An Aid to Hidden Surface Removal in Real Time CGI Systems

D. J. Tomlinson
7 Ryecroft Gardens, Goring-by-Sea, Worthing, Sussex, UK
Formally of: Department of Engineering and Applied Science, University of Sussex, UK

One of the limiting factors in the development of real-time Computer Generated Image systems for displaying perspective colour scenes is finding a suitable solution to the hidden surface problem. This paper presents a solution to this problem, whereby spatial properties of a three-dimensional 'model' held in the Computer Generated Image system database are utilized in order to minimize hidden surface computations for producing moving perspective scenes. This solution is particularly acceptable to situations involving a fixed model and moving eye point. Often Computer Generated Image systems resort to processing individual surfaces from objects in the model to obtain display priority levels; the solution presented is based upon the use of preprocessed relative priority levels which only require complete objects to be given priority levels thus reducing real-time computation. Once an absolute display priority level for an object is found it will be shown that all surfaces forming that object have known display priority levels. Techniques used to obtain object display priority levels are discussed, together with background information concerning the hidden surface problem.

INTRODUCTION

All Computer Generated Image (CGI) systems designed for the purpose of producing daylight three-dimensional scenes, make use of information specifying surfaces that represent some form of model. The display end of the CGI system necessitates for two-dimensional surfaces to be generated from the three-dimensional surfaces in the model. The calculations to perform the three- to two-dimensional transformation are well understood and enable a viewer to see a two-dimensional picture with perspective depth qualities, thus appearing as a three-dimensional scene. However, the two- to three-dimensional transformations entail depth information to be lost and so extra processing must be performed to remove whole, or part of, two-dimensional surfaces that are hidden behind other surfaces within the scene displayed. The problem of removing these unwanted surfaces is called 'the hidden surface problem'.

This is one of the most challenging problems in computer graphics and has been, and still is, a major obstacle in the development of computer graphic systems for displaying realistic three-dimensional scenes.

For real-time applications the necessary high speed of removal of occulted surfaces makes the hidden surface problem difficult to overcome.

REVIEW OF PAST SOLUTIONS

For a line drawing three-dimensional scene of any general layout, the number of computer calculations required to perform hidden line removal grows exponentially with the scene complexity. If the scene is restricted to being composed from concave surfaces then, as shown in two papers by Sutherland,[1,2] the number of computer calculations to remove hidden lines is reduced to the square of the number of surfaces within the scene.

One of the first solutions to the hidden surface problem was published by Jones.[3] The method described by Jones is to examine all the points along a given line of sight. The line of sight is terminated when it 'hits' an opaque surface. Thus, surfaces represent areas upon the display screen which 'mask' further projection of the line of sight. Since many lines of sight need to be examined to produce a complete display screen, this process is very slow. Jones' algorithm assumes that there are ever-present outer surfaces which restrict a line of sight from being infinitely extended, and the algorithm is best utilized for a scene consisting of large surfaces near to the viewer.

Many solutions to the hidden surface problem have been based upon comparing each surface plane within the model with every other. These comparisons are based on a measurement of distance from the viewer. However, this type of solution requires massive computing power and is not suitable for real-time applications.

The first solution to the hidden surface problem, which enabled a real-time solution to be thought possible, was presented by Warnock.[4] Essentially, Warnock solves the hidden surface problem by dividing the picture up into four sub-pictures. Each sub-picture is examined in turn to find display contentions. If the sub-picture contains no display anomalies, then the contents of the sub-picture are placed in a display file, otherwise it is divided up into four more sub-pictures. Division of a sub-picture continues until the size of the resultant sub-pictures are smaller than the picture resolution, in which case the display file contains a reference to the nearest of the surfaces in contention. The Warnock algorithm proved to be a milestone in the development of an ideal solution to the hidden surface problem and was capable of processing a picture consisting of 1000 lines in less than one minute.

In 1969, in addition to Warnock's paper, another paper from the University of Utah was published, by Romney, Watkins and Evans.[5] This paper was directly aimed at the advocation of a real-time solution to the hidden surface problem. The hidden line algorithm devised by Romney *et al.* placed the restriction that the displayed scene must consist of triangular planar surfaces. The algorithm is based upon the inspection of an imaginary scan line which traverses horizontally across

CCC-0010-4620/82/0025-0429 $06.50

the display window. Along this scan line the transition points, from one triangle to another, are found. If two or more triangles are found to be on the same scan line point, then a depth sort technique is used to establish which one should 'enter' the scan line list. The scan line list holds the order of the transition points and their x-begin and x-end values.

The prominent feature of the algorithm is that the order of one scan line transition points are used to determine the next scan line transition points. The order of transition points for each scan line remain the same unless a triangle enters or exits from the displayed scene. For each new scan line, new x-begin and x-end values are calculated. Storing the order of the scan line transition points greatly speeds up the cycle time of the algorithm. The exact speed of the algorithm depends on the number of times the order of the scan line transition points are changed but typically the algorithm can remove the hidden lines from a scene consisting of 100 triangles in about five seconds. Since the algorithm is scan line based, it was implemented by Romney *et al.*, on a computer graphics system which had a colour raster scan display device. Since 1968 there have been a number of papers published which present various solutions to the hidden surface problem. A paper by Sutherland, Sproull and Schumaker[6] characterized the many different approaches taken to solve the hidden surface problem. This characterization is an authoritative and highly regarded piece of work and has resulted in a better understanding of the hidden surface problem. Consequently, new and improved algorithms have been developed.

Sutherland *et al.* show that there are three basic classes of approach taken to solve the hidden surface problem. The first class utilizes the three-dimensional co-ordinate data used to represent the model or object. The second class of solution acts upon the two-dimensional display data, and the third class uses data from each co-ordinate system. By studying the algorithms within each class, the authors conclude that there are four techniques which would be of assistance in designing an algorithm to solve the hidden surface problem.

Since a hidden surface algorithm will probably require a sort process, the first technique would be to utilize a sort function with an execution time that is linearly proportioned to the scene complexity. Secondly, the algorithm should perform display calculations to a limited accuracy, as the display scene is generated upon the display screen to a limited resolution. A third technique is to utilize coherence within the picture produced. For example, the algorithm by Romney *et al.* utilizes line-to-

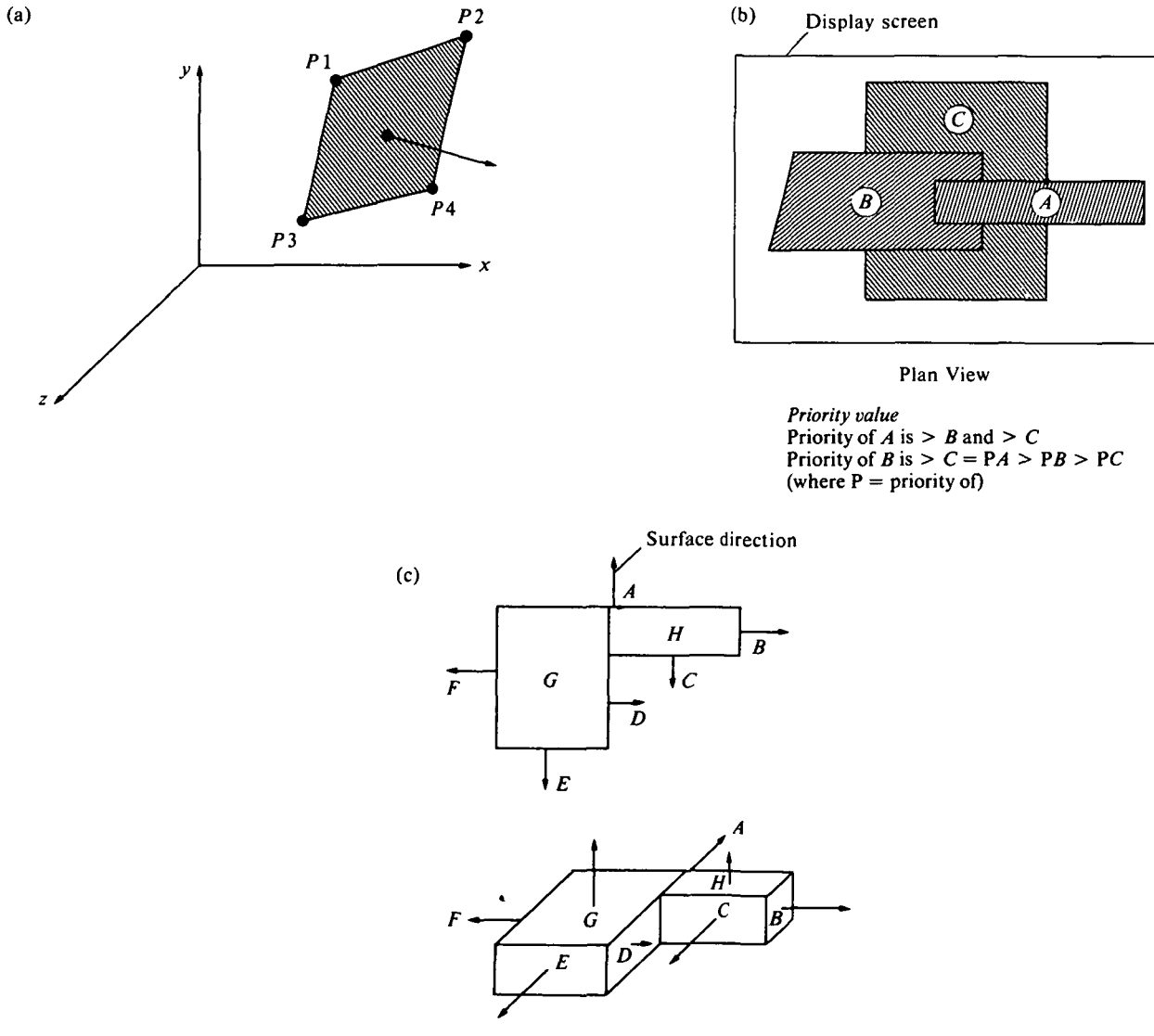


Figure 1. (a) An example of a surface; (b) the use of priority values; and (c) an object.

line coherence by the use of a list to store the order of the transition points along a scan line. The fourth technique which Sutherland *et al.* found to be useful is to capitalize on the layout of the model or object to be displayed.

The solution to the hidden surface problem adopted in the CGI system at the University of Sussex, uses the latter two techniques to enable a real-time algorithm to be implemented.

TECHNIQUES USED IN THE SUSSEX REAL-TIME SOLUTION

This section presents a novel solution to the hidden surface problem. The solution is designed for real-time removal of hidden surfaces and, therefore, caters for moving scenes which have a display rate of 25 times per second. In the 'Sussex Solution' emphasis is placed upon as much pre-processing of the model as possible before the real-time running of the system. The pre-processing enables hidden surface removal functions required during a simulation run to be minimized.

Before the Sussex Solution can be described it is necessary to define the following three terms:

(1) **Surface**. A quadrilateral plane in three-dimensional space. Each surface has a colour and is one-sided. The front face of a surface is defined by the direction of a perpendicular vector.
(2) **Priority value**. A value associated with each surface to denote the display order of the surface on the screen. A surface with a higher priority value will be displayed on the screen in preference to a surface with a lower priority value.
(3) **Object**. One surface or a group of two or more surfaces which are usually joined together. Each surface is joined to another surface by its edge.

Figure 1 shows an example of a surface, the use of priority values and an object. The above definitions restrict a model to be formed from planar quadrilateral surfaces.

The purpose of the pre-processing of the model is to enable absolute priority values to be assigned to surfaces being displayed in real-time. If objects within the model are 'well-spaced', then it is possible to use distance, between any point within an object to the viewer, as an unambiguous measure of absolute priority. Thus if two objects, *A* and *B*, are well-spaced then, for any given viewpoint, if *A* obscures *B* then every point within *A* is nearer the viewer than any point within *B*. Hence a measure of distance can be used to give unambiguous priority values as shown in Fig. 2.

Willis has shown that restricting objects to be well-spaced severely limits the layout of the model.[7] However, since objects are formed from surfaces, then it is adequate to ensure that surfaces are well-spaced, providing that the absolute priority value for each surface is based upon a measure of distance between any point on the surface and the viewer. Forcing surfaces to be well-spaced does not restrict the layout of a model so much as forcing objects to be well-spaced.

Willis has proved that a test of well-spaced surfaces in a model can be performed by considering what is called surface boundary limits.[7] Briefly, each surface of an object has a surface boundary which is calculated mathematically using proximity techniques. A surface

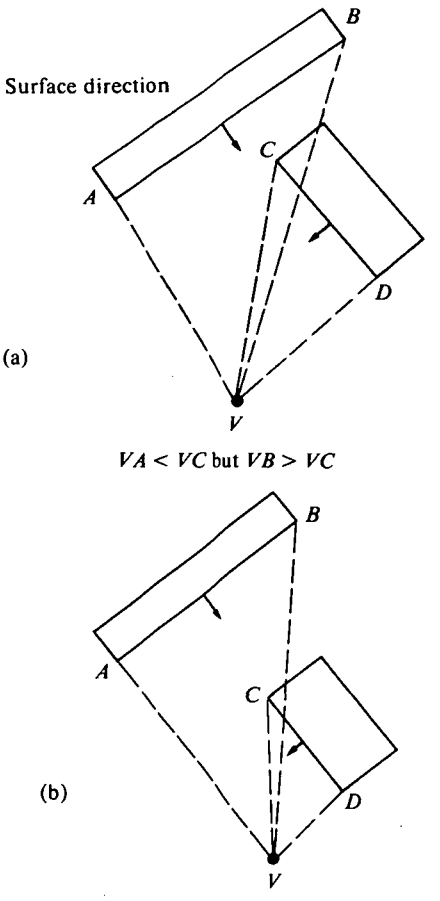


Figure 2. (a) Distance as an ambiguous measure of priority (closely-spaced objects); (b) distance as an unambiguous measure of priority (well-spaced objects).

boundary represents a volume around a surface which, if it contains another surface, then a viewing position exists that causes ambiguous priority assignment. During the pre-processing of the model, if a surface boundary contains a surface from another object, then the surface associated with the boundary is divided into two surfaces which reduces the volume of the boundary. Testing of surface boundaries continues until all the surfaces within the model are well-spaced.

However, this technique does not take account of related surfaces within the model, that is, the surfaces used to form objects. These surfaces are closely-spaced so they cannot be assigned a priority value based upon a measure of distance, but by careful examination of various properties of objects, this problem can be overcome.

For scenes representing terrain surfaces, there are two categories of surfaces or objects. The first class contains all the surfaces which have a *fixed* priority regardless of viewing position. For example, the ground plane of a model will always have the lowest priority, similarly surfaces laying directly on top of the ground plane will have the next higher priority level, and so on.

For the second class of surfaces no such fixed priority values can be assigned, as the surfaces form objects that represent buildings (or more generally, three-dimensional objects). A simple technique is now presented whereby surfaces forming an object can be assigned *relative*

priority values, which can be used in real-time to generate absolute priority levels.

ASSIGNING RELATIVE PRIORITY VALUE

If a group of surfaces that form an object can be given relative priority values then all the other surfaces can be assigned absolute priority values, once the absolute priority value from any one surface of the group is known, by consideration of their relative priority values. These relative priority values hold only within the object and the assignment holds for any number of surface groups (objects), providing that they are well-spaced. Hence, a scheme is required to test the structure of an object to ensure that the surfaces can be given a relative priority value. Cases exist where surfaces are positioned such that relative priority values cannot be found, as will be shown.

A paper by Schumacker *et al.* considers the problem of ambiguous priority assignment.[8] In the paper it is stated that it is possible to assign priority values to a group of surfaces providing that, for a chosen viewpoint, a 'cyclic obscuration' does not exist. An example of a cyclic obscuration is given in Fig. 3, and is where surface *A* obscures surface *B*, surface *B* obscures surface *C* and surface *C* obscures surface *A*. The conclusion reached by the paper proves that cyclic obscuration does not exist if the surfaces in the model can be separated by two planes or by a star configuration of three planes. This conclusion can be condensed by saying that every subset of three surfaces must be separable by a star configuration. If a set of surfaces fail the star configuration test then there exists a viewpoint such that the view seen contains a cyclic obscuration.

This is interesting but, in order to give relative priority values to a group of surfaces, it is required that a test is devised to check that for *any viewpoint* the priority values of the surfaces within the group remain fixed. This is a different and very much more difficult task. The test developed by the author uses what is termed a Priority Matrix.

The rules for creating a Priority Matrix are defined as follows:

Let *Si*, ($i := 1$ to n) be the surface within an object which is constructed from n surfaces.

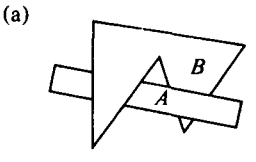
The algorithm to construct the Priority Matrix is:

```
For i := 1 to n;
  For j := 1 to n
    If i = j then Eij := 0
      else if Si > Sj then Eij := 1
        else Eij := 0;
  end;
end;
```

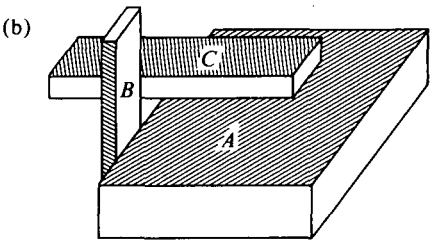
Where *Eij* is the entry in the Priority Matrix for column *i*, row *j* and > denotes 'obscures' (i.e. $Si > Sj$ means surface *i* obscures surface *j*).

For such Priority Matrix, it is possible to observe that the rows and columns identify the following. (1) The entries in a column identify the surfaces that obscure the surface represented by that column. (2) The entries in a row identify the surfaces obscured by the surface for that row.

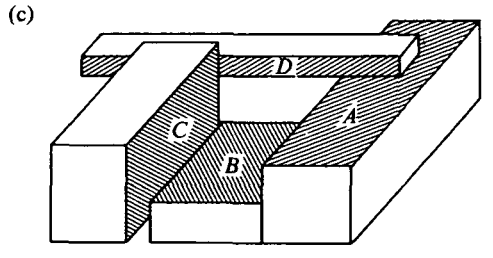
These two observations enable two important properties of the matrix to be found: (1) A column of zeros denotes 'nothing obscures the surface for that column'. (2) A row of zeros denotes 'the surface for that row obscures nothing'.



A obscures *B*
B obscures *A*
—Resolve by all surfaces quadrilateral with no concavities (i.e. External angles of edges 180°)



Surface *A* obscures surface *B*
Surface *B* obscures surface *C*
Surface *C* obscures surface *A*
—Resolve by star configuration test



Surface *A* obscures surface *B*
Surface *B* obscures surface *C*
Surface *C* obscures surface *D*
Surface *D* obscures surface *A*
—Resolve by considering subsets of three surfaces. (This example fails because of *ACD* not being divisible by star configuration.)

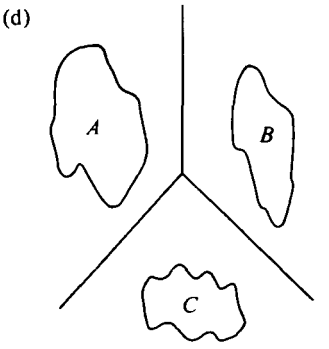
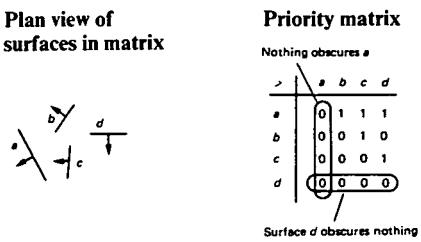


Figure 3. (a) Cyclic obscuration for two surfaces; (b) cyclic obscuration for three surfaces; (c) cyclic obscuration for four surfaces; and (d) star configuration to test for cyclic obscuration between three objects.

For example, these properties exist in the group of surfaces below:



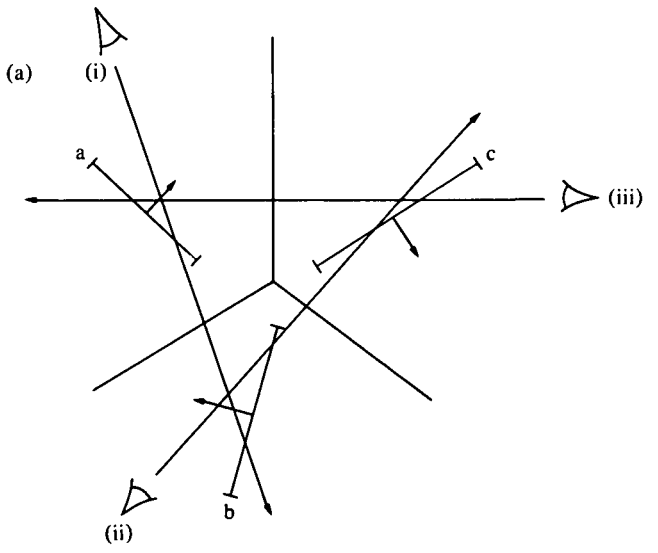
Before continuing, the term 'circuit' is defined as a collection of surfaces that cannot have priority values which are independent of viewpoint. Thus, for the surfaces forming a circuit, it is not possible to assign relative priority values since the priority values are dependent on viewpoint. Figure 4 contains three surfaces that form a circuit. If an object does contain a circuit, then the collection of surfaces involved in the circuit cannot be given relative priority values and, the object will require some form of adjustment.

It is possible to check for circuits during the production of a priority order from the Priority Matrix. The priority order is a list of the surfaces, with the surface given the highest priority at the head of the list and the surface given the lowest priority at the tail of the list. This represents a partial ordering problem and an algorithm to obtain a priority list from a Priority Matrix is given in Fig. 5.

The operation of the algorithm is to construct an ordered list of surfaces by considering the two properties of the matrix described earlier. Columns that contain all zeros denote that the associated surface for that column is not obscured by anything. Hence the highest priority value can be assigned to such a surface. Since one surface has now been given a priority value and been placed in the priority list, the surface is removed from the matrix by the action of deleting an appropriate row and column. The priority values for the surfaces remaining is, therefore, dependent on a reduced matrix which does not consider the obscuring effects of the surface within the priority list. The formation of the priority list continues until no surfaces are left in the matrix. This algorithm first assembles the priority list in head first order, which is a result of initially searching for columns with zeros, but the algorithm works equally well if the priority list is first assembled in tail first order, in which case rows full of zeros are initially searched for.

If the matrix is not empty but a column of zeros cannot be found, then this means that no surface can be placed

Surface group 'passes' star configuration

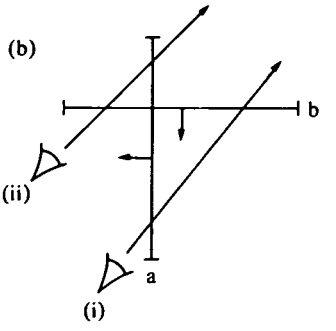


a obscures b, b obscures c, c obscures a

Priority matrix for surface group

>	a	b	c
a	0	$1^{(i)}$	0
b	0	0	$1^{(ii)}$
c	$1^{(iii)}$	0	0

If there is no row or column with zeros then a circuit exists. Therefore the surfaces *cannot* be assigned relative priority values.



Note: this is *not* antisymmetric (because of excess 1) then circuit of 2 surfaces exists.

Figure 4. (a) Three surfaces and (b) two surfaces forming a circuit.

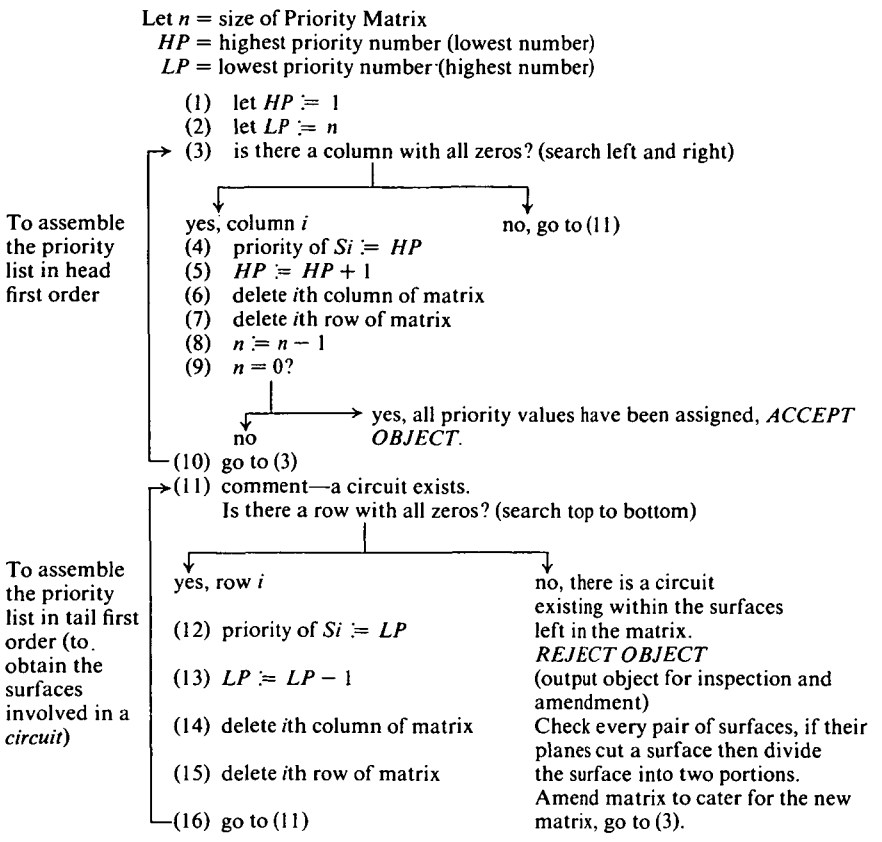


Figure 5. Matrix reduction algorithm.

in the head of the list. Thus a circuit of some kind *must* exist. A circuit can be more closely identified by constructing the priority list in a tail first order, as in the second half of the algorithm. Once the matrix is reduced to a stage where it contains no columns or rows filled with zeros, then the circuit is identified. However, it is only possible to state that at least one circuit exists between the surfaces left in the matrix. The exact nature of the circuit(s) present can only be determined by consideration of the elements left in the matrix.

An example where a circuit exists

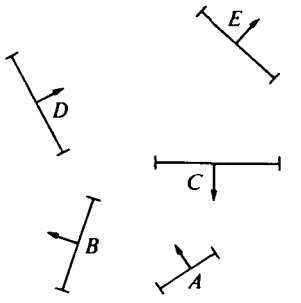
Figure 6 shows a plan view of a group of five surfaces. Normally, the surfaces will collectively form an object, but this group of surfaces is for demonstration purposes only. The Priority Matrix is shown for this surface group.

The first stage of the algorithm is to look for columns with zeros. The column found proves that nothing obscures surface *E*. Thus surface *E* is placed at the head of the priority list. The row and column for surface *E* are deleted which results in a reduced matrix where no columns of zeros are found. At this stage it can be concluded that a circuit exists but the circuit does not include surface *E*.

The next stage of the algorithm looks for rows of zeros. The row for surface *A* contains zeros which is interpreted as meaning that surface *A* obscures nothing. Therefore, surface *A* is placed at the tail of the priority list. The empty elements in the list, between *E* and *A*, must equal in number the surfaces left in the matrix. The row and column for surface *A* are deleted. The matrix remaining contains no row or column with zeros and no decision can be reached as to the priority order of the surfaces left in the matrix. The surfaces left in the matrix, therefore, form a circuit.

The matrix remaining has certain properties. If the matrix is anti-symmetric then this implies that there are no circuits consisting of only two surfaces. An example of a circuit of two surfaces was given in Fig. 4 and because the two surfaces obscure each other, the matrix cannot be anti-symmetric. Due to the obscuration of the surfaces causing a '1' to be entered in the matrix, it can be generally stated that a circuit of two surfaces exists if the matrix is not anti-symmetric because of an extra '1'. The converse is also true, if the matrix is not anti-symmetric because of an extra '0', then no circuit of two surfaces exists, but then, no circuit, *or* a circuit of three or more surfaces, may exist.

When a circuit is detected, the algorithm terminates. There are two distinct types of circuits. Identification of the type of circuit is performed by finding the location of the intersection line of the planes associated with every pair of surfaces involved in the circuit. The first type of circuit is unresolvable and is the case in which no intersection line falls within an area bounded by a surface belonging to the circuit. The second type of circuit can be resolved and is identified by the case in which an intersection line cuts a surface in the circuit. This type of circuit is broken by dividing the surface cut by the intersection line into two separate surfaces. This action creates an extra surface but allows the algorithm to continue. An example of surfaces forming a circuit which



Priority matrix for surface group

>	A	B	C	D	E
A	0	0	0	0	0
B	1	0	1	0	0
C	1	0	0	1	0
D	1	1	0	0	0
E	1	0	0	1	0

Priority matrix (stage 1)

>	A	B	C	D	E
A	0	0	0	0	0
B	1	0	1	0	0
C	1	0	0	1	0
D	1	1	0	0	0
~~E~~	~~1~~	~~0~~	~~0~~	~~1~~	~~0~~

Column of zeros ⇒ nothing obscures E

Priority matrix (stage 2)

>	~~A~~	B	C	D
~~A~~	~~0~~	~~0~~	~~0~~	~~0~~
B	1	0	1	0
C	1	0	0	1
D	1	1	0	0

Row of zeros ⇒ A obscures nothing

Priority matrix remaining

>	B	C	D
B	0	1	0
C	0	0	1
D	1	0	0

B obscures C
C obscures D } circuit found
D obscures B

Figure 6. Finding a circuit.

can be broken is shown in Fig. 7(a). In this figure, surface B is divided into surfaces B_1 and B_2 by the plane of surface A. The plane of surface C also cuts surface B and so B could have been divided differently. Surface B only needs to be divided into two portions by the plane of A *or* the plane of C to enable the circuit involving the three surfaces to be broken.

Both types of circuits can be resolved by allowing the surfaces contained within the circuit to have two relative priority values. Which priority value is chosen depends upon the location of the viewer. An example of this is shown in Fig. 7(b). Here the surfaces have two priority lists. Which list used depends upon which side of surface B is being viewed. Since the circuit resolving method is viewpoint dependent, the decision concerning which relative priority list to use must be made during real-time processing. It therefore involves extra processing during a simulation run which is undesirable. Extremely complex objects, formed from a large number of surfaces, may contain many circuits and so displaying the object may necessitate a great deal of decision making. Therefore, it is proposed not to use this circuit resolving method unless no other solution can be used. Other solutions preferred are to use the method which divides a surface into two portions, generating another surface, or to reconstruct the object.

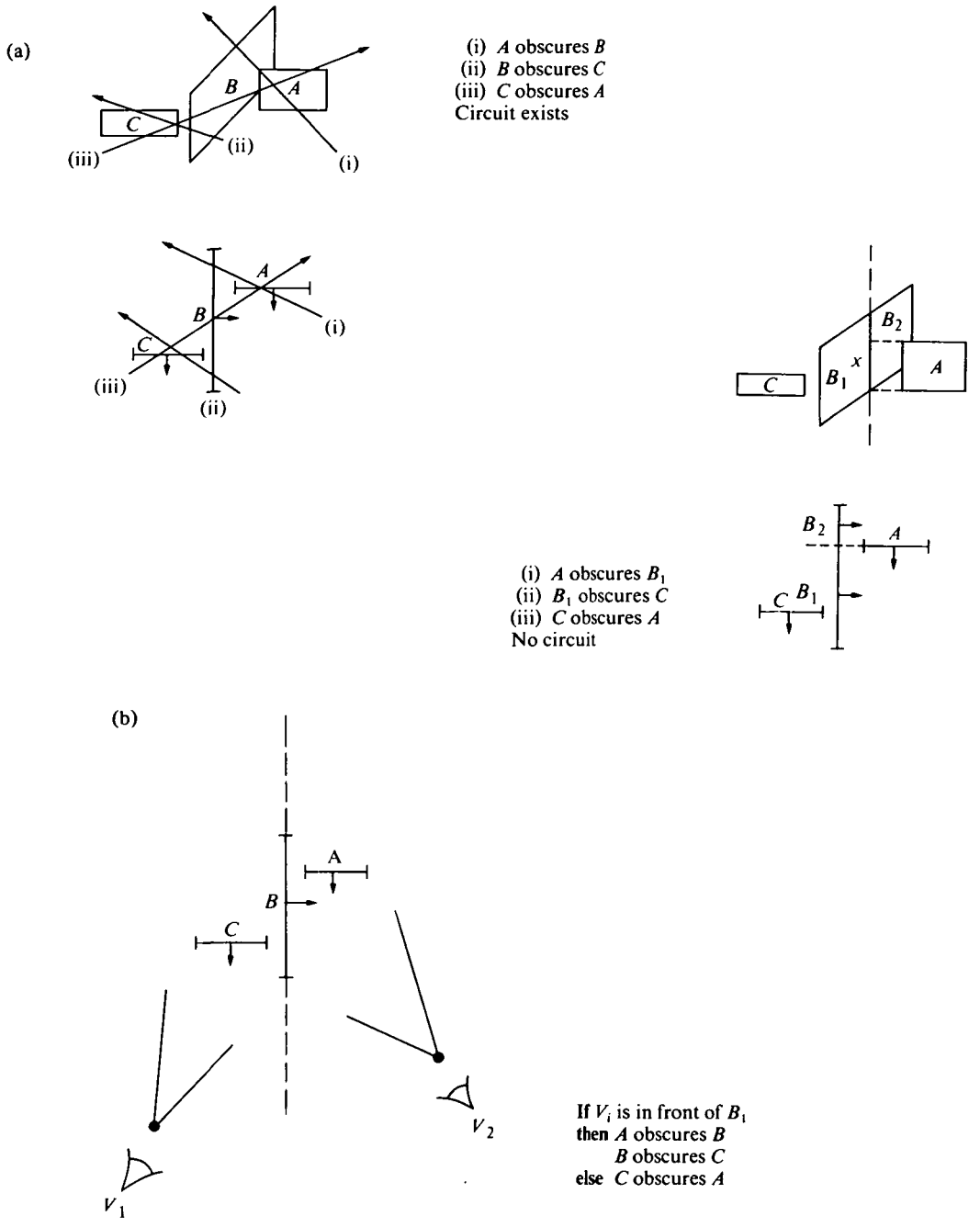


Figure 7. Resolving a situation where a circuit exists. (a) *Method 1*. Surface B is divided into two surfaces B_1 and B_2. The cut line x is the line of intersection of the planes of surfaces A and B. Priority list = B_1, C, A, B_2; (b) *Method 2:* Priority list is dependent on viewpoint, i.e. priority list for V_1 = C, A, B or C, B, A, for V_2 = A, B, C.

An example where a circuit does not exist

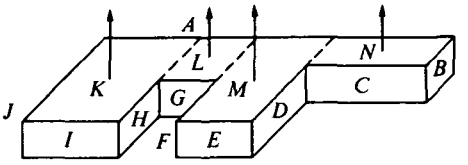
Figure 8 shows an object which is constructed from 14 surfaces. This is typical of the type of object the three-dimensional model will contain. The top of the object is formed from four separate surfaces but since the four surfaces are in the same plane which, in this example, is not cut by any other surface, the four surfaces could be treated as one.

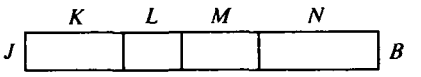
The completed matrix, before being processed by the algorithm, contains a number of columns, all full with zeros. The algorithm in its present form searches for a single column full of zeros. The example given in Fig. 8 assumes that the algorithm searches from left to right across the columns of the matrix. The first column of zeros found, therefore, corresponds to surface A. Surface A is then placed at the head of the priority list and the row and column for surface A deleted from the matrix. Searching for a column of zeros in the reduced matrix is then performed, again on a left to right basis.

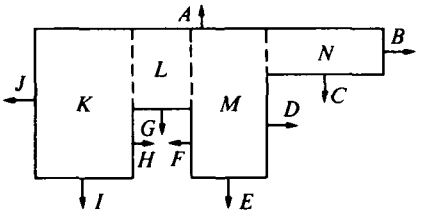
Processing of the matrix by the algorithm continues as shown in the figure and the complete priority list is formed. The object is, therefore, accepted by the algorithm. A perspective picture of the object with assigned priority values is given in Fig. 8. From this picture it can be seen that for every possible viewpoint, the priority values assigned to the surfaces are always correct.

The complete priority list in Fig. 8 requires that there are at least 14 priority values available. The priority value of some of the surfaces in the object, relative to other surfaces, does not matter if the surfaces are not involved in an obscuration. For example, the surface *A* does not obscure surface *M*, and surface *M* does not obscure surface *A*, they can have the same priority value.

This can be incorporated into the algorithm by searching for all columns full of zeros before reducing the matrix. The surfaces associated with the columns found can be assigned the same priority value because each surface, by definition of a column full of zeros, is obscured by nothing. An example of this method of searching for columns of zeros is shown in Fig. 9. The initial matrix contains columns full with zeros for surfaces, *A*, *B*, *E*, *I*, *J*, *K*, *L*, *M* and *N*. All these surfaces are assigned the highest priority value. The rows and columns for these







Priority matrix (stage 1 and 2)

>	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	0	0	0	1	0	1	0	1	0	0	0	0	0	0
B	0	0	0	1	0	0	0	1	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	1	1	0	0	0	0	0	0
E	0	0	1	0	0	0	1	1	0	0	0	0	0	0
F	0	0	1	0	0	0	0	0	0	0	0	0	0	0
G	0	0	1	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	1	0	0	1	1	0	0	0	0	0	0	0
J	0	0	1	0	0	1	1	0	0	0	0	0	0	0
K	0	0	1	0	0	1	1	0	0	0	0	0	0	0
L	0	0	1	0	0	1	0	1	0	0	0	0	0	0
M	0	0	1	0	0	0	1	1	0	0	0	0	0	0
N	0	0	0	1	0	0	0	1	0	0	0	0	0	0

Stage 1: priority list = A
Stage 2: priority list = A, B

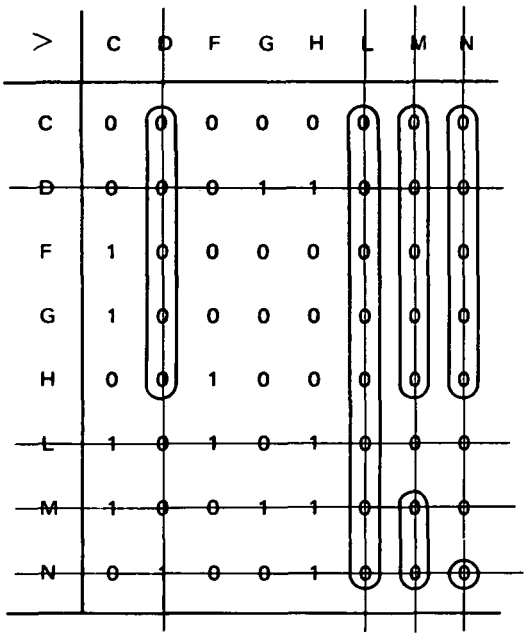
Priority matrix (stage 2, 3, 4, 5 and 6)

>	C	D	E	F	G	H	I	J	K	L	M	N
C	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	1	1	0	0	0	0	0	0
E	1	0	0	0	1	1	0	0	0	0	0	0
F	1	0	0	0	0	0	0	0	0	0	0	0
G	1	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	1	0	0	0	0	0	0	0	0
I	1	0	0	1	1	0	0	0	0	0	0	0
J	1	0	0	1	1	0	0	0	0	0	0	0
K	1	0	0	1	1	0	0	0	0	0	0	0
L	1	0	0	1	0	1	0	0	0	0	0	0
M	1	0	0	0	1	1	0	0	0	0	0	0
N	0	1	0	0	0	1	0	0	0	0	0	0

Stage 3: priority list = A, B, E
Stage 4: priority list = A, B, E, I
Stage 5: priority list = A, B, E, I, J
Stage 6: priority list = A, B, E, I, J, K

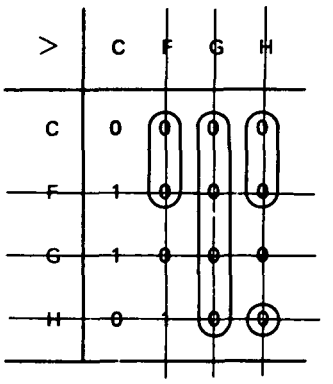
Figure 8. Assigning surface priorities.

Priority matrix (stage 7, 8, 9 and 10)



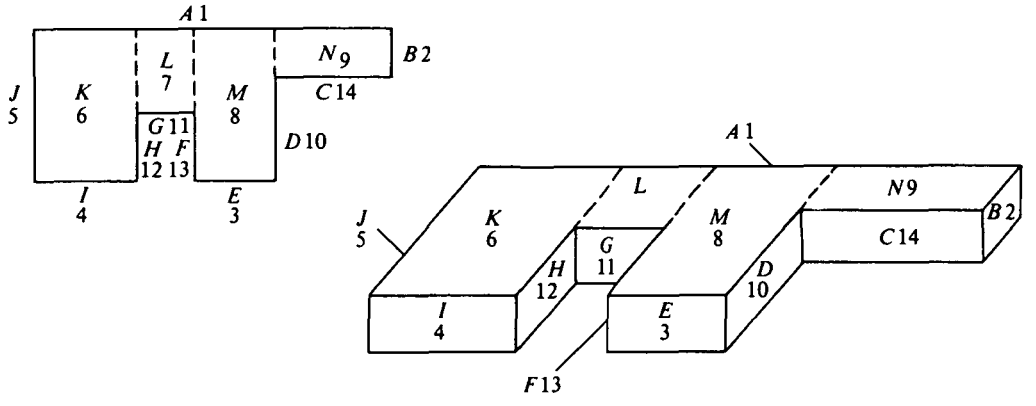
Stage 7: priority list = A, B, E, I, J, K, L
Stage 8: priority list = A, B, E, I, J, K, L, M
Stage 9: priority list = A, B, E, I, J, K, L, M, N
Stage 10: priority list = A, B, E, I, J, K, L, M, N, D

Priority matrix (stage 11, 12, 13, 14)



Stage 11: priority list = A, B, E, I, J, K, L, M, N, D, G
Stage 12: priority list = A, B, E, I, J, K, L, M, N, D, G, H
Stage 13: priority list = A, B, E, I, J, K, L, M, N, D, G, H, F
Stage 14: priority list = A, B, E, I, J, K, L, M, N, D, G, H, F, C

Complete priority list =	A	B	E	I	J	K	L	M	N	D	G	H	F	C
Priority value =	1	2	3	4	5	6	7	8	9	10	11	12	13	14



surfaces are deleted from the matrix which results in a very much reduced matrix containing five surfaces. This time, surfaces *D* and *F* have associated columns full of zeros. Surfaces *D* and *F* are, therefore, assigned the same priority value and the matrix reduced by deleting the rows and columns of these surfaces.

The algorithm continues and the completed priority list, together with a picture of the object is given in Fig. 9. The resultant priority list performs the same task as the list in Fig. 8, but only four priority values are required. Thus, a significant saving in the number of priority values required to display a scene consisting of many objects is achieved.

A second example of finding relative priority values for the surfaces forming an object is given in Fig. 10. This figure shows an object which contains surfaces placed on top of other surfaces to represent doors and windows. This is the type of object that is best suited to be used with the developed algorithm. The technique of placing surfaces on top of other surfaces enables objects to look realistic and so improves the appearance of the complete model.

Priority matrix (stage 1)

>	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	0	0	0	1	0	1	0	1	0	0	0	0	0	0
B	0	0	0	1	0	0	0	1	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	1	1	0	0	0	0	0	0
E	0	0	1	0	0	0	1	1	0	0	0	0	0	0
F	0	0	1	0	0	0	0	0	0	0	0	0	0	0
G	0	0	1	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	1	0	0	1	1	0	0	0	0	0	0	0
J	0	0	1	0	0	1	1	0	0	0	0	0	0	0
K	0	0	1	0	0	1	1	0	0	0	0	0	0	0
L	0	0	1	0	0	1	0	1	0	0	0	0	0	0
M	0	0	1	0	0	0	1	1	0	0	0	0	0	0
N	0	0	0	1	0	0	0	1	0	0	0	0	0	0

Stage 1: priority list = A, B, E, I, J, K, L, M, N//

Priority matrix (stage 2)

>	C	D	F	G	H
C	0	0	0	0	0
D	0	0	0	1	1
F	1	0	0	0	0
G	1	0	0	0	0
H	0	0	0	0	0

Stage 2: priority list = A, B, E, I, J, K, L, M, N // D, F //

Priority matrix (stage 3)

>	C	G	H
C	0	0	0
G	1	0	0
H	0	0	0

Stage 3: priority list = A, B, E, I, J, K, L, M, N // D, F // G, H

Complete priority list = A, B, E, I, J, K, L, M, N // D, F // G, H // C

Priority value = 1 2 3 4

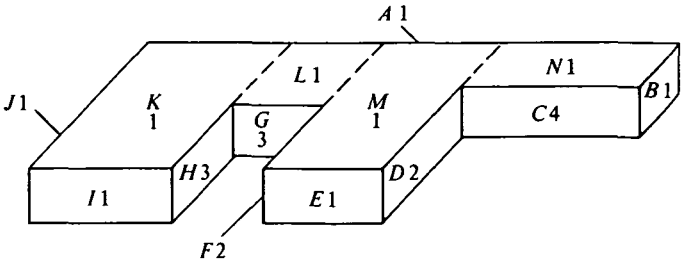


Figure 9. Compressing the number of priority values.

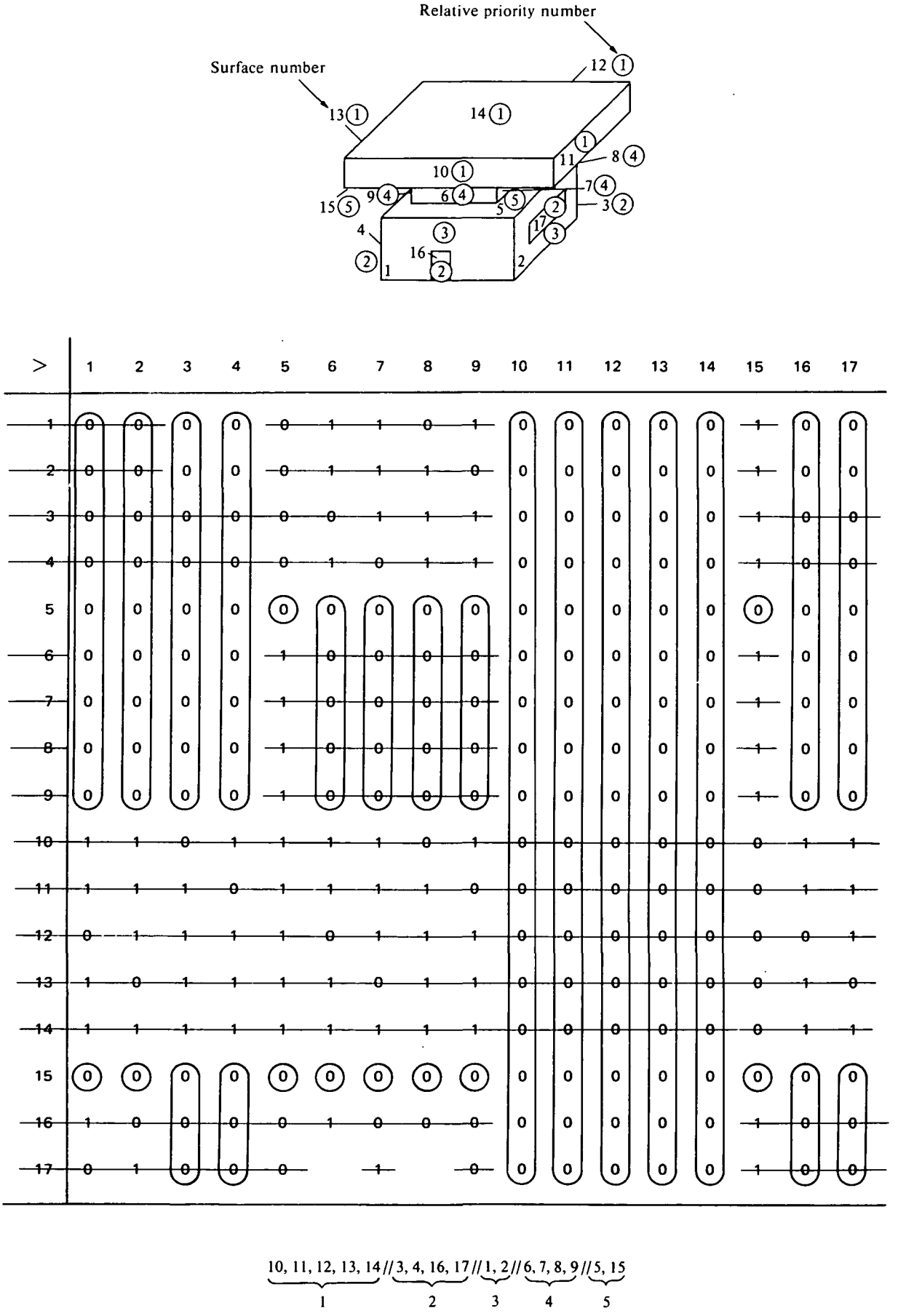


Figure 10. Another example of finding relative priority values for the surfaces forming an object.

SUMMARY

A simple technique has been described which enables *relative* priority values of surfaces within objects to be generated. This technique is designed to be used off-line, so that during real-time only the priority of complete objects need be computed. From the priority values for complete objects, the *absolute* priority values for the individual surfaces are easily found.

Many different techniques can be used to find the priority values for complete objects; the excellent proximity techniques devised by Willis are to be used in the Sussex CGI System.[7] In fact any technique which produces object priority values in real-time can be used with the techniques described in this paper.

Another approach for finding the object priority values is to use the separating planes concept as detailed by Schumacker *et al.*[8] The separating planes technique simply divides the model into 'areas' which are separated by a separating plane. The location of objects within the model are thus held in a binary tree structure. Traversal of this tree in real-time according to viewing position yields the ordering of each area enclosed by separating planes as a function of distance, and hence the ordering directly reflects priority values of the areas. Providing only one object is held within one bounded area, then object priority values *are* the area priority values.

REFERENCES

1. I. E. Sutherland, Computer inputs and outputs. *Scientific American* (No. 9) (1966).
2. I. E. Sutherland, Computer displays. *Scientific American* **222** (No. 6) (1970).
3. C. B. Jones, A new approach to the hidden surface problem. *Computer Journal* **14** (No. 3), 232–237 (1971).
4. J. E. Warnock, A hidden surface algorithm for computer generated half-tone pictures. University of Utah. Tech. Rep. RADC-TR-69-249 (1969).
5. G. S. Rowney, G. S. Watkins and D. C. Evans, Real-time display of computer generated half-tone perspective pictures. IFIP. Vol. 2. 1968. pp. 973–978 (1968).
6. I. E. Sutherland, R. F. Sproull and R. A. Schumacker, A characterization of ten hidden surface algorithms. *ACM Computing Surveys* **6** (No. 1) March (1974).
7. P. J. Willis, A real-time hidden surface technique. *Computer Journal* **20** (No. 4), 335–339 (1978).
8. R. Schumacker, B. Brand, M. Gilliland and W. Sharp, Study for applying computer-generated images to visual simulation. General Electric Co., Tech. Rep. AFHRL-TR-69-14 (1969).

Received July 1981