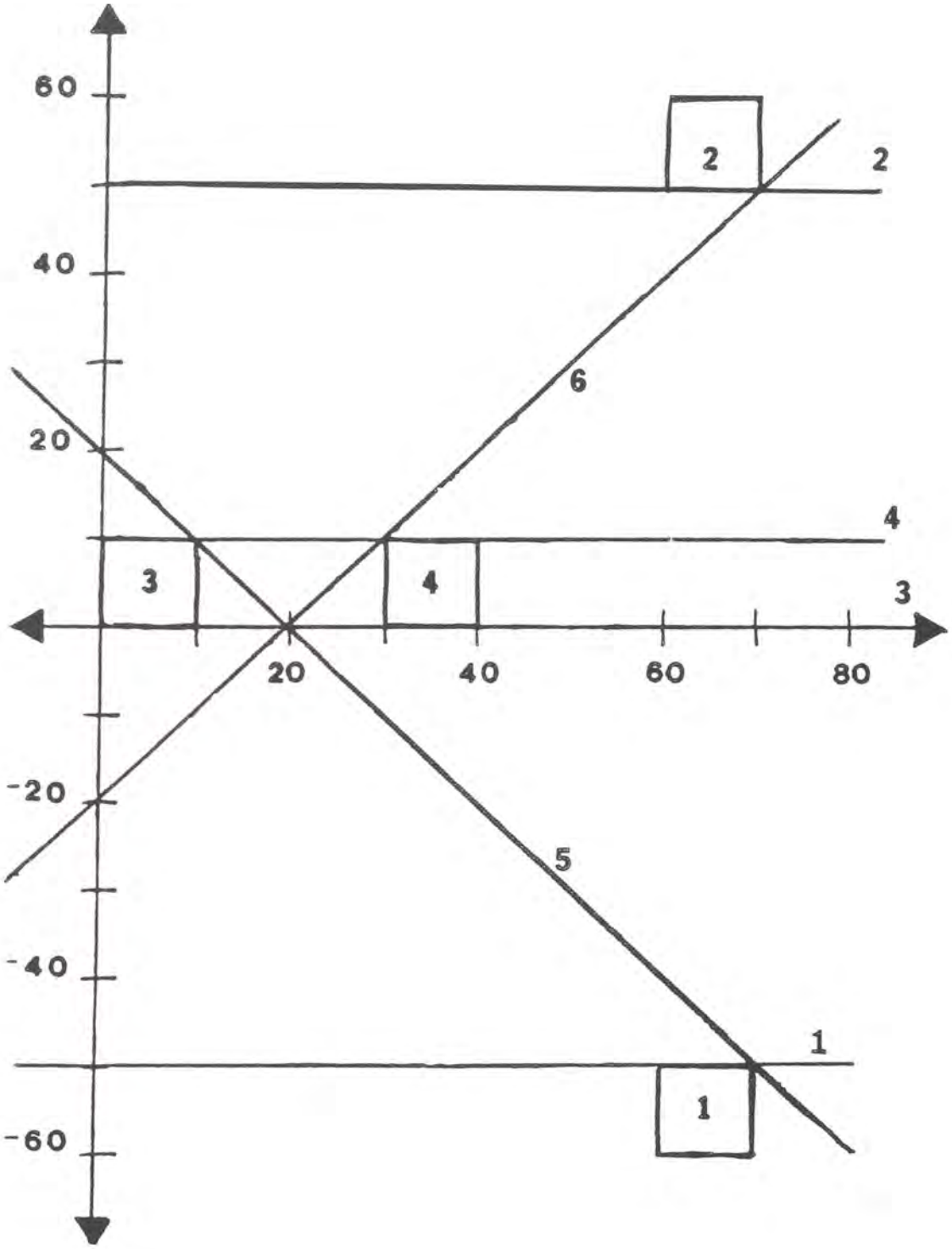

Figure 15. First Subset Candidate Separation Planes.

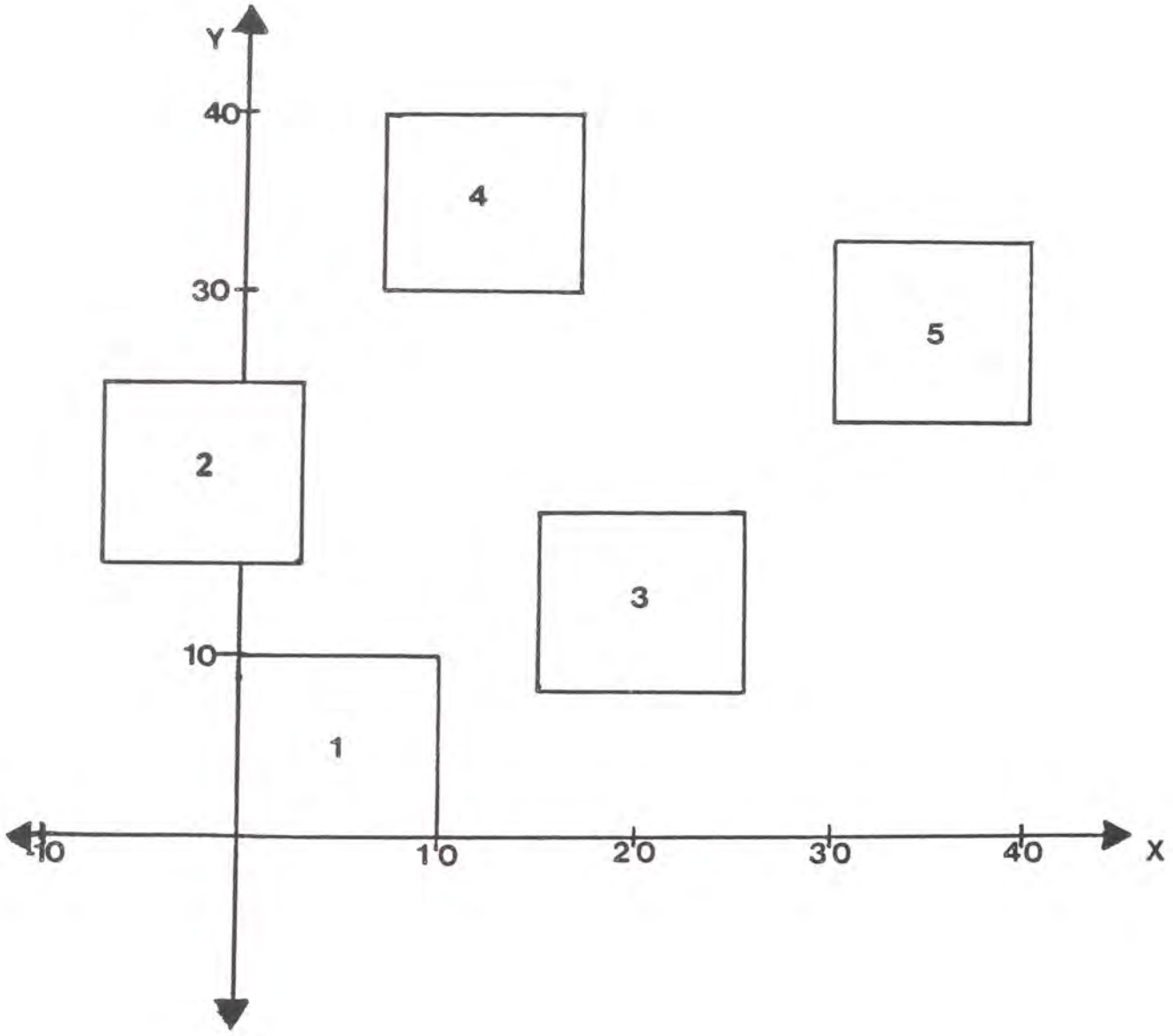

Figure 16. Five Object Model Test Case.

The output of the program for this test case is shown in Figure 17. This model requires four separation planes. The required planes are shown in Figure 18. These four planes divide the viewpoint area into nine regions. The counts for these regions are:

VIEWPOINT REGION	OBJECT 1	2	3	4	5
VP1	0	1	2	3	4
VP2	1	0	2	3	4
VP3	2	1	3	0	4
VP4	3	2	1	0	4
VP5	4	3	2	1	0
VP6	3	2	1	4	0
VP7	2	3	1	4	0
VP8	1	2	0	3	4
VP9	2	1	0	3	4.

As can be seen this is a listable set.

```
NUMER OF SEPARATION PLANES REQUIRED =  4

FOR PLANE NUMBER  1
THE UNIT NORMAL IS:  1.0  0.0  0.0
AND THE DIRECTED DISTANCE IS: 25.0
THE TRUE/FALSE PAIRS FOR THIS PLANE ARE:     TRUE      FALSE
                                            ------    -------
                                              5         1
                                              5         2
                                              5         3
                                              5         4
FOR PLANE NUMBER  2
THE UNIT NORMAL IS:  0.0  1.0  0.0
AND THE DIRECTED DISTANCE IS: 25.0
THE TRUE/FALSE PAIRS FOR THIS PLANE ARE:     TRUE      FALSE
                                            ------    -------
                                              4         1
                                              4         2
                                              4         3
FOR PLANE NUMBER  3
THE UNIT NORMAL IS:  1.0  0.0  0.0
AND THE DIRECTED DISTANCE IS: 10.0
THE TRUE/FALSE PAIRS FOR THIS PLANE ARE:     TRUE      FALSE
                                            ------    -------
                                              3         1
                                              3         2
FOR PLANE NUMBER  4
THE UNIT NORMAL IS:  0.0  1.0  0.0
AND THE DIRECTED DISTANCE IS: 10.0
THE TRUE/FALSE PAIRS FOR THIS PLANE ARE:     TRUE      FALSE
                                            ------    -------
                                              2         1
```

Figure 17. Program Output for Five Object Model Test Case.

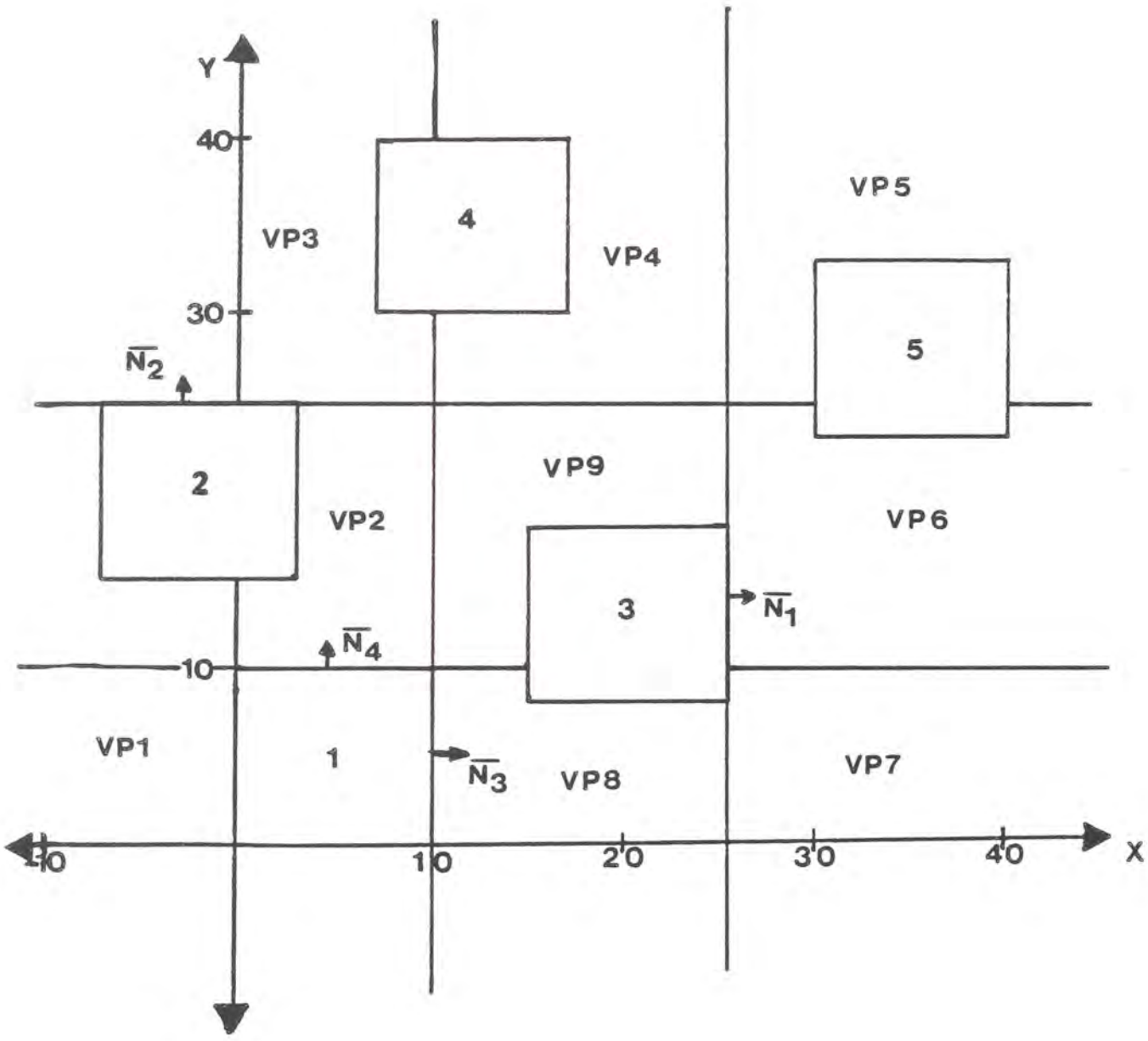

Figure 18. Five Object Model with Separation Planes.

CONCLUSION

Many different types of algorithms are available that determine the proper priority of objects that occult one another. Many of these algorithms, however, require large amounts of processing time or additional user interaction. Some algorithms generate large amounts of data that must be stored to insure proper operation. The final algorithm used must be computationally efficient for real time requirements needed in flight simulation. It must be a compromise between fidelity and computing cost (Steiner 1985).

The method described in this report uses separation planes to determine priority. Separating planes can be determined manually using the geometry of the model. This technique has the downfall of requiring a large amount of information stored in the CIG hardware.

The algorithm developed evaluates the minimum number of separating planes required to yield the correct priority. The computer program written to exercise this algorithm and various examples have been given in order to demonstrate its capabilities. This program is in use at General Electric Simulation and Control Systems Department, and as of the date of this report has yet to generate an incorrect set of planes.

APPENDIX

```
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C
C                ROUTINE NAME: SEPPLN
C
C                      AUTHOR: KEITH A. STUMP
C                              GENERAL ELECTRIC (SCSD)
C                              DAYTONA BEACH, FL
C
C                DATE CREATED: 12/4/85
C
C>----------------------------------------------------------------------
C
C                 ENTRY POINT: SEPPLN
C
C            CALLING ROUTINES: NONE
C
C             CALLED ROUTINES: LOAD(OBJCNT,OSEPCNT,SYSERR)
C                              TRUE_FALSE(OBJCNT,IPLNCO,EDGPNT,ICODE)
C
C>----------------------------------------------------------------------
C
C     FUNCTIONAL DESCRIPTION:
C
C          This routine is the main code for the separation plane
C     function. It is responsible for gathering all the potential
C     separation planes and loading the true/false data table with
C     the plane information. All faces are assumed to be potential
C     separation planes along with any planes supplied by the user.
C     Errors are issued if any two objects are unable to be
C     separated from each other. A listable set of separation planes
C     is determined and the required plane information is displayed
C     to the user.
C
C>----------------------------------------------------------------------
C
C       VARIABLE DEFINITIONS:
C
C      NAME        TYPE               DESCRIPTION                 USE
C     --------   ---------   ------------------------------   ----------
C     CNDPNT      INTEGER    CANDIDATE PLANE POINTER            LOCAL
C     CURNC       INTEGER    CURRENT PLUS/MINUS DIFFERENCE      LOCAL
C     FOUND       LOGICAL    PLANE LIST FOUND FLAG              LOCAL
C     I           INTEGER    INDEX VARIABLE                     LOCAL
C     ICAND       LOGICAL    CANDIDATE PLANE FLAG               LOCAL
C     IEDGCNT     INTEGER    FACE EDGE COUNT                    LOCAL
C     IEDGPNT     INTEGER    FACE EDGE POINTER                  LOCAL
C     IFAC        INTEGER    FACE INDEX                         LOCAL
C     IFACE       INTEGER    FACE EDGE COUNT ARRAY              COMMON
C     IFACCNT     INTEGER    OBJECT FACE COUNT                  LOCAL
C     IFACPNT     INTEGER    OBJECT FACE POINTER                LOCAL
C     IFALSE      INTEGER    PLANE FALSE ARRAY                  LOCAL
C     IFLCNT      INTEGER    FALSE ELEMENT COUNT                LOCAL
C     IFLPNT      INTEGER    FALSE ELEMENT POINTER              LOCAL
C     INEED       INTEGER    NECESSARY PLANE ARRAY              LOCAL
C     IOBJ        INTEGER    OBJECT INDEX COUNT                 LOCAL
C     IOBJECT     INTEGER    OBJECT FACE ARRAY                  COMMON
C     IPLNCO      INTEGER    POTENTIAL SEP. PLANE COUNT         LOCAL
C     IPLNPT      INTEGER    CURRENT PLANE POINTER              LOCAL
C     ISEL        INTEGER    SELECTED PLANE POINTER             LOCAL
C     ISET        INTEGER    SET/SUBSET OBJECT ARRAY            LOCAL
C     ISPMAP      INTEGER    OBJECT PAIR SEPARATION MAP         LOCAL
C     ISUB        INTEGER    SET/SUBSET COUNT ARRAY             LOCAL
C     ITRCNT      INTEGER    TRUE ELEMENT COUNT                 LOCAL
```

```
C     ITRPNT     INTEGER     TRUE ELEMENT POINTER                LOCAL
C     ITRUE      INTEGER     PLANE TRUE ARRAY                    LOCAL
C     J          INTEGER     INDEX VARIABLE                      LOCAL
C     K          INTEGER     INDEX VARIABLE                      LOCAL
C     KOBJ       INTEGER     SUBSET OBJECT NUMBER                LOCAL
C     KSET       INTEGER     SUBSET POINTER                      LOCAL
C     KSTOP      INTEGER     SUBSET STOP POINTER                 LOCAL
C     KSUB       INTEGER     PRESENT SUBSET POINTER              LOCAL
C     LFALSE     INTEGER     FALSE ELEMENT LIST                  LOCAL
C     LTRUE      INTEGER     TRUE ELEMENT LIST                   LOCAL
C     MXOBJ      INTEGER     MAXIMUM NUMBER OF OBJECTS           PARAMETER
C     NC         INTEGER     SUBSET PLUS/MINUS DIFFERENCE        LOCAL
C     NCNEW      INTEGER     SUBSET NEW DIFFERENCE               LOCAL
C     NCOLD      INTEGER     SUBSET OLD DIFFERENCE               LOCAL
C     NEEDCO     INTEGER     NUMBER OF SEP. PLANES NEEDED        LOCAL
C     NELE       INTEGER     NUMBER OF ELEMENTS IN SUBSET        LOCAL
C     NM         INTEGER     NUMBER OF MINUS OBJECTS             LOCAL
C     NMALL      INTEGER     NUMBER OF MINUS ALL OBJECTS         LOCAL
C     NMSEL      INTEGER     SELECTED PLANE MINUS OBJECTS        LOCAL
C     NP         INTEGER     NUMBER OF PLUS OBJECTS              LOCAL
C     NPALL      INTEGER     NUMBER OF PLUS ALL OBJECTS          LOCAL
C     NPSEL      INTEGER     SELECTED PLANE PLUS OBJECTS         LOCAL
C     NSET       INTEGER     TOTAL SUBSET COUNT                  LOCAL
C     OBJCNT     INTEGER     OBJECT COUNT                        PASSED
C     OSEPCNT    INTEGER     SUPPLIED SEPARATION PLANE COUNT     PASSED
C     OVERTEX    INTEGER     SUPPLIED VERTEX NUMBER ARRAY        COMMON
C     SYSERR     LOGICAL     SYSTEM ERROR FLAG                   PASSED
C     TFCNT      INTEGER     TRUE/FALSE PLANE COUNT              LOCAL
C     TRUFLS     INTEGER     TRUE/FALSE DATA ARRAY               COMMON
C
C<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      PROGRAM SEPPLN
C
      IMPLICIT NONE
C
C    >-  OBJECT/MODEL INTERFACE COMMON
C
      INTEGER IEDGE,IFACE,IOBJECT,OVERTEX,TRUFLS
      REAL    DIST,NORMAL,XVERT,YVERT,ZVERT
      COMMON /MODEL/ IEDGE(200),IFACE(100),IOBJECT(10),OVERTEX(10),
     +               XVERT(100),YVERT(100),ZVERT(100),
     +               DIST(100),NORMAL(3,100),TRUFLS(10,100)
C
C    >-  SET LOCAL VARIABLE TYPES
C
      INTEGER CNDPNT,CURNC,I,IEDGCNT,IEDGPNT,IFAC,IFACCNT,IFACPNT
      INTEGER IFCNT,IFLCNT,IFLPNT,ITRCNT,ITRPNT,IOBJ,IPLNCO,IPLNPT
      INTEGER ISEL,J,K,KOBJ,KSET,KSTOP,KSUB,MXOBJ,NC,NCOLD,NCNEW
      INTEGER NEEDCO,NELE,NM,NP,NMALL,NPALL,NPSEL,NMSEL,NSET,OBJCNT
      INTEGER OSEPCNT,TFCNT,IFALSE(100,100),ITRUE(100,100),INEED(2,100)
      INTEGER ISET(100),ISUB(100),LFALSE(100),LTRUE(100)
      LOGICAL FOUND,ICAND,SYSERR
C
C    >-  OBTAIN OBJECT COUNT AND OPERATOR SUPPLIED
C    >-  SEPARATION PLANE COUNTS. MODULE LOAD ALSO LOADS
C    >-  THE OBJECT INFORMATION IN THE INTERFACE COMMON.
C
      CALL LOAD(OBJCNT,OSEPCNT,SYSERR)
C
C    >-  IF THERE IS MORE THAN ONE OBJECT AND NO ERROR HAS OCCURED
C
      IF (OBJCNT.GT.1.AND..NOT.SYSERR) THEN
```

```
C
C    >-  LOAD THE TRUE/FALSE TABLE WITH THE STATE OF EACH OBJECT WITH
C    >-  RESPECT TO A GIVEN PLANE.
C    >-
C    >-            1 = OBJECT ON TRUE SIDE OF PLANE
C    >-           -1 = OBJECT ON FALSE SIDE OF PLANE
C    >-            0 = PLANE INTERSECTS OBJECT
C    >-
C    >-  USE ALL FACES WHICH SEPARATE ANY TWO OR MORE OBJECTS AS
C    >-  POSSIBLE SEPARATION PLANES. ALSO USE ANY SEPARATION PLANES
C    >-  OBTAINED FROM OTHER OBJECTS OR MANUALLY INPUT BY THE USER
C    >-  AS POTENTIAL SEPARATION PLANES.
C    >-
C    >-  INITIALIZE POTENTIAL SEPARATION PLANE COUNT
C
          IPLNCO = 0
C
C    >-  INITIALIZE FACE AND EDGE POINTERS
C
          IFACPNT = 1
          IFACCNT = 0
          IEDGPNT = 1
          IEDGCNT = 0
C
C    >-  STEP THROUGH ALL FACES OF ALL THE OBJECTS
C
          DO IOBJ = 1,OBJCNT
C
C    >-  SET FACE COUNTS AND POINTERS FOR CURRENT OBJECT
C
            IFACPNT = IFACPNT + IFACCNT
            IFACCNT = IOBJECT(IOBJ)
C
C    >-  STEP THROUGH EACH FACE OF AN OBJECT
C
            DO IFAC = IFACPNT, IFACPNT + IFACCNT - 1
C
C    >-  INCREMENT POTENTIAL PLANE COUNTER
C
              IPLNCO = IPLNCO + 1
C
C    >-  SET EDGE COUNTS AND POINTERS FOR CURRENT FACE
C
              IEDGPNT = IEDGPNT + IEDGCNT
              IEDGCNT = IFACE(IFAC)
C
C    >-  CHECK TO MAKE SURE THE FACE HAS AT LEAST THREE
C    >-  VERTICES. IF NOT NOTIFY OPERATOR AND EXIT.
C
              IF (IEDGCNT.LT.3) THEN
                WRITE(6,100)
 100            FORMAT(1H ,'FACE FOUND WITH LESS THAN THREE',
     +                     ' VERTICES.',/,' CHECK INPUT DATA',/,
     +                     ' PROGRAM EXITING...')
                SYSERR = .TRUE.
              ELSE
C
C    >-  LOAD THE FACE NORMAL, DIRECTED DISTANCE AND TRUE/FALSE
C    >-  TABLE FOR THIS PLANE.
C
                CALL TRUE_FALSE(OBJCNT,IPLNCO,IEDGPNT,1)
              ENDIF
            ENDDO
```

```
            ENDDO
C
C
C     >-    CHECK FOR SYSTEM ERROR
C
            IF (.NOT.SYSERR) THEN
C
C     >-    NEXT USE ANY USER INPUT SEPARATION PLANES
C     >-    AS POTENTIAL PLANES.
C
              DO I = 1,OSEPCNT
C
C     >-    INCREMENT POTENTIAL PLANE COUNT
C
                IPLNCO = IPLNCO + 1
C
C     >-    GET LOCATION OF FIRST VERTEX
C
                IEDGPNT = OVERTEX(I)
C
C     >-    LOAD THE FACE NORMAL, DIRECTED DISTANCE AND TRUE/FALSE
C     >-    TABLE FOR THIS PLANE.
C
                CALL TRUE_FALSE(OBJCNT,IPLNCO,IEDGPNT,2)
              ENDDO
C
C    >-   IF NO ERROR HAVE OCCURRED, THEN DO THE LISTABILITY ROUTINE
C
              IF (.NOT.SYSERR) THEN
C
C     >-    INITIALIZE THE NUMBER OF NECESSARY PLANES
C
                NEEDCO = 0
C
C     >-    INITIALIZE THE SET COUNTS
C
                 DO I = 1,OBJCNT
                  ISET(I) = I
                ENDDO
C
C     >-    INITIALIZE TOTAL SUBSET COUNT, FIRST SUBSET COUNT
C     >-    AND PRESENT SUBSET POINTER.
C
                NSET = 1
                ISUB(1) = OBJCNT
                KSUB = 0
C
C     >-    START THE BINARY DIVISION LISTABILITY LOOP
C     >-    LOOP UNTIL ALL OBJECTS HAVE BEEN DIVIDED INTO
C     >-    A SUBSET BY THEMSELVES OR IN A LISTABLE SUBSET
C
              DO WHILE (KSUB.LT.NSET)
C
C     >-    INCREMENT THE PRESENT SUBSET POINTER
C
                 KSUB = KSUB + 1
C
C     >-    OBTAIN THE NUMBER OF ELEMENTS IN THE PRESENT SUBSET
C     >-    FROM THE SUBSET COUNT LIST
C
                 NELE = ISUB(KSUB)
C
C     >-    IF THE PRESENT SUBSET CONTAINS MORE THAN ONE ELEMENT
```

```
C    >-  IT MUST BE BROKEN DOWN INTO SMALLER GROUPS
C
            IF (NELE.GT.1) THEN
C
C    >-  INITIALIZE SELECTED PLANE POINTER, SUBSET SIZE
C    >-  DIFFERENCE AND CANDIDATE PLANE POINTER
C
                ISEL   = 0
                NC     = 999999
                CNDPNT = 0
C
C    >-  SEARCH THRU THE TRUE/FALSE DATA TABLE TO FIND
C    >-  A PLANE WHICH SEPARATES THE PRESENT SUBSET INTO
C    >-  THE TWO LARGEST GROUPS
C
                  DO WHILE (CNDPNT.LT.IPLNCO)
C
C    >-  INCREMENT THE CANDIDATE PLANE POINTER
C
                    CNDPNT = CNDPNT + 1
C
C    >-  INITIALIZE THE NUMBER OF PLUS OBJECTS,
C    >-  THE NUMBER OF MINUS OBJECTS AND CANDIDATE
C    >-  PLANE FLAG.
C
                    NP     = 0
                    NM     = 0
                    ICAND  = .TRUE.
C
C    >-  INITIALIZE SUBSET POINTER AND
C    >-  SUBSET STOP POINTER WITHIN ISET ARRAY
C
                    KSET  = KSUB - 1
                    KSTOP = KSUB + NELE - 1
C
C    >-  SEARCH THRU THE PLANE DATA IN THE TRUE/FALSE
C    >-  DATA TABLE AND DETERMINE THE NUMBER OF ELEMENTS
C    >-  ON THE TRUE SIDE AND FALSE SIDE. IF THE PLANE
C    >-  INTERSECTS AND ELEMENT OF THE SUBSET THEN REJECT
C    >-  THIS PLANE AS A CANDIDATE.
C
                    DO WHILE ((KSET.LT.KSTOP).AND.(ICAND))
C
C    >-  INCREMENT SUBSET POINTER
C
                      KSET = KSET + 1
C
C    >-  PULL OUT SUBSET OBJECT NUMBER FROM THE
C    >-  ORDERED SET OF OBJECT NUMBERS
C
                      KOBJ = ISET(KSET)
C
C    >-  IF THE OBJECT IS ON THE TRUE OR FALSE SIDE OF
C    >-  THE PLANE THEN INCREMENT THE PLUS/MINUS COUNTERS.
C    >-  IF THE PLANE INTERSECTS THE PARTITION THEN DISCARD
C    >-  IT AS A POTENTIAL PLANE.
C
                      IF (TRUFLS(KOBJ,CNDPNT).EQ.1) THEN
C
C    >-  THE OBJECT IS ON THE TRUE SIDE, INCREMENT THE
C    >-  PLUS COUNT.
C
                        NP = NP + 1
```

```
C
                        ELSEIF (TRUFLS(KOBJ,CNDPNT).EQ.-1) THEN
C
C     >-  THE OBJECT IS ON THE FALSE SIDE, INCREMENT THE
C     >-  MINUS COUNT.
C
                          NM = NM + 1
C
                        ELSE
C
C     >-  THE OBJECT INTERSECTS THE PLANE. FLAG IT AS NOT
C     >-  A CANDIDATE PLANE.
C
                          ICAND = .FALSE.
C
                        ENDIF
                      ENDDO
C
C     >-  IF THIS IS A SEPARATION PLAN THEN CHECK TO
C     >-  SEE IF IT DIVIDES THE SUBSET INTO THE LARGEST
C     >-  GROUP FOUND SO FAR. IF IT DOES THEN SAVE IT
C     >-  AS THE SELECTED DIVIDING PLANE.
C
                    IF ((ICAND).AND.(NP.NE.0).AND.(NM.NE.0)) THEN
C
C     >-  COMPUTE CURRENT PLUS/MINUS DIFFERENCE
C
                      CURNC = IABS(NP - NM)
C
C     >-  IF THIS NEW DIFFERENCE IS LESS THAN THE OLD DIFFERENCE
C     >-  SAVE THE DIFFERENCE, THE PLANE NUMBER, THE
C     >-  NUMBER OF ELEMENTS ON THE TRUE SIDE, AND
C     >-  THE NUMBER OF ELEMENTS ON THE FALSE SIDE.
C
                      IF (CURNC.LT.NC) THEN
                        NC    = CURNC
                        ISEL  = CNDPNT
                        NPSEL = NP
                        NMSEL = NM
C
C     >-  CALCULATE PLANE COUNT FOR ALL THE OBJECTS
C
                        NPALL = 0
                        NMALL = 0
                        DO I = 1,OBJCNT
                          IF (TRUFLS(I,CNDPNT).EQ.1) THEN
                            NPALL = NPALL + 1
                          ELSEIF (TRUFLS(I,CNDPNT).EQ.-1) THEN
                            NMALL = NMALL + 1
                          ENDIF
                        ENDDO
C
C     >-  SAVE COUNT IN NCOLD FOR LATTER COMPARISON
C
                        NCOLD = IABS(NPALL - NMALL)
C
C     >-  IF THEY ARE EQUAL, THEN CHECK TO SEE WHICH ONE DIVIDES
C     >-  THE MOST OBJECTS.
C
                      ELSEIF (CURNC.EQ.NC) THEN
C
C     >-  CALCULATE THE PLANE COUNT FOR THE NEW PLANE
C
```

```
                        NPALL = 0
                        NMALL = 0
                        DO I = 1,OBJCNT
                          IF (TRUFLS(I,CNDPNT).EQ.1) THEN
                            NPALL = NPALL + 1
                          ELSEIF (TRUFLS(I,CNDPNT).EQ.-1) THEN
                            NMALL = NMALL + 1
                          ENDIF
                        ENDDO
                        NCNEW = IABS(NPALL - NMALL)
C
C    >- IF THE NEW ONE DIVIDES MORE OBJECTS, USE IT.
C
                        IF (NCNEW.LT.NCOLD) THEN
                          NC    = CURNC
                          NCOLD = NCNEW
                          ISEL  = CNDPNT
                          NPSEL = NP
                          NMSEL = NM
                        ENDIF
                      ENDIF
                    ENDIF
                  ENDDO
C
C    >-  IF A CONDIDATE PLANE COULD NOT BE FOUND TO DIVIDE
C    >-  THE SUBSET INTO TWO GROUPS BY THE BINARY DIVISION
C    >-  METHOD THEN NOTIFY THE OPERATOR AND EXIT.
C
                  IF (ISEL.EQ.0) THEN
                    WRITE(6,200)
200                 FORMAT(1H ,'SEPARATION PLANES COULD NOT BE FOUND.')
                    KSUB = NSET
                    SYSERR = .TRUE.
                  ELSE
C
C    >-  CHECK THE NECESSARY PLANE LIST TO SEE IF THIS
C    >-  PLANE WAS ALREADY SELECTED.
C
                    I = 0
                    FOUND = .FALSE.
C
                    DO WHILE ((.NOT.FOUND).AND.(I.LT.NEEDCO))
                      I = I + 1
                      IF (INEED(1,I).EQ.ISEL) THEN
                        FOUND = .TRUE.
                      ENDIF
                    ENDDO
C
C    >-  IF THE PLANE WAS ALREADY SELECTED THEN SET THE
C    >-  PLANE POINTER AND THE TRUE/FALSE PAIR COUNT SO
C    >-  THAT THE NEW DATA WILL BE ADDED TO THE EXISTING
C    >-  DATA.
C
                    IF (FOUND) THEN
                      IPLNPT = I
                      TFCNT  = INEED(2,I)
                  ELSE
C
C    >-  THIS IS A NEW PLANE AND NEEDS TO BE ADDED TO THE
C    >-  NECESSARY PLANE LIST. SET THE PLANE POINTER AND
C    >-  THE TRUE/FALSE PLANE COUNT FOR A NEW PLANE. ALSO
C    >-  INCREMENT THE NECESSARY PLANE COUNT.
C
```

```
                        NEEDCO = NEEDCO + 1
                        IPLNPT = NEEDCO
                        TFCNT  = 0
                      ENDIF
C
C    >-  INITIALIZE THE TRUE COUNT AND THE FALSE COUNT
C    >-  AND THE TRUE/FALSE PLANE COUNT.
C
                      ITRCNT = 0
                      IFLCNT = 0
C
C    >-  LOOP THRU THE ELEMENTS OF THE SUBSET TO FIND
C    >-  AN ELEMENT ON THE TRUE SIDE OF THE PLANE
C
                      DO I = KSUB, KSUB + NELE - 1
C
C    >-  SET THE TRUE POINTER TO THE PROPER VALUE
C
                        ITRPNT = ISET(I)
C
C    >-  IF THE ELEMENT DISIGNATED BY THE TRUE POINTER
C    >-  IS IN FACT ON THE TRUE SIDE OF THE PLANE THEN
C    >-  SEARCH THRU THE TRUE/FALSE DATA TABLE FOR THIS
C    >-  SUBSET AND FIND THE ELEMENTS ON THE FALSE SIDE
C    >-  GENERATING A TRUE/FALSE PAIR.
C
                       IF (TRUFLS(ITRPNT,ISEL).EQ.1) THEN
C
C    >-  SAVE THE ELEMENT NUMBER AND INCREMENT
C    >-  THE TRUE ELEMENT COUNT.
C
                         ITRCNT = ITRCNT + 1
                         LTRUE(ITRCNT) = ITRPNT
C
C    >-  SEARCH THRU SUBSET OF TRUE/FALSE DATA TABLE
C    >-  TO FIND FALSE ELEMENTS AND GENERATE TRUE/FALSE
C    >-  PAIRS.
C
                         DO J = KSUB, KSUB + NELE - 1
C
C    >-  SET THE FALSE POINTER TO THE PROPER VALUE
C
                           IFLPNT = ISET(J)
C
C    >-  CHECK FOR A FALSE ELEMENT
C
                           IF (TRUFLS(IFLPNT,ISEL).EQ.-1) THEN
C
C    >-  INCREMENT THE TRUE/FALSE PAIR COUNT
C    >-  AND STORE THE TRUE FALSE PAIR.
C
                             TFCNT = TFCNT + 1
                             ITRUE(IPLNPT,TFCNT) = ITRPNT
                             IFALSE(IPLNPT,TFCNT) = IFLPNT
                           ENDIF
                         ENDDO
C
C    >-  IF THE TRUE POINTER DOES NOT POINT TO
C    >-  AN ELEMENT ON THE TRUE SIDE OF THE PLANE
C    >-  THEN STORE IT AS A FALSE ELEMENT AND
C    >-  INCREMENT THE FALSE ELEMENT COUNT.
C
                       ELSEIF(TRUFLS(ITRPNT,ISEL).EQ.-1) THEN
```

```
C
                        IFLCNT = IFLCNT + 1
                        LFALSE(IFLCNT) = ITRPNT
                      ENDIF
                    ENDDO
C
C    >-  SAVE THE SELECTED PALNE NUMBER AND THE NUMBER OF
C    >-  TRUE/FALSE PAIRS ASSOCIATED WITH THIS PLANE INTO
C    >-  THE NECESSARY PLANE ARRAY AND INCREMENT THE
C    >-  NUMBER OF NECESSARY PLANES.
C
                    INEED(1,IPLNPT) = ISEL
                    INEED(2,IPLNPT) = IFCNT
C
C    >-  SORT THE SUBSET INTO A TRUE GROUP AND A FALSE GROUP.
C
                    DO I = 1,ITRCNT
                      K = KSUB + I - 1
                      ISET(K) = LTRUE(I)
                    ENDDO
C
                    DO I = 1,IFLCNT
                      K = K + 1
                      ISET(K) = LFALSE(I)
                    ENDDO
C
C    >-  PLACE THE NEW SUBSET COUNTS INTO THE SUVET COURNT LIST
C
                    J = NSET - KSUB + 1
                    DO I = 1,J
                      K=NSET + 1 - I
                      ISUB(K+1) = ISUB(K)
                    ENDDO
                    ISUB(KSUB) = NPSEL
                    ISUB(KSUB+1) = NMSEL
C
C    >-  RESET THE SUBSET POINTER
C
                    KSUB = KSUB - 1
C
C    >-  INCREMENT TOTAL NUMBER OF SETS
C
                    NSET = NSET + 1
                  ENDIF
                ENDIF
              ENDDO
            ENDIF
C
C    >-  IF A "LISTABLE" SET OF SEPARATION PLANES COULD BE FOUND THEN
C    >-  WRITE THE INFORMATION FOR THE USER.
C
            IF (.NOT.SYSERR) THEN
              WRITE(6,300) NEEDCO
 300          FORMAT(1H ,'NUMER OF SEPARATION PLANES REQUIRED =',I3,/)
              DO I = 1, NEEDCO
                J = INEED(1,I)
                WRITE(6,325) I,NORMAL(1,J),NORMAL(2,J),NORMAL(3,J),
     +                       DIST(J)
 325            FORMAT(1H ,'FOR PLANE NUMBER',I3,/,
     +                   ' THE UNIT NORMAL IS:',3F5.1,/,
     +                   ' AND THE DIRECTED DISTANCE IS:',F5.1)
                WRITE(6,350)
 350            FORMAT(1H ,'THE TRUE/FALSE PAIRS FOR THIS PLANE',
```

```
     +                    ' ARE:    TRUE     FALSE',/,X,I45,
     +                             '------   -------')
                IFCNT = INEED(2,I)
                DO J = 1,IFCNT
                  WRITE(6,375) ITRUE(I,J),IFALSE(I,J)
 375              FORMAT(X,I46,I3,7X,I3)
                ENDDO
              ENDDO
              WRITE(6,400)
 400          FORMAT(1H , 'NORMAL TERMINATION')
            ENDIF
C
          ENDIF
        ENDIF
      END
```

```
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C
C               ROUTINE NAME: LOAD
C
C                     AUTHOR: KEITH A. STUMP
C                             GENERAL ELECTRIC
C                             SIMULATION AND CONTROL
C                             SYSTEMS DEPARTMENT (SCSD)
C                             DAYTONA BEACH, FL
C
C               DATE CREATED: 12/4/85
C
C>----------------------------------------------------------------------
C
C                ENTRY POINT: LOAD(OBJCNT,OSEPCNT,SYSERR)
C
C           CALLING ROUTINES: SEPPLN
C
C            CALLED ROUTINES: NONE
C
C>----------------------------------------------------------------------
C
C     FUNCTIONAL DESCRIPTION:
C
C          This routine is the main code that gathers the objects
C     of the model. This routine also gathers the user supplied
C     separation planes. All the required plane information is
C     loaded into the model interface common.
C
C>----------------------------------------------------------------------
C
C       VARIABLE DEFINITIONS:
C
C      NAME        TYPE                 DESCRIPTION                 USE
C    --------   ---------   ------------------------------   ----------
C    CRDTYP     INTEGER     CARD TYPE                         LOCAL
C                            V - VERTEX CARD
C                            F - FACE CARD
C                            S - SEPARATION PLANE CARD
C                            E - END OF DATA CARD
C    CROSS      INTEGER     VERTEX CROSS-REFERENCE ARRAY      LOCAL
C    FOUND      LOGICAL     VERTEX FOUND FLAG                 LOCAL
C    HAVOBJ     LOGICAL     VERTEX / FACE OBJECT MATCH FLAG   LOCAL
C    I          INTEGER     INDEX VARIABLE                    LOCAL
C    II         INTEGER     INDEX VARIABLE                    LOCAL
C    IBUFFER    INTEGER     CARD INPUT BUFFER                 LOCAL
C    IEDGE      INTEGER     VERTEX LIST FOR FACE EDGES        COMMON
C    IEDGPNT    INTEGER     FACE EDGE POINTER                 LOCAL
C    IFACE      INTEGER     NUMBER OF VERTICES IN FACE        COMMON
C    IFACPNT    INTEGER     FACE POINTER                      LOCAL
C    IOBJECT    INTEGER     OBJECT FACE ARRAY                 COMMON
C    IVNUM      INTEGER     VERTEX NUMBER                     LOCAL
C    JFACPNT    INTEGER     PREVIOUS FACE POINTER             LOCAL
C    LOOP       LOGICAL     CARD LOOP FLAG                    LOCAL
C    NODES      INTEGER     FACE VERTEX NODE NUMBERS          LOCAL
C    OBJCNT     INTEGER     MODEL OBJECT COUNT                PASSED
C    OSEPCNT    INTEGER     SUPPLIED SEPARATION PLANE COUNT   PASSED
C    OSTART     INTEGER     OBJECT START VERTEX               LOCAL
C    OVERTEX    INTEGER     SUPPLIED VERTEX NUMBER ARRAY      COMMON
C    SYSERR     LOGICAL     SYSTEM ERROR FLAG                 PASSED
C    VERCNT     INTEGER     VERTEX COUNT                      LOCAL
C    XPOS1      REAL        X VERTEX POSITION 1               LOCAL
C    XPOS2      REAL        X VERTEX POSITION 2               LOCAL
```

```
C     XPOS3        REAL        X VERTEX POSITION 3                LOCAL
C     XVERT        REAL        X VERTEX ARRAY                     COMMON
C     YPOS1        REAL        Y VERTEX POSITION 1                LOCAL
C     YPOS2        REAL        Y VERTEX POSITION 2                LOCAL
C     YPOS3        REAL        Y VERTEX POSITION 3                LOCAL
C     YVERT        REAL        Y VERTEX ARRAY                     COMMON
C     ZPOS1        REAL        Z VERTEX POSITION 1                LOCAL
C     ZPOS2        REAL        Z VERTEX POSITION 2                LOCAL
C     ZPOS3        REAL        Z VERTEX POSITION 3                LOCAL
C     ZVERT        REAL        Z VERTEX ARRAY                     COMMON
C
C<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE LOAD(OBJCNT,OSEPCNT,SYSERR)
C
      IMPLICIT NONE
C
C    >-  DECLARE OBJECT/MODEL INTERFACE COMMON
C
      INTEGER IEDGE,IFACE,IOBJECT,OVERTEX,TRUFLS
      REAL    DIST,NORMAL,XVERT,YVERT,ZVERT
      COMMON/MODEL/ IEDGE(200),IFACE(100),IOBJECT(10),OVERTEX(10),
     +              XVERT(100),YVERT(100),ZVERT(100),
     +              DIST(100),NORMAL(3,100),TRUFLS(10,100)
C
C    >-  DECLARE LOCAL VARIABLE TYPES
C
      INTEGER CRDTYP,I,II,IEDGCNT,IEDGPNT,IFACPNT,IVNUM,JFACPNT
      INTEGER OBJCNT,OSEPCNT,OSTART,VERCNT
      INTEGER CROSS(100),IBUFFER(20),NODES(10)
      REAL    XPOS1,YPOS1,ZPOS1,XPOS2,YPOS2,ZPOS2,XPOS3,YPOS3,ZPOS3
      LOGICAL FOUND,HAVOBJ,LOOP,SYSERR
C
C    >-  OPEN THE DATA FILE
C
      OPEN(UNIT=11, TYPE='OLD', FILE='OBJECT.DATA',
     +     FORM='FORMATTED', ERR=990)
C
C    >-  INITIALIZE LOCAL VARIABLES
C
      OSEPCNT = 0
      VERCNT = 0
      OSTART = 0
      IEDGPNT = 0
      OBJCNT = -1
      JFACPNT = 0
      SYSERR = .FALSE.
      HAVOBJ = .TRUE.
C
C    >-  READ UNTIL END OF FILE
C
      LOOP = .TRUE.
      DO WHILE (LOOP)
C
C    >-  READ THE RECORD FROM THE FILE
C
        READ(11,100,ERR=999) IBUFFER
  100   FORMAT(20A4)
C
C    >-  DECODE CARD TYPE
C
        DECODE(2,200,IBUFFER,ERR=980) CRDTYP
  200   FORMAT(A2)
```

```
C
C    >-  CHECK FOR A VERTEX CARD
C
          IF (CRDTYP.EQ.'V ') THEN
C
C    >-  SET THE POINTERS FOR THE OBJECT
C
            IF (HAVOBJ) THEN
              HAVOBJ = .FALSE.
              OSTART = VERCNT + 1
              OBJCNT = OBJCNT + 1
C
C    >-  STORE THE FACE COUNT FOR THIS OBJECT
C
              IF (OBJCNT.GE.1) THEN
                IOBJECT(OBJCNT) = IFACPNT - JFACPNT
                JFACPNT = IFACPNT
              ENDIF
            ENDIF
C
C    >-  DECODE THE REST OF THE VERTEX INFORMATION
C
            DECODE(31,300,IBUFFER,ERR=980) IVNUM,XPOS1,YPOS1,ZPOS1
 300        FORMAT(2X,I3,3F8.2)
C
C    >-  INCREMENT THE VERTEX COUNT AND CHECK FOR ERROR
C
            VERCNT = VERCNT + 1
            IF (VERCNT.LE.100) THEN
C
C    >-  STORE THE VERTEX VALUES IN THE OBJECT COMMON
C
              XVERT(VERCNT) = XPOS1
              YVERT(VERCNT) = YPOS1
              ZVERT(VERCNT) = ZPOS1
C
C    >-  SET THE CROSS-REFERENCE ARRAY FOR THE OBJECT
C
              CROSS(VERCNT) = IVNUM
            ELSE
C
C    >-  CURRENT ARRAY LIMITS EXCEEDED. NOTIFY OPERATOR
C    >-  AND EXIT
C
              WRITE(6,350)
 350          FORMAT(1H ,'VERTEX ARRAY LIMIT EXCEEDED.',/,
     +                 ' PROGRAM EXITING...')
              SYSERR = .TRUE.
            ENDIF
C
C    >-  CHECK FOR A FACE CARD
C
          ELSEIF (CRDTYP.EQ.'F ') THEN
C
C    >-  CHECK TO MAKE SURE VERTICES FOR THE FACE
C    >-  HAVE BEEN GIVEN.
C
            IF ((VERCNT - OSTART).GT.0) THEN
C
C    >-  DECODE THE EDGE INFORMATION FOR THE FACE.
C    >-  THE FACES HAVE BEEN LIMITED TO TEN VERTICES.
C
              DECODE(32,400,IBUFFER,ERR=980) (NODES(I),I=1,10)
```

```
 400        FORMAT(2X,1013)
C
C    >-  STORE THE EDGE INFORMATION IN THE EDGE ARRAY.
C    >-  USE THE CROSS-REFERENCE ARRAY TO OBTAIN THE
C    >-  CURRENT LOCATION OF THE VERTEX.
C
            I = 1
            DO WHILE (NODES(I).NE.0)
C
C    >-  CHECK ONLY THE VERTICES FOR THIS OBJECT
C    >-  THE VARIABLE FOUND IS USED TO MAKE SURE A
C    >-  VERTEX IS FOUND
C
              FOUND = .FALSE.
              DO II = OSTART, VERCNT
                IF (NODES(I).EQ.CROSS(II)) THEN
                  FOUND = .TRUE.
                  IEDGPNT = IEDGPNT + 1
C
C    >-  CHECK FOR ARRAY OVERFLOW. IF SO NOTIFY OPERATOR.
C
                  IF (IEDGPNT.GT.200) THEN
                    NODES(I + 1) = 0
                    SYSERR = .TRUE.
                    WRITE(6,415)
415                 FORMAT(1H ,'EDGE ARRAY LIMIT EXCEEDED.',/,
     +                         ' PROGRAM EXITING...')
                  ELSE
                    IEDGE(IEDGPNT) = II
                  ENDIF
                ENDIF
              ENDDO
C
C    >-  IF FOUND IS STILL FALSE AN ERROR HAS OCCURED.
C    >-  NOTIFY OPERATOR AND EXIT.
C
              IF (.NOT.FOUND) THEN
                NODES(I) = 0
                SYSERR = .TRUE.
                WRITE(6,425)
 425            FORMAT(1H ,'VERTEX FOR FACE NOT FOUND.',/,
     +                   ' PROGRAM EXITING...')
              ELSE
C
C    >-  INCREMENT COUNTER AND CONTINUE
C
                I = I + 1
              ENDIF
            ENDDO
C
C    >-  STORE THE EDGE COUNT IN THE FACE ARRAY
C
            IFACPNT = IFACPNT + 1
            IFACE(IFACPNT) = I - 1
C
C    >-  SET HAVE OBJECT FLAG
C
            HAVOBJ = .TRUE.
          ELSE
C
C    >-  NO VERTICES WERE FOUND FOR FACE.
C    >-  NOTIFY OPERATOR AND EXIT.
C
```

```
            WRITE(6,450)
 450        FORMAT(1H ,'NO VERTICES FOR FACE WERE GIVEN.',/,
     +              ' PROGRAM EXITING...')
            SYSERR = .TRUE.
          ENDIF
C
C    >-  CHECK FOR OPERATOR INPUT SEPARATION PLANES
C
        ELSEIF (CRDTYP.EQ.'S ') THEN
C
C    >-  DECODE PLANE INFORMATION
C
          DECODE(74,500,IBUFFER,ERR=980) XPOS1,YPOS1,ZPOS1,
     +          XPOS2,YPOS2,ZPOS2,XPOS3,YPOS3,ZPOS3
 500      FORMAT(2X,9F8.2)
C
C    >-  LOAD VERTEX INFORMATION
C
          IF ((VERCNT+3).LE.100) THEN
            XVERT(VERCNT + 1) = XPOS1
            YVERT(VERCNT + 1) = YPOS1
            ZVERT(VERCNT + 1) = ZPOS1
            XVERT(VERCNT + 2) = XPOS2
            YVERT(VERCNT + 2) = YPOS2
            ZVERT(VERCNT + 2) = ZPOS2
            XVERT(VERCNT + 3) = XPOS3
            YVERT(VERCNT + 3) = YPOS3
            ZVERT(VERCNT + 3) = ZPOS3
C
C    >-  SET VERTEX START LOCATION
C
            OSEPCNT = OSEPCNT + 1
            OVERTEX(OSEPCNT) = VERCNT + 1
C
C    >-  INCREMENT VERTEX COUNT BY 3
C
            VERCNT = VERCNT + 3
          ELSE
C
C    >-  CURRENT ARRAY LIMITS EXCEEDED.
C    >-  NOTIFY OPERATOR AND EXIT.
C
            WRITE(6,350)
            SYSERR = .TRUE.
          ENDIF
C
C    >-  CHECK FOR END OF DATA CARD
C
        ELSEIF (CRDTYP.EQ.'E ') THEN
C
C    >-  SET LOOP FLAG FALSE TO STOP
C
          LOOP = .FALSE.
C
C    >-  IF CARD TYPE HAS NOT BEEN FOUND THEN A
C    >-  ERROR CONDITION EXISTS. NOTIFY OPERATOR.
C
        ELSE
          WRITE(6,600)
 600      FORMAT(1H ,'INVALID CARD TYPE. CHECK INPUT FILE.')
        ENDIF
C
C    >-  IF SYSTEM ERROR, THEN EXIT LOOP.
```

```
C
       IF (SYSERR) LOOP = .FALSE.
      ENDDO
C
C    >-  ADJUST OBJECT COUNT, AND SKIP TO END.
C
      OBJCNT = OBJCNT + 1
      IOBJECT(OBJCNT) = IFACPNT - JFACPNT
      GOTO 1000
C
C    >-  TEST FOR OPERATION ERRORS
C
 980  CONTINUE
      WRITE(6,700)
 700  FORMAT(1H ,'ERROR DURING CARD DECODE',/,
     +           ' PROGRAM EXITING...')
      SYSERR = .TRUE.
      GOTO 1000
 990  CONTINUE
      WRITE(6,750)
 750  FORMAT(1H ,'ERROR DURING FILE OPEN',/,
     +           ' PROGRAM EXITING...')
      SYSERR = .TRUE.
      GOTO 1000
 999  CONTINUE
      WRITE(6,800)
 800  FORMAT(1H ,'ERROR DURING FILE READ',/,
     +           ' PROGRAM EXITING...')
      SYSERR = .TRUE.
      GOTO 1000
C
 1000 CONTINUE
      CLOSE(UNIT=11)
      RETURN
      END
```

```
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C
C               ROUTINE NAME: TRUE_FALSE
C
C                     AUTHOR: KEITH A. STUMP
C                             GENERAL ELECTRIC (SCSD)
C                             DAYTONA BEACH, FL
C
C               DATE CREATED: 12/4/85
C
C>----------------------------------------------------------------------
C
C                ENTRY POINT: TRUE_FALSE(OBJCNT,IPLNCO,EDGPNT,ICODE)
C
C           CALLING ROUTINES: SEPPLN
C
C            CALLED ROUTINES: NONE
C
C>----------------------------------------------------------------------
C
C     FUNCTIONAL DESCRIPTION:
C
C          This routine loads the True/False state table
C     which shows the state of all objects with a given separation
C     plane. This routine checks the used vertices of an object
C     to see if those vertices lie on the true or false side of the
C     plane.
C          The True/False state table contains an entry for each
C     potential plane which shows for each object whether that object
C     is on the true side, false side, or is intersected by the plane.
C     The information is stored in the table as:
C
C                         -1 = False side;
C                          1 = True side;
C                and       0 = intersected by.
C
C>----------------------------------------------------------------------
C
C       VARIABLE DEFINITIONS:
C
C       NAME        TYPE                  DESCRIPTION                 USE
C     --------    ---------    ------------------------------    ----------
C     DIST        REAL         DIRECTED DISTANCE                  COMMON
C     EDGPNT      INTEGER      EDGE POINTER                       PASSED
C     FIRST       LOGICAL      FIRST VERTEX FLAG                  LOCAL
C     ICODE       INTEGER      INPUT TYPE CODE                    PASSED
C                               1 = OBJECT VERTEX LIST
C                               2 = SEPARATION PLANE LIST
C     IEDGCNT     INTEGER      FACE EDGE COUNT                    LOCAL
C     IEDGE       INTEGER      OBJECT EDGE ARRAY                  COMMON
C     IEDGPNT     INTEGER      FACE EDGE POINTER                  LOCAL
C     IFAC        INTEGER      FACE INDEX                         LOCAL
C     IFACCNT     INTEGER      OBJECT FACE COUNT                  LOCAL
C     IFACE       INTEGER      FACE EDGE COUNT ARRAY              COMMON
C     IFACPNT     INTEGER      OBJECT FACE POINTER                LOCAL
C     INTER       LOGICAL      INTERSECTION FLAG                  LOCAL
C     IOBJ        INTEGER      OBJECT INDEX COUNT                 LOCAL
C     IOBJECT     INTEGER      OBJECT FACE ARRAY                  COMMON
C     IPLNCO      INTEGER      SEPARATION PLANE COUNT             PASSED
C     IVERT       INTEGER      VERTEX INDEX                       LOCAL
C     NORM        REAL         FACE NORMAL                        LOCAL
C     NORMAL      REAL         FACE NORMAL ARRAY                  COMMON
C     OBJCNT      INTEGER      OBJECT COUNT                       PASSED
```

```
C      OSTATE      INTEGER      OLD VERTEX TRUE/FALSE STATE        LOCAL
C      PLNDST      REAL         PLANE DIRECTED DISTANCE            LOCAL
C      STATE       INTEGER      VERTEX TRUE/FALSE STATE            LOCAL
C      TRUFLS      INTEGER      OBJECT TRUE/FALSE ARRAY            LOCAL
C      VDST        REAL         VERTEX DIRECTED DISTANCE           LOCAL
C      VECA        REAL         VECTOR A                           LOCAL
C      VECB        REAL         VECTOR B                           LOCAL
C      VMAG        REAL         VECTOR MAGNITUDE                   LOCAL
C      XSAVE       REAL         FIRST NODE X LOCATION              LOCAL
C      XVERT       REAL         NODE X VERTEX ARRAY                COMMON
C      YSAVE       REAL         FIRST NODE Y LOCATION              LOCAL
C      YVERT       REAL         NODE Y VERTEX ARRAY                COMMON
C      ZSAVE       REAL         FIRST NODE Z LOCATION              LOCAL
C      ZVERT       REAL         NODE Z VERTEX ARRAY                COMMON
C
C<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE TRUE_FALSE(OBJCNT,IPLNCO,EDGPNT,ICODE)
C
      IMPLICIT NONE
C
C    >-  OBJECT/MODEL INTERFACE COMMON
C
      INTEGER IEDGE,IFACE,IOBJECT,OVERTEX,TRUFLS
      REAL    DIST,NORMAL,XVERT,YVERT,ZVERT
      COMMON /MODEL/ IEDGE(200),IFACE(100),IOBJECT(10),OVERTEX(10),
     +               XVERT(100),YVERT(100),ZVERT(100),
     +               DIST(100),NORMAL(3,100),TRUFLS(10,100)
C
C    >-  DECLARE LOCAL VARIABLE TYPES
C
      INTEGER EDGPNT,I,ICODE,IEDGCNT,IEDGPNT,IFAC,IFACCNT,IFACPNT
      INTEGER IOBJ,IPLNCO,IVERT,OBJCNT,OSTATE,STATE
      REAL PLNDST,VDST,VMAG
      REAL NORM(3),VECA(3),VECB(3),XSAVE,YSAVE,ZSAVE
      LOGICAL FIRST,INTER
C
C    >-  USE THREE VERTICES TO CALCULATE FACE NORMAL.
C    >-  CALCULATE VECTOR A, NODE 1 - NODE 2.
C
      IF (ICODE.EQ.1) THEN
        VECA(1) = XVERT(IEDGE(EDGPNT))
     +          - XVERT(IEDGE(EDGPNT + 1))
        VECA(2) = YVERT(IEDGE(EDGPNT))
     +          - YVERT(IEDGE(EDGPNT + 1))
        VECA(3) = ZVERT(IEDGE(EDGPNT))
     +          - ZVERT(IEDGE(EDGPNT + 1))
C
C    >-  CALCULATE VECTOR B, NODE 3 - NODE 2
C
        VECB(1) = XVERT(IEDGE(EDGPNT + 2))
     +          - XVERT(IEDGE(EDGPNT + 1))
        VECB(2) = YVERT(IEDGE(EDGPNT + 2))
     +          - YVERT(IEDGE(EDGPNT + 1))
        VECB(3) = ZVERT(IEDGE(EDGPNT + 2))
     +          - ZVERT(IEDGE(EDGPNT + 1))
C
        XSAVE = XVERT(IEDGE(EDGPNT))
        YSAVE = YVERT(IEDGE(EDGPNT))
        ZSAVE = ZVERT(IEDGE(EDGPNT))
      ELSE
        VECA(1) = XVERT(EDGPNT)
     +          - XVERT(EDGPNT + 1)
```

```
      VECA(2) = YVERT(EDGPNT)
     +        - YVERT(EDGPNT + 1)
      VECA(3) = ZVERT(EDGPNT)
     +        - ZVERT(EDGPNT + 1)
C
C     >-  CALCULATE VECTOR B, NODE 3 - NODE 2
C
      VECB(1) = XVERT(EDGPNT + 2)
     +        - XVERT(EDGPNT + 1)
      VECB(2) = YVERT(EDGPNT + 2)
     +        - YVERT(EDGPNT + 1)
      VECB(3) = ZVERT(EDGPNT + 2)
     +        - ZVERT(EDGPNT + 1)
C
      XSAVE = XVERT(EDGPNT)
      YSAVE = YVERT(EDGPNT)
      ZSAVE = ZVERT(EDGPNT)
    ENDIF
C
C     >-  CROSS VECTORS TO FORM NORMAL
C
    NORM(1) = VECA(2) * VECB(3) - VECB(2) * VECA(3)
    NORM(2) = VECB(1) * VECA(3) - VECA(1) * VECB(3)
    NORM(3) = VECA(1) * VECB(2) - VECB(1) * VECA(2)
C
C     >-  FIND MAGNITUDE OF FACE NORMAL
C
    VMAG = SQRT(NORM(1) * NORM(1) +
   +            NORM(2) * NORM(2) +
   +            NORM(3) * NORM(3))
C
C     >-  NORMALIZE FACE NORMAL
C
    NORM(1) = NORM(1) / VMAG
    NORM(2) = NORM(2) / VMAG
    NORM(3) = NORM(3) / VMAG
C
C     >-  CALCULATE DIRECTED DISTANCE OF PLANE.
C     >-  THIS IS DONE USING THE FIRST VERTEX.
C
    PLNDST = (NORM(1) * XSAVE +
   +          NORM(2) * YSAVE +
   +          NORM(3) * ZSAVE )
C
C     >-  LOAD THE UNIT NORMALS AND DIRECTED DISTANCE FOR THIS
C     >-  PLANE.
C
    NORMAL(1,IPLNCO) = NORM(1)
    NORMAL(2,IPLNCO) = NORM(2)
    NORMAL(3,IPLNCO) = NORM(3)
    DIST(IPLNCO) = PLNDST
C
C     >-  INITIALIZE FACE AND EDGE POINTERS
C
    IFACPNT = 1
    IFACCNT = 0
    IEDGPNT = 1
    IEDGCNT = 0
C
C     >-  LOOP THRU ALL OBJECTS TO CHECK SEPARATION
C
    DO IOBJ = 1,OBJCNT
C
```

```
C    >-  SET FACE COUNTS AND POINTERS FOR CURRENT OBJECT
C
          IFACPNT = IFACPNT + IFACCNT
          IFACCNT = IOBJECT(IOBJ)
C
C    >-  STEP THRU EACH FACE OF THE OBJECT UNTIL DONE
C    >-  OR A INTERSECTION IS FOUND. INITIALIZE FACE POINTER
C    >-  AND INTERSECTION FLAG.
C
          IFAC = IFACPNT - 1
          INTER = .FALSE.
C
          DO WHILE (IFAC.LT.(IFACPNT+IFACCNT-1))
C
C    >-  INCREMENT FACE POINTER
C
            IFAC = IFAC + 1
C
C    >-  SET EDGE COUNTS AND POINTERS FOR CURRENT FACE
C
            IEDGPNT = IEDGPNT + IEDGCNT
            IEDGCNT = IFACE(IFAC)
C
C    >-  STEP THRU EACH VERTEX UNTIL DONE OR A
C    >-  INTERSECTION IS FOUND. INITIALIZE VERTEX POINTER
C    >-  AND FIRST VERTEX FLAG
C
            IVERT = IEDGPNT - 1
            FIRST = .TRUE.
            DO WHILE ((IVERT.LT.(IEDGPNT+IEDGCNT-1)).AND.(.NOT.INTER))
C
C    >-  INCREMENT VERTEX POINTER
C
              IVERT = IVERT + 1
C
C    >-  DETERMINE WHETHER A VERTEX IS ON THE TRUE SIDE,
C    >-  ON THE FALSE SIDE OR INTERSECTS THE PLANE
C    >-  BEING ANALYZED.
C    >-  COMPUTE THE DIFFERENCE BETWEEN THE PERPENDICULAR
C    >-  DISTANCE FROM THE VERTEX TO THE PLANE.
C
              VDST = (NORM(1) * XVERT(IEDGE(IVERT)) +
     +                NORM(2) * YVERT(IEDGE(IVERT)) +
     +                NORM(3) * ZVERT(IEDGE(IVERT))) -
     +                PLNDST
C
C    >-  DETERMINE THE VALUE OF STATE BASED ON THE
C    >-  VALUE OF VDST
C
              IF (ABS(VDST).LT..001) THEN
                STATE = 0
              ELSEIF (VDST.LT.0.0) THEN
                STATE = -1
              ELSE
                STATE = 1
              ENDIF
C
C    >-  CHECK STATES TO SEE IF THEY HAVE SWAPPED,
C    >-  MEANING AN INTERSECTION. IF THE VERTEX IS ON THE
C    >-  PLANE THEN THIS PRODUCES A DON'T CARE SITUATION.
C
               IF (STATE.NE.0) THEN
C
```

```
                IF (FIRST) THEN
C
                  FIRST = .FALSE.
                  OSTATE = STATE
C
                ELSEIF (OSTATE.NE.STATE) THEN
C
                  INTER = .TRUE.
                ENDIF
              ENDIF
            ENDDO
          ENDDO

C
C     >-  LOAD THE TRUE/FALSE DATA TABLE WITH THE OBJECT STATE
C     >-  FOR THIS PLANE.
C
          IF (INTER) THEN
            TRUFLS(IOBJ,IPLNCO) = 0
          ELSE
            TRUFLS(IOBJ,IPLNCO) = OSTATE
          ENDIF
C
        ENDDO
C
        RETURN
        END
```

REFERENCES

Bennett, William S. "Computer-Generated Graphics." In Computer Image Generation, pp. 9-26. Edited by Bruce J. Schachter. New York: John Wiley & Sons, 1983.

Foley, James D., and Van Dam, Andries. Fundamentals of Interactive Computer Graphics. Reading, Massachusetts: Addison-Wesley Publishing Co., 1982.

Giloi, Wolfgang K. Interactive Computer Graphics. Princeton, New Jersey: Prentice-Hall, Inc., 1978.

Newman, William M., and Sproull, Robert F. Principles of Interactive Computer Graphics. New York: McGraw-Hill Book Co., 1979.

Pavilidis, Theo. Algorithms for Graphics and Image Processing. Rockville, Maryland: Computer Science Press, 1982.

Schachter, Bruce J., ed. Computer Image Generation. New York: John Wiley & Sons, 1983.

Steiner, Walter R. "Survey of Texture and Shading Techniques for Visual Flight Simulation." Masters thesis, University of Central Florida, 1985.