ANALYSIS OF A HIDDEN-LINE ALGORITHM

By

CHARLES WAYNE BROWN

Bachelor of Science

University of Texas at Arlington

Arlington, Texas

1977

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
MASTER OF SCIENCE
December, 1980

ANALYSIS OF A HIDDEN-LINE

ALGORITHM

Thesis Approved:

J P Chandler
Thesis Adviser

J K. Phillips

Donald D Fisher

Norman N Durham
Dean of the Graduate College

PREFACE

The goal for this research was to develop an efficient hidden-line algorithm for a small computer system. A hidden-line algorithm published by J. G. Griffith was used as a basis for the research. The algorithm was successfully implemented on a mini-computer and extensive analysis and testing were done. This work showed that Griffith's algorithm was a linear growth algorithm as compared to the complexity of the picture environment. Several enhancements were added to the original algorithm to achieve even greater efficiency.

Acknowledgements and thanks go to Mr. John Houston of Graphics Construction, Inc. in Tulsa, Oklahoma, who presented the initial idea for this research and provided the computer time and facilities used in the algorithm implementation and testing. My thanks also go to Mr. Houston for his valuable suggestions during the entire project. I am appreciative of Dr. J. P. Chandler who, as my thesis adviser, was always available for help on problems and who guided the direction of the work. I wish to thank Dr. D. Fisher and Dr. R. Phillips for their suggestions on the final draft.

I am very thankful for the continual love and encouragement of my parents, and especially my mother for typing the first rough draft of this paper. My thanks go to all my friends who encouraged me throughout the entire project, and especially for the support of my dear friends, Stacy and Paula Rinehart, who provided me with living quarters during

my final month of research. I am very appreciative of Denise Bower, who did such an excellent job of typing the final draft and for her extra work involved in the final approval.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Since the beginning developments in computer graphics capabilities, the problems of representing three dimensional objects in a two dimensional picture have been studied in great detail. Initially, in the early sixties, only the hardware capability for drawing lines existed. Faces, therefore, were represented as closed loops or "circuits" of straight lines. Representing objects as pictures in this form, often called "wire frame" drawings, can cause confusion and even optical illusions if all lines of an object are presented. The need for an algorithm to eliminate all lines not "visible" from a particular view of an object becomes quite apparent. This is the hidden-line problem. (A version of the hidden-line problem for "ruled surfaces" is a relatively simple problem and is not discussed in this thesis

Beginning in the middle sixties and moving into the early seventies, the hardware capabilities for graphics greatly improved to make shaded drawings possible. These advances greatly improved the visual quality of computer generated pictures but the problem then became how to eliminate nonvisible shaded faces. This is the hidden-surface problem.

In the beginning years of research, several solutions to the hidden-line problem were developed. However, when the hidden-surface problem

began to receive attention, the emphasis in research shifted almost entirely in its direction. Little new work has been published on the hidden-line problem since the nineteen sixties.[2] This is unfortunate because many areas of applications such as architecture and engineering have a need for efficient three-dimensional line drawings, especially in applications with mini and micro computers with only line drawing capabilities. A new algorithm for hidden-line elimination was published by J. G. Griffith (6) which has possible applications in these areas.

The description of Griffith's algorithm in the article, "Eliminating Hidden Edges in Line Drawings" (6), states that this is a linear growth algorithm, which means that the computer time required for a drawing increases in a linear rate as the complexity of the object increases. (Almost all previous hidden-line algorithms have a "squared-law" growth rate. Given a drawing with N objects, the computer time required for hidden-line removal is proportional to N^2.) Further research into Griffith's method is needed to verify his results and to search for possible improvements to the algorithm that could enhance its efficiency. Also, some valid comparison with other previously existing algorithms must be made to establish Griffith's algorithm as a better or worse solution to the hidden-line problem. The contents of this thesis presents the results of the research done in these areas.

First, a brief history and overview of hidden-line, hidden-surface algorithms will be discussed for the reader's background information and later algorithm comparisons. Then, in Chapter III, a description

---

[2] J. G. Griffith has published several different papers on these problems in the seventies besides the one to be studied here. (See Bibliography.)

of Griffith's algorithm will be presented showing details of his data structure and method for hidden-line removal. In Chapter IV, various changes to improve Griffith's algorithm are discussed and the effects of these changes are examined. Comparisons are made between Griffith's algorithm and ten other hidden-line, hidden-surface algorithms. The final chapter discusses implemention of Griffith's algorithm, problems that were encountered, and suggestions for future work with Griffith's ideas.

CHAPTER II

HISTORY OF HIDDEN-LINE ALGORITHMS

Categorization

When discussing many different algorithms that solve the same relative problem, there is a need for an efficient means of comparison. By categorizing the algorithms, many insights into the hidden-line problem are made which might be hard to understand if each algorithm were studied separately. A very nice categorization of ten of the most prominent hidden-line, hidden-surface algorithms published before 1974 was presented by Sutherland, Sproull, and Shumacker (10). The following paragraphs are a description of their categorization scheme.

Four criteria are used as a basis for categorization and analysis:

1. First, a major difference is the resolution, or accuracy, of the final picture produced by each algorithm. An algorithm is said to work in <u>object space</u> (10, p. 19) if the final output is as accurate as the accuracy of the computer used. Almost all calculations are done to machine accuracy. However, if the output of a drawing is limited to a certain screen resolution size, there is no need for this kind of accuracy. Algorithms that calculate a drawing only to a limited resolution are said to work in <u>image space</u>.

2. The types of comparisons for hidden-line determinations are quite different for each method. Many "tricks" are used to eliminate as many unnecessary comparisons as possible. Obviously, if the complicated and time consuming tests for intersecting or overlapping lines can be reduced to only those cases that actually have intersections, then much efficiency can be gained.

3. Each algorithm takes advantage of one (or more) specific characteristics of the problem to attempt to minimize the solution process. These common relationships are given the name coherence. For example, Appel uses the fact that if a vertex on an edge is visible, then all other edges with this vertex will very likely be visible (hence, edge to edge coherence). By taking advantage of coherence relationships, computations can be reduced and, in some cases, eliminated.

4. Also, to analyze the efficiency of each algorithm, the various sorting and searching techniques are compared.

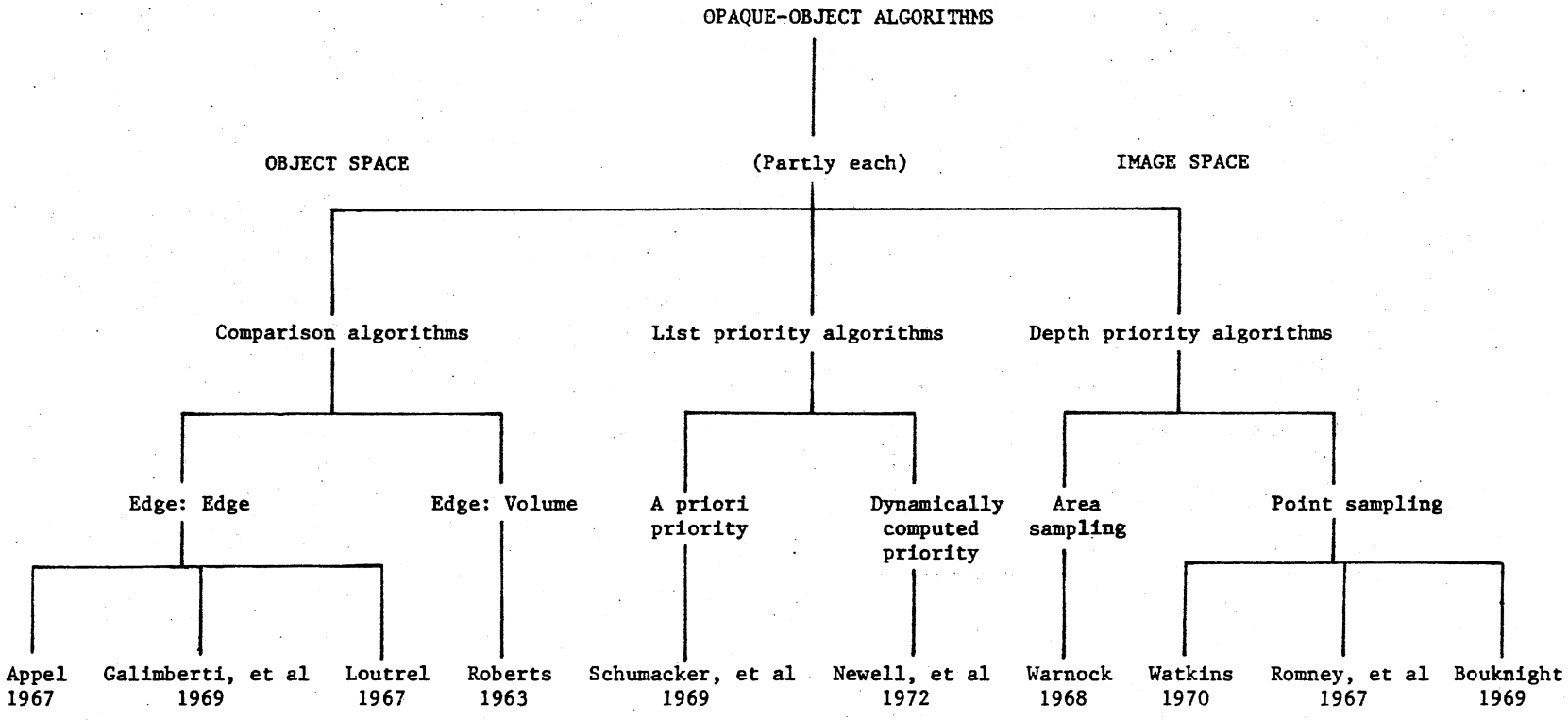
Referring to Figure 1 the categories for each algorithm can be seen. Appel, Galimberti, Loutrel, and Roberts solved the hidden-line problem in object space, to the nearest accuracy of the computer used. Warnock[1], Watkins, Romney, and Bouknight solved the hidden-surface problem in image space, because their goal was television type output which has a fixed resolution size. Schumacker and Newell also solved the hidden-surface problem but partly in both "spaces". Their calculations were done to machine accuracy but the shaded drawings produced must be output to the limited resolution of a television type display screen.[2]

Each group of algorithms uses a different method for finding visible line segments or faces. The object space algorithms use comparison tests to find intersecting lines and to determine which lines are visible. These tests require many mathematical calculations and are often quite time consuming. The Schumacker and Newell algorithms use a priority scheme to determine face visibility. The priorities can be calculated once for an object regardless of the position of the observer. By avoiding repetitious calculations for visibility, these algorithms can produce pictures in real-time; meaning that the time

---

[1]There are hidden-line versions of the Warnock algorithm.

[2]All above authors are cited in (10).



Source: Sutherland (10, p. 19)

Figure 1. Sutherland's Categorization of Ten Hidden-line, Hidden-surface Algorithms

taken for picture computation is less than the time required for a single scan of a raster screen output device (television type screen). The image space algorithms use depth priority schemes to determine visibility on a specified area of the screen. Extensive sorting is used to give faces priorities according to their depth, and their position in these priority lists help to determine visibility. It must be noted that all ten of the algorithms put restrictions on the kind of objects allowed in their picture environments. The methods used for visibility determination are valid only within these restrictions. These environment restrictions are noted in Sutherland's (10) article but are not used for categorization.

Now that a general overview has been established, a more detailed look at a few selected algorithms is needed. The first one will be Roberts' (9) algorithm, which was the first practical solution to the hidden-line problem. Next, a discussion of Appel's (1) method will illustrate edge by edge comparisons. (Galimberti's and Loutrel's methods are variations on this same theme.) Newell's (7) algorithm will help explain priority schemes in greater detail. And finally, Watkins' (11) algorithm will explain scan-line algorithms and depth priority.

Roberts' Algorithm

Roberts' (9) algorithm solves the hidden-line problem using linear algebra techniques to compare each edge in an environment with each spatial volume. A spatial volume is defined by a set of convex polygon faces. By restricting the shapes to convex polygons, each

face of a solid can be represented by a plane equation of the form $aX + bY + cZ + d = 0$. The mathematical relationships between points along an edge and these planes can determine the visibility of a given edge.

Roberts' algorithm can be broken into three distinct steps.

1. Clipping against the screen boundary.
2. Rejecting back edges.
3. Testing the edge against polygonal volumes.

Parts of the environment can be outside of the particular view of an observer. Those edges not in view (outside the screen boundary) or those portions of edges partially hidden by the screen boundary can be eliminated by multiplying an edge equation times a special volume matrix which represents the edges of the view boundary. The resulting parameters provide the maximum and minimum values which define the visible portion of an edge. Next, the back edges are eliminated. Any given solid volume will hide some of its own edges. The position of the observer determines which edges are automatically hidden. By determining the direction of a vector normal to the face of a solid, a face can be recognized as a front face or back face. Edges on back faces are eliminated from further testing; they are totally hidden. The remaining edges after these first two steps must now undergo much more complex tests to determine visibility.

Each edge is tested for visibility against every solid volume in the environment. The edge is represented parametrically as

$$v = s + t(r - s) \qquad 0 \leqslant t \leqslant 1$$

where r and s are the two endpoints. For every value of t along the edge, an imaginary vector is created which points in the direction of

the observer. If any of the vectors pass through a face, then the edge is totally or partially hidden. If no face intersections occur, then the edge is totally visible. To determine these conditions, the parametric equation defining the imaginary vectors is multiplied by a volume matrix which represents a particular volume. The resulting parameter values are tested against boundary values which satisfy the required conditions for visibility.

Roberts' solution to the hidden-line problem is often called the "classical" solution because of its use of mathematical relationships. Although his method is very good, its performance suffers because of the enormous number of calculations required in the matrix multiplications and because of the number of tests involved. Each edge is compared to every volume in the environment which makes the computer time required grow proportionally to N^2, where N is the number of objects in the environment. For complicated scenes, the algorithm is not practical.

Appel's Algorithm

Appel's (1) algorithm works in the same kind of environment as Roberts': that is, polyhedra made up of planar polygonal faces. But Appel's approach is totally different. The algorithm introduces the concepts of a material edge, a contour edge, and quantitative invisibility to define the edges in an environment. These properties of edges help determine visibility.

An edge has three possible classifications. Edges hidden by their own volumes are called back edges and they are immediately eliminated.

Edges bounding two possibly visible faces are referred to as material edges. Those edges bounding an invisible face and a potentially visible face are called contour edges. An edge is broken into segments based on its intersection with contour edges and each resulting edge segment is assigned a quantitative invisibility value. A visible edge segment has an initial value of zero. Each time the edge crosses behind a contour edge the quantitative invisibility is incremented by one and then correspondingly decremented when it comes out from behind a contour edge. Only those edge segments with a resulting quantitative invisibility of zero are visible.

The task of finding initial quantitative invisibility values is time consuming (an edge endpoint is compared to every face in the environment). However, for any given point, all edges emanating from this point have the same quantitative invisibility (normally). This coherence relationship reduces to a great extent the amount of required calculations. By following "circuits" through the drawing, previous ending quantitative invisibility values are passed on to other beginning edges. Appel developed an efficient method to examine every edge in an environment using a minimum number of these circuits. Problems arise, however, when several special cases can make the quantitative invisibility values wrong. For example, if a point lies on a contour edge, then some edges emanating from this point will possibly have a higher quantitative invisibility value than others, depending on which edges emanate behind the face and which edges emanate out away from the face. This case and others must be tested before quantitative invisibility

values at an edge endpoint can be passed on to other edges. These tests become quite detailed and inhibit the efficiency of the quantitative invisibility scheme.

The algorithm's efficiency is proportional to N^2. Loutrel, and Galimberti and Montanari used this same basic idea but with special enhancements. Their algorithms are also proportional to N^2. The problem of having to compare each edge in an environment with every other edge was still not solved.

Newell's Algorithm

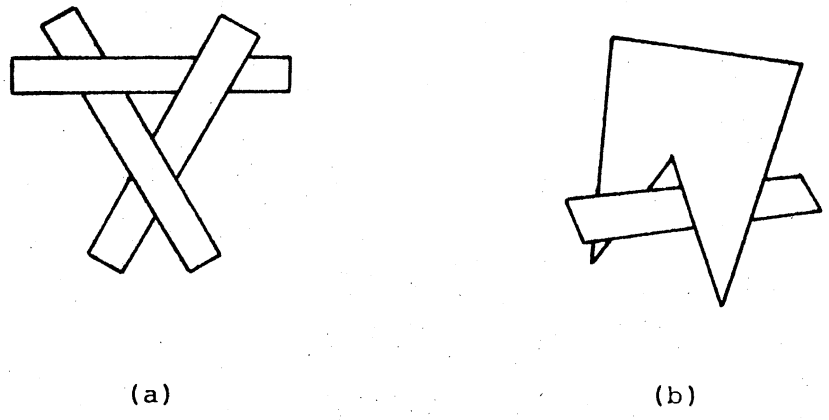
Newell's (7) algorithm solves the hidden-surface problem in what could be described as the "painter's" algorithm. All polygons in an environment are ordered according to their distance from the observer. After proper ordering, the faces are output or "painted" onto an output screen (or frame buffer) starting with the most distant face and proceeding up to the nearest face. The hidden-surface problem becomes a sorting problem to determine the correct order of output.

The algorithm is called a priority algorithm because each face is given a priority based on its distance from the observer; those nearer faces to the observer having higher priorities. Many different tests are used to assign priority values. The simpler tests are applied first, and if these fail, more complex tests are used until a priority can be determined. The initial step orders all faces according to their closest point to the observer. Using this order of faces, the following tests are done between adjacent faces. A face has higher priority over the next face if any of the following tests are true:

1. A depth minimax test shows that there is no overlap in depth.
2. An XY minimax test shows no overlap in X or Y.
3. All vertices of the face are nearer to the observer than the plane which contains the next face.
4. All vertices of the next face are farther away from the observer than the plane which contains the face.
5. A complete overlap test which shows no overlap in X or Y.

Once any of these tests are true, none of the others need to be applied.

Two problems arise which must be solved. First, the priority relationships are not transitive; that is, faces can obscure other faces which in turn might "cycle" back and obscure the original faces (Figure 2a). Second, because Newell allows concave faces, two faces can possibly obscure each other (Figure 2b). If a face in the priority list tries to shift priorities more than once, then one of the above problems is assumed to be true. The face is subdivided into two



(a) (b)

Figure 2. Special Cases for Newell's Priority Scheme

smaller faces and the priority tests are repeated. Subdivisions continue until no more than one priority shift per face is required for correct ordering.

Newell's algorithm is much more efficient than Roberts' or Appel's algorithms. The development of priority schemes made it feasible for real-time hidden-surface pictures. Schumacker developed the first real-time algorithm which has been operational since 1968. His algorithm used a clustered priority scheme and was implemented in hardware.

Watkins' Algorithm

Watkins' (11) algorithm solves the hidden-surface problem by the scanline approach, which is based on the output needed for a raster (television type) screen. Raster type screens have a limited number of possible dots (technically called pixels) in which to represent a picture. By taking scan lines horizontally across the screen, those pixels visible on that line can be determined, and giving each one a desired shade of gray will create a shaded picture. Obviously this algorithm works in the image space creating a picture only as accurate as the screen.

Two steps are involved in determining visible parts of a scan line. First, all intersections between the scan line and the polygon faces that it crosses are found. Each polygon face "owns" a segment of the scan line; that portion of the scan line between face intersection points. If a face scan line segment has no intersections with other face segments, then that face segment is visible and no other action is required. However, overlapping segments must be further tested to determine

visibility. The overlapping section is divided into "sample spans" which satisfy the condition that the visibility in each span does not change (that is, the XZ-plane projections of the faces corresponding with each span do not intersect.) The second step involves determining which face is visible in each sample span. Because only one face is visible in each plan by definition, then a simple Z depth analysis can determine visibility. After all scan lines are processed, the picture can be generated.

Watkins uses a sort on the Y coordinates to avoid as many comparisons as possible. By presorting the faces, a face is not tested for intersections with a scan line until it comes into range and intersections are possible. After all of a face has been examined, the face can be removed from the list of possible faces and never be compared again. All unnecessary intersection calculations are eliminated. Going one step further, scan line algorithms can make use of the coherence from one scan line to the next, and further reduce the number of calculations required.

Real-time pictures have been produced using Watkins' algorithm.

Summary

Many very different ideas and solutions to the hidden-line, hidden-surface problem have been implemented. All of the algorithms which have produced pictures in real-time are hidden-surface algorithms. None of the hidden-line algorithms have even come close to such speeds. The need for more efficient hidden-line algorithms, comparable to the hidden-surface algorithms, is evident.

CHAPTER III

THE GRIFFITH ALGORITHM

Introduction

Now that a brief summary of hidden-line algorithms has been presented, a detailed description of Griffith's algorithm is needed. Griffith used many of the same ideas as the previously discussed algorithms but his combination of these ideas and his introduction of several new ideas such as a "masking line" makes his algorithm distinctively different. Approximately one-half of the computer code required for the implementation of his algorithm is for the creation of the data structure, which will be discussed first. Then the hidden-line removal will be examined.

Data Structure

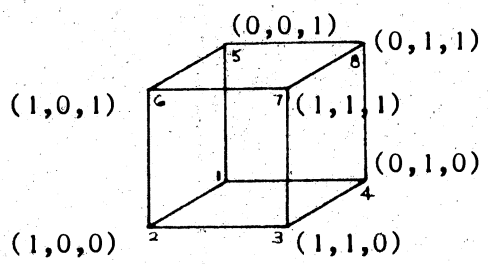
The Griffith (6) data structure is the single most important part of his algorithm for efficient hidden-line removal. To establish the final data structure as shown in Figures 5 and 6, the initial vertices must first go through transformation equations to create the desired view of the object. Then the vertex nodes, face nodes, and edge nodes are created. Finally the screen area is divided into a two-dimensional grid and each face is linked into every grid cell that it covers or intersects.

Initial input of data for an object uses the following form: the number of vertices, each vertex given by its X, Y, Z cartesian coordinates, the number of faces, and a list of face descriptions. A face description is a list of vertex pointers which point to adjacent vertices of the face perimeter followed by at least one zero to indicate the end of the list. The vertices are real numbers (having whole and fractional parts) and the vertex pointers are integers. Figure 3 is an example of input for a simple cube. Any polygon shaped face is allowed, convex or concave with the assumption that all edges are straight lines; no curves allowed. It is also assumed that all faces intersect only on edge boundaries.

The vertices must undergo two transformations to establish a desired view. The observer is assumed to be on the positive Z axis looking at the origin, with the positive X axis to the right and the positive Y axis upward[1] (Figure 4). The first transformation requires as many as three rotations, one around each axis to align the axes to this orientation. Given the three angles of rotation, which are part of the initial input, all three rotations are done at once by multiplying each vertex by a 3 x 3 rotation matrix. Then the maximum distance of the object from the origin is found. The object lies totally within a sphere of this radius. To create the final picture, the second transformation produces a perspective in a two-dimensional picture plane which is parallel to the X-Y plane and perpendicular to the surface of the sphere (Figure 4). This environment makes perspective generation

---

[1]Most algorithms have the observer on the negative Z axis as standard notation. Caution must be used when referring to other literature in discussing maximum and minimum values of Z.




```
8
 0.0000000     0.0000000     0.0000000
 1.0000000     0.0000000     0.0000000
 1.0000000     1.0000000     0.0000000
 0.0000000     1.0000000     0.0000000
 0.0000000     0.0000000     1.0000000
 1.0000000     0.0000000     1.0000000
 1.0000000     1.0000000     1.0000000
 0.0000000     1.0000000     1.0000000
6
1   2   3   4   0
1   5   6   2   0
6   7   3   2   0
7   8   4   3   0
4   8   5   1   0
5   8   7   6   0
```

Figure 3. Input Requirements

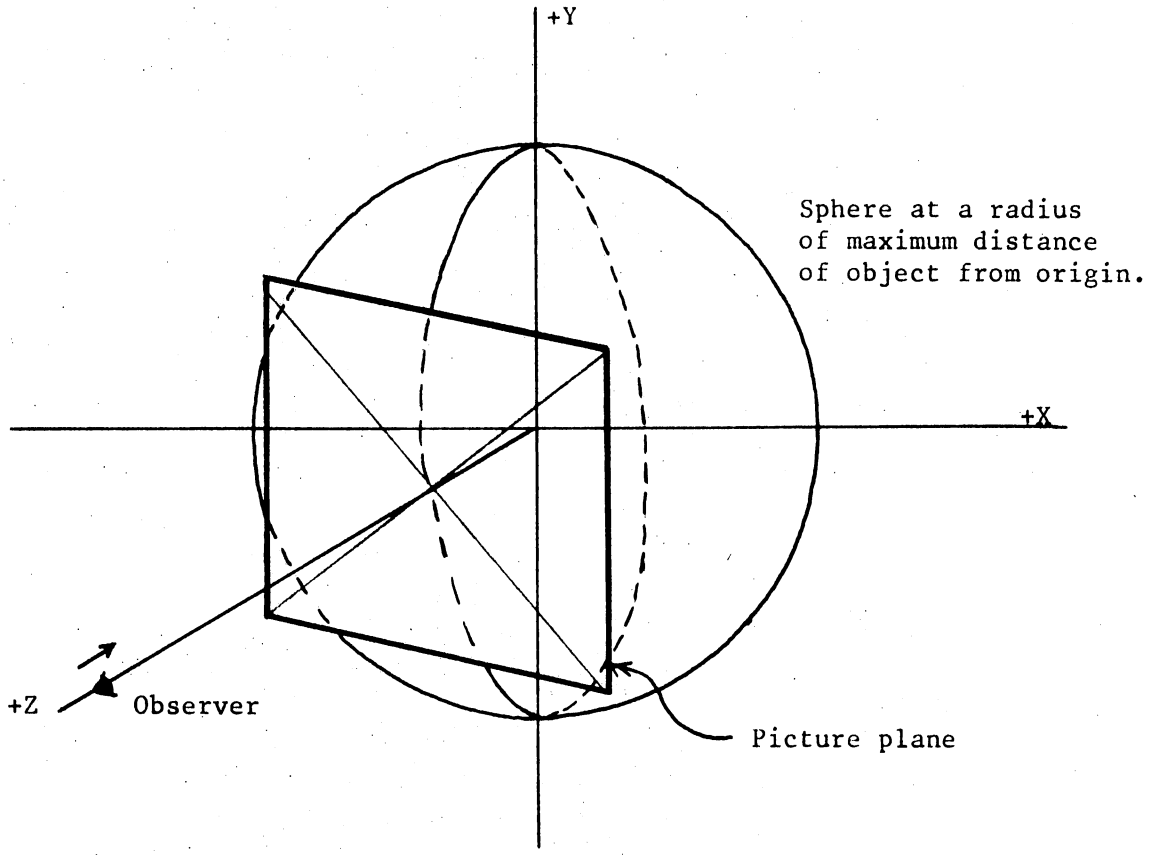


Figure 4. View of Object After Rotation and Location of the Picture Plane for Perspective

very straightforward because the entire object lies inside the sphere picture plane. If this were not true, then a clipping algorithm would be needed to "throw away" everything outside of the picture plane. Therefore, only exterior views of any object are possible; close up views or interior views would require clipping and a different method for perspective generation. (No perspective transformation is done on the Z (depth) coordinates. They retain their required depth relationships in three-space.)

The vertex nodes are the first nodes to be established. The X, Y, and Z real number vertices are mapped into an integer "world" using linear mapping functions based on the maximum and minimum values for each axis. This linear "world" must be large enough to retain an accurate description of the object but small enough to prevent integer overflow in math calculations. The last two fields of a vertex node are links (referred to in Figure 5 as L1 and L2). Link L1 is used to establish an efficient drawing order (to minimize pen movements) for the final output drawing. To do this the picture plane is divided up into a two-dimensional square grid. Each vertex is initially placed into its appropriate grid cell, and then link L1 links all the vertices together by tracing through the grid, one row at a time. Link L2 is a pointer to a list of edges with this vertex as their starting point. By "visiting" each vertex through link L1 and each edge of a vertex through link L2, an efficient method is created that guarantees access to each edge in the drawing.

A face node contains three fixed fields and a variable number of vertex pointers. Field three indicates how many vertex pointers make

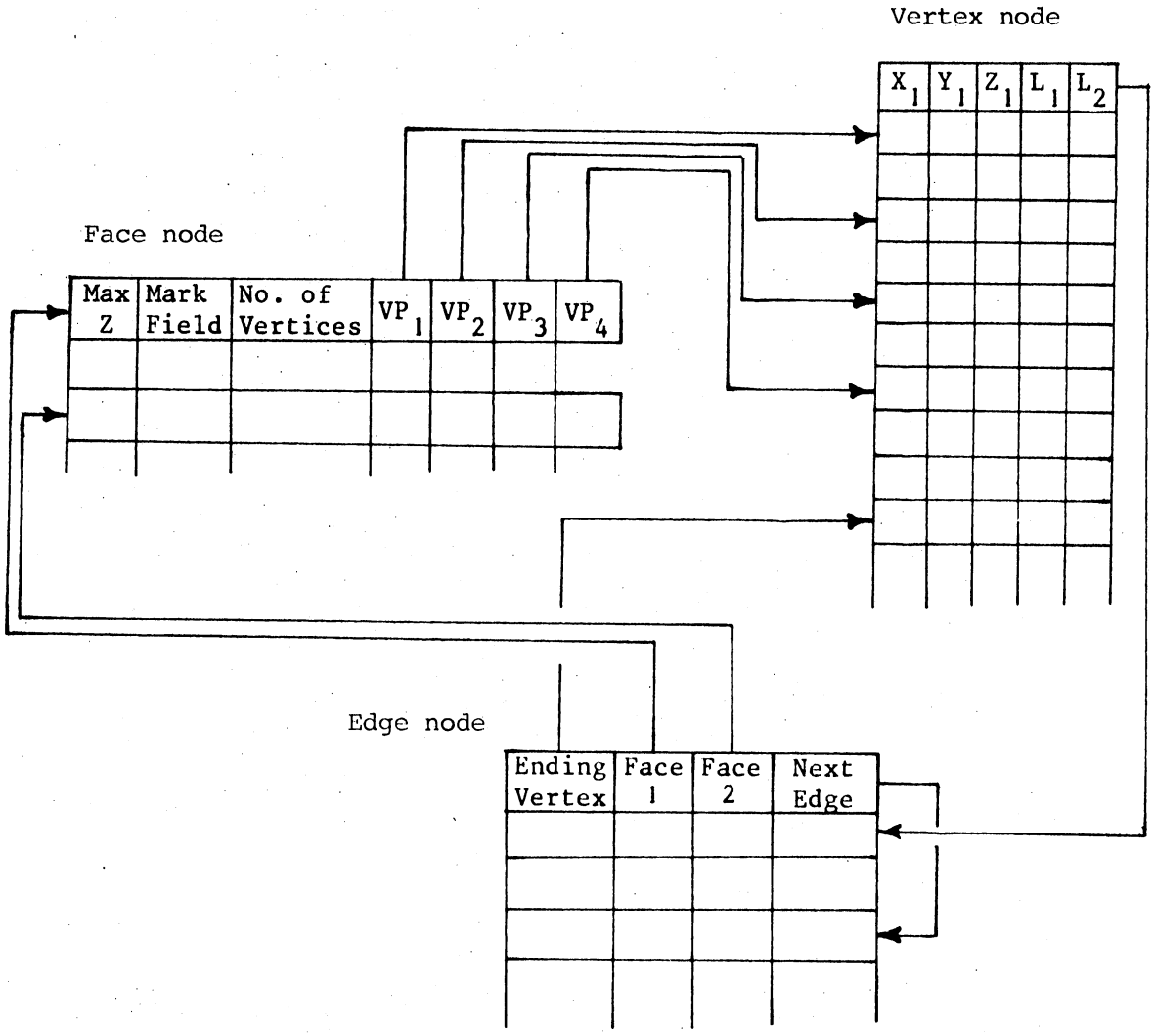


Figure 5. Griffith Data Structure

up this face. Field two is used as a link in setting up the data structure but is used as a "marking field" in the hidden-line removal process to avoid repetitious face comparisons. Field one stores the maximum Z value on the face (its nearest point to the observer's eye.)

An edge node has four fields. Each edge node is in a linked list of edges, each having the same starting vertex. The L2 link in the vertex node points to the first edge in the list and the fourth field in the edge node (labeled "next edge" in Figure 5) points to each succeeding edge in the list. If the field is zero, then no more edges exist with this starting vertex. The middle two fields are pointers to the faces which have this edge as one of its boundaries. If the edge is used by only one face, then both fields point to the same face. The first field is a vertex pointer to the ending vertex of this edge. Each edge node is guaranteed to be unique by the restriction that the starting vertex pointer is always greater than any of its ending vertex pointers.

As the face nodes are established, each face is sorted according to is maximum Z value using a "bucket" sort which is a form of a radix sort. The resulting list of faces begins with the farthest face from the observer and ends with the nearest face. The next task is to link each face into every grid cell that it covers or intersects. To do this exactly would require many calculations to establish the intersections of each edge of a face with the grid lines of the grid. To eliminate these costly computations the maxmin test is used. The maximum and minimum X and Y values are used to create a surrounding rectangle and the face is linked into each grid cell that this rectangle

covers (Figure 6, Rectangle C). Obviously, this is very inaccurate for slender faces at angles other than approximately zero or 90 degrees, but the enormous savings in computation makes it worthwhile. In the final structure, each cell contains a list of faces such that the first face is the nearest face and each succeeding face is farther and farther away from the observer (Figure 6). It is possible that two faces may not be in the correct order in a cell because only the maximum Z is used for sorting (Figure 2), but the ordering is good enough to establish a maxmin test for the Z coordinates, to be discussed later.

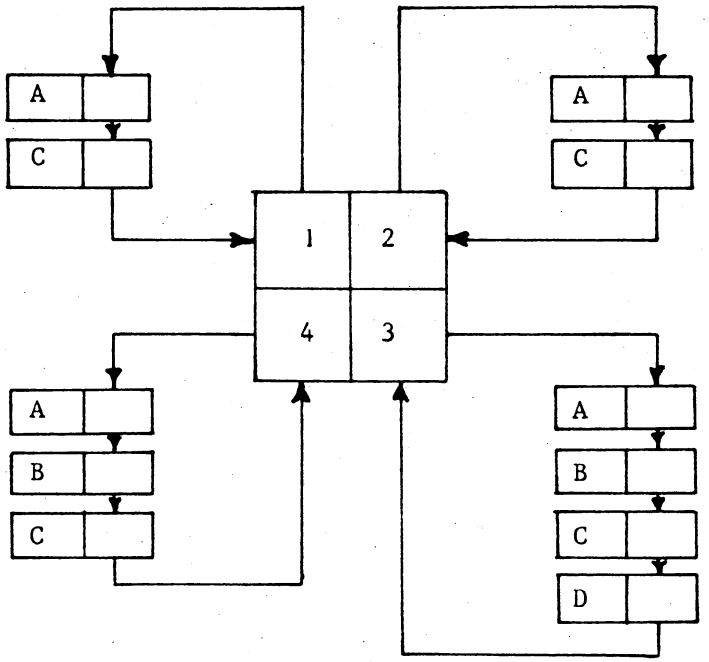
The data structure is now complete and ready for hidden-line removal.

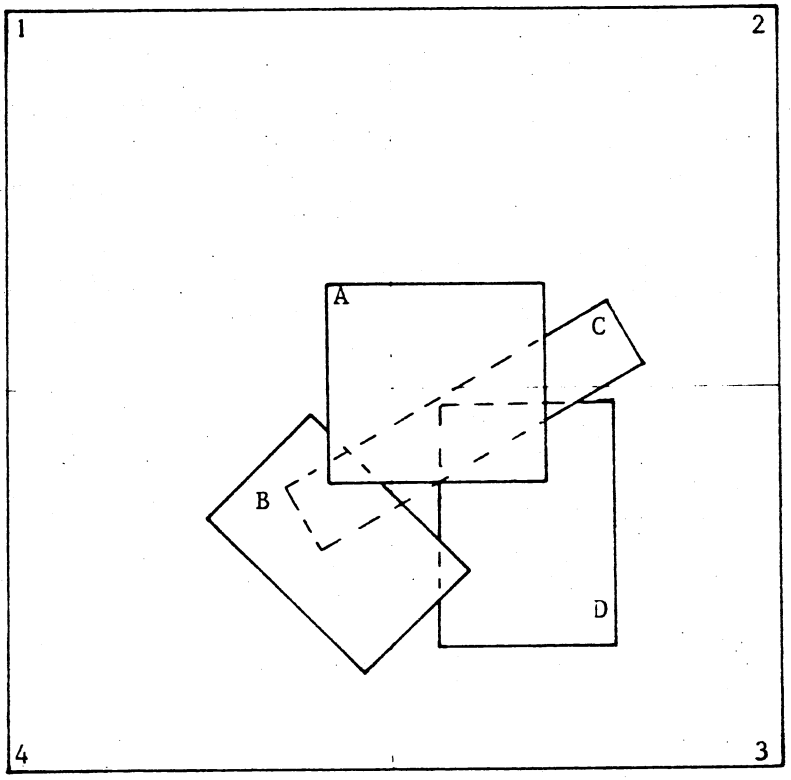
Hidden-Line Removal

First, a very general description of the method is needed. Each edge is visited once and compared to every face that could possibly hide it. If, at any time during the comparisons, the edge is determined to be totally hidden, then comparisons begin on a new edge. If after all comparisons, there is still part of the edge visible, then the visible part is output to a storage device for later drawing.

Using L1 and L2 vertex links, a starting vertex is established. Each edge with this vertex is examined before a new starting vertex is established. All edges are examined in this manner.

To compare an edge to all of the possible faces that cover it, each face in every cell that the edge crosses must be examined. Constants are calculated to determine which cells the edge crosses. Three things help to eliminate unnecessary face comparisons.





Note: Sixteen vertices results in the four cells - a two by two grid (one cell for every four vertices.)

Figure 6. Grid Cell Structure

1. A face is marked in its "mark field" with a unique marker associated with the edge being compared. By testing the mark field, no other comparisons will be made with this face, even if the face reappears in other cells along the edge.
2. A face cannot hide one of its own edges. Therefore, using the two face pointers in the edge node, the faces that the edge are part of are "marked" and never compared.
3. A maxmin test is used to compare the depth of the edge with a face. If the minimum Z value on the edge is greater than the maximum Z value of the face, then the face cannot possibly hide the edge and no comparisons are needed with any of the further faces in this cell.

Comparing a face with an edge is a costly operation and the above methods help eliminate many comparisons, along with the fact that only faces in the edge cells are examined. An edge and all remaining faces which could possibly hide it undergo a detailed analysis to determine which segments are visible and which are hidden.

Individual points on the edge can be represented using the parametric equations:

$$X = X_1 + \lambda (X_2 - X_1)$$

$$Y = Y_1 + \lambda (Y_2 - Y_1)$$

where (X_1, Y_1) is the starting vertex,

(X_2, Y_2) is the ending vertex, and

$0 \leq \lambda \leq 1$.

By representing intersection points in terms of λ values, only one number must be stored. Intersections not on the actual edge are quickly

recognized by the conditions $\lambda < 0$ or $\lambda > 1$. Also, the λ's are easily sorted to determine consecutive edge segments along an edge.

Visible edge segments are found through the introduction of a masking line. Given an edge and a face, the face is "cut" by a plane which is defined by the edge and the viewpoint. The intersection of the face and this plane is the masking line. The endpoints of the masking line are taken as the intersection points with the face boundary. Since concave polygons are allowed, there is the possibility for many masking line segments. The masking line segments may completely "mask out" (or cover) the edge, only partially cover the edge, or miss the edge entirely. Two conditions must be met for a masking line segment to cover an edge or a portion of an edge:

1. The face on which the masking line lies must be in front of the edge segment.
2. The masking line segment must be inside the boundary of the face.

To determine the depth relationship between the face and an edge, the depth on the face along the masking line must be found. This could be done using the plane equation for the face as Roberts' (9) algorithm does, but a much simpler method is available using the masking line. Because the masking line and the edge lie exactly on top of each other in the two dimensional picture plane, the same parametric equations can be used to describe both of them. A linear relationship exists between the lambda values along the edge and the Z depth along the masking line. This linear relationship is calculated using a least squares approximation. Once at least two intersection points are known for the masking

line. As stated earlier, intersecting faces are not allowed, and therefore, if part of a masking line segment is in front of an edge, then all of it is. By testing the Z value at the midpoint of the edge segment against the corresponding Z value of the masking line (on the face) the depth is determined. If Z_m is greater than Z_e, then the face can possibly hide the edge segment, and the next test is applied. Otherwise, the segment is visible and not part of the final masking line segments. The next masking line segment is then put through these tests until all segments have been examined.

The final test for visibility is whether the masking line segment is inside or outside of the face. To determine this, an "even, odd crossing test" is used. The number of intersections are counted between the face and a line which starts at the midpoint of the edge segment and goes to some point at infinity (in practice, some point near the boundary of the picture plane). If the number of "crossings" is odd, then the edge segment is inside the face. If the number is even, then it is outside the face (Figure 7). If the masking line segment is inside the face boundary, then its endpoints (a lambda pair) are stored as one of the masking line segments.

The final results of the masking line tests are a list of ordered pairs of lambdas $(\lambda_i, \lambda_{i+1})$, $(\lambda_{i+2}, \lambda_{i+3})$, ... which represent the edge segments no longer visible. These segments must be removed from the list of edge segments that are visible. To do this, the lambdas are merged into a single ordered list. The edge segment lambdas are made negative so that they can still be distinguished from the masking edge segment lambdas. Then, using two switches which change states for each lambda, the visible edge segments are determined and listed as ordered

pairs of lambdas (λ_i, λ_{i+1}), ($\lambda_{i+2}, \lambda_{i+3}$) ... (See Appendix A for a more detailed description of this method using switches.)

This process of detailed face, edge analysis is repeated each time one or more intersections occur on a face. If the edge segments totally disappear, then no output is done, and a new edge is established for comparisons. If, after all possible comparisons, some edge segments still remain, they are output to a storage device for later drawing.

The algorithm is finished after all edges have been examined.

Summary

Each edge that is examined for visibility is compared to a minimum number of faces. This is the key to the performance of the algorithm. Minimax tests in the X, Y, and Z directions help to make only those face comparisons which have possible intersections. Also, faces which

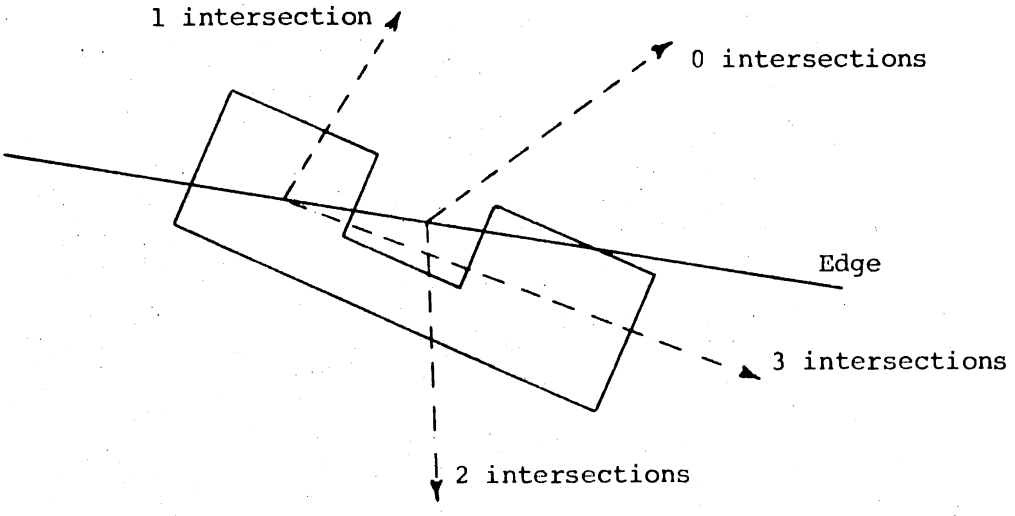


Figure 7. "Even, Odd Crossing Test"

include the edge as a side are omitted from the tests. The size of Griffith's data structure which allows all of this to happen is a major problem for small computers. This problem will be discussed in chapter V, but first a discussion of some improvements to Griffith's algorithm.

CHAPTER IV

CHANGES AND IMPROVEMENTS IN GRIFFITH'S ALGORITHM

After thorough study of Griffith's algorithm, four changes were made:

1. The method of input for the rotation angles was changed to simplify user input and program interaction.
2. Because of integer size and overflow problems on the small computers, some of the integer "mapped" values are changed back to real numbers before calculations are performed.
3. The elimination of all back faces is done while the data structure is being created.
4. A new maxmin test in the Z (depth) comparisons was implemented.

These will be discussed in detail and the results of the implementations given.

View Calculation

To achieve a desired view, Griffith's algorithm simply reads in three angle values for rotation, one for each axis. A 3 x 3 matrix is set up which is multiplied by each individual vertex for the desired rotations. Rotations about the three axes are not commutative; that is, a different view will be computed if the order of rotations is changed. Therefore, it is very important that the user of the algorithm can visualize the type of view desired, the rotations needed to obtain

that view, and the correct order of rotation required. This is not always an easy task for beginners in computer graphic drawings.

An easier, more straightforward, way of "asking" the user for a desired view is to establish a line of sight through which an observer wishes to view an object. This line of sight can be established by locating two points; the point being observed and the point position of the observer. The distance between these points determines the relative size of the final drawing and the "amount" of perspective generated. Only two angles of rotation are required to orient the observer on the positive axis with the positive X axis to the right and the positive Y axis upward. (The third angle of rotation is not used.) Another great flexibility with this method is the ability to look at points other than the origin. Restricting the view to the origin greatly limits the possible number of views. If a point other than the origin is specified as the point being observed, then a simple translation is required with the rotation to align the line of sight with the Z axis. In summary, this method increases the number of views possible and allows the computer to do the "dirty work" of calculating the correct rotation angles.

Implementation into FORTRAN required approximately 30 lines of code. Because the program was run interactively, these computations were not included in the timed portion of the algorithm.

Integer Conversions

Griffith's algorithm converts all vertex coordinates into integers using linear mapping functions based on the maximum and minimum values for each axis. Two restrictions control the size of these linear

mapping functions. If the integer values are too small, then the accuracy of the final drawing can be distorted, even to the extreme of making nonvisible lines visible and parallel lines skewed. From experiments using different ranges of values (based on powers of two for simplicity), the minimum integer range producing no visible distortions on complex drawings was from -1024 to +1024. However, because of overflow problems on various calculations, there is also an upper limit based on the integer size of the computer being used. The equation (I*J) - (K*L) produced the largest possible number from any of the calculations in the algorithm. Therefore, to prevent overflow, each integer value would need to be restricted to the range -128 to +128 for an integer word size of 16 bits; the range -2048 to +2048 for an integer word size of 24 bits, etc.

Virtually all small computers today are 8 bit or 16 bit word machines, as was the computer used for this thesis, a PDP-11/34. The largest integer value in the FORTRAN implemented on this computer was 16 bits. This required a change in Griffith's FORTRAN implementation of his algorithm.

Two sections of code were changed from integer calculations to real calculations: the solution for intersections between lines, and the test for an edge segment to determine whether it is in front of or behind a face. On the PDP-11/34, the real arithmetic calculations are both software and hardware supported. The hardware supported arithmetic was used in all testing. In the extensive testing done, no loss of accuracy was ever discovered from using real calculations and no appreciable increase in computing time was evident.

Back Edge Elimination

The solution to hidden-line elimination can be greatly enhanced if all back faces can be eliminated quickly from the view. A back face is any face on the "back side" of an object. On the average, half of any given solid object is always not visible and therefore half of its faces are back faces. Using a certain convention for describing faces and vector algebra, back faces can be eliminated easily.

Intuition says that by removing approximately half of the faces, the algorithm should run twice as fast and use half as much storage. This proved to be correct, as shown later in the testing discussion.

Implementation in FORTRAN required four lines of code inserted into the face node portion of the data structure. After a face is read in, it is immediately tested to determine whether it is a back face. If it is, the face is left completely out of the data structure along with each of its edges. To establish the correct convention for the vertex pointers order (listed clockwise around the perimeter when looking at the "exterior" of the face), a few minor changes had to be made in Griffith's object building programs.

New Maxmin Test

The maxmin test for depth comparisons uses the maximum Z value on a face compared to the minimum Z value on an edge. This comparison eliminates many face tests but there is a convenient way to make it even better because of the grid cell structure.

The edge is compared to faces in each cell that it passes through. It seems logical then, to compare the maximum Z on a face with the

minimum Z in the cell where the face appears on the edge segment. Whereas a face might fail the broad test against the minimum Z of the edge, it could possibly be eliminated from comparisons by this more exact test (Figure 8).

This test required a total change in the way cells were found along an edge. Griffith used an approximation method which did not solve for exact intersections with the grid cell boundaries. Instead, his method required the calculation of some constants and then only addition and some tests to find the next cell. The new test required exact solutions at the grid line intersections to solve for the Z values at these points.

The implementation in FORTRAN took approximately the same number of lines of code as Griffith's original code. However, when considering efficiency, several mathematical divisions are required in place of his simple addition. This makes the process of finding cells along an edge a little more costly. Testing on a single cube showed an increase in computer time using this method which was attributed to the added "cost" involved in the mathematical computations. But in testing more complex objects, the new method was always comparable to, or a little better than, Griffith's original algorithm. In some cases there was approximately a 5% increase in efficiency. As Table IX shows, the cases where this new test helps the most is for "long" faces compared to fairly close short faces. From the experiments run, in normal applications, this "improvement" adds substantially nothing to the efficiency of the algorithm. But for the occasional special drawing conditions which do occur, this new maxmin test does indeed eliminate enough face comparisons to make it useful.

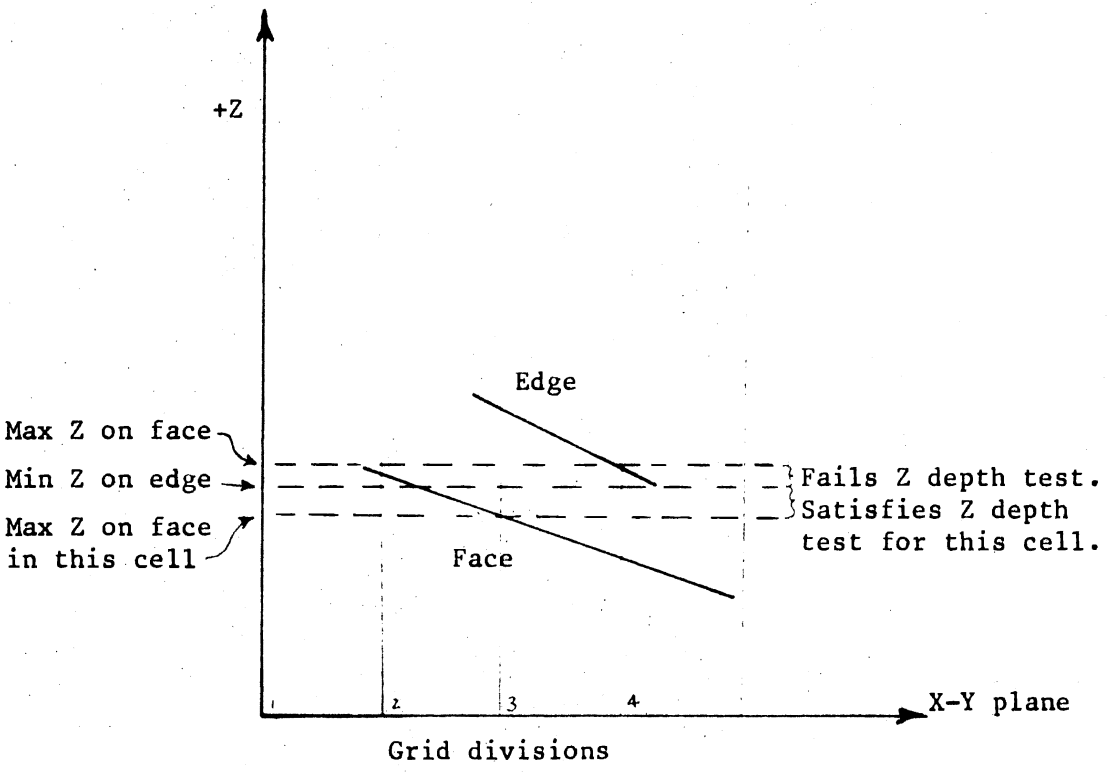


Figure 8. New Depth Z Comparison by Grid Intersections (Grid 3)

Algorithm Testing--Results and Comparisons

Three kinds of comparisons and tests were done on the Griffith algorithm:

1. The algorithm was compared to ten other hidden-line, hidden-surface algorithms using the information presented by Sutherland (10).
2. The same testing that Griffith presents in his published article was simulated to validate his results.
3. The improved versions presented previously were tested against Griffith's original algorithm to measure their benefits.

Each of these will be discussed in the following paragraphs but refer to Appendices B and C for actual figures.

Comparison to Other Algorithms

Griffith's algorithm must be categorized somewhere between the object space algorithms and the image space algorithms. The initial input and final output are related to the object space in that they have high accuracy. But the calculations are done to a limited "resolution": limited, not by the resolution of an output screen, but by the magnitude of integers in the computer being used. Also, this is a comparison algorithm comparing edges to faces. This creates a difficulty in linking Griffith's algorithm into the Sutherland tree structure. The best "fit" is between Roberts and Schumacker, linked half way between the object space and "partly each" space and linked to the comparison algorithms.

The coherence used by Griffith is comparable, in some ways, to Warnock's objects in local areas on the screen to reduce the number

of tests and comparisons needed. There are few apparent similarities with the hidden-line algorithms of Appel, Galimberti, Loutrel, and Roberts. Griffith does not use the notion of quantitative invisibility or edge coherence. Neither is there any of the plane equations as used by Roberts. Griffith avoids the calculation of the plane equation by the introduction of his masking line and least squares approximation. A major distinction of Griffith's method is that it does not have to take care of the special conditions that always seem to arise, such as a face intersecting another face at a single point. Griffith's general masking line concept takes care of all special conditions. The other algorithms do not even discuss solutions to these problems.

In order to compare Griffith's algorithm with the same scheme that Sutherland uses, the algorithm was broken into its major operations and sorting and searching routines. From this examination, the edge to face comparisons are the dominant cost of the algorithm. (This was also verified during early testing of the program by printing out intermediate test values.) The masking line test for visibility is the only other major time consuming task. The figures show that if excessive face comparisons are made with each edge, the results are disastrous. The way to increase the efficiency is to decrease the number of face comparisons.

An interesting relationship appeared between the cost and the depth complexity of the environment. The depth complexity is defined as a measure of how many front faces are pierced by an arbitrary ray from the viewpoint, on the average. Because the depth complexity remained constant in each succeedingly more complex environment, the face sizes, on the average decreased in height and also in the amount of screen area

that they covered. Since each face still has approximately the same number of faces around it on the screen area, the number of face comparisons for each edge remains fairly constant for all three environments. Assuming this is correct, the algorithm grows on a linear basis as a function of the total number of edges. Just as important is the indication that as the depth complexity increases while the number of faces stays constant, many more edge-face comparisons would be required for each edge. The algorithm would tend to grow more nearly exponentially as the depth complexity increased. These are trends in the data and not exactly accurate for any given environment. (See Appendix B for an example.) Other factors that could not be included in the cost calculations are the amount of overlap among faces and the Z depth relationship between faces. Both of these allow the edge-face comparisons to be cut short, which contributes much to the efficiency of Griffith's algorithm. In conclusion, the algorithm has a much greater possibility of a linear growth rate if the environment depth complexities stay small. As the depth complexity grows, the analysis becomes less precise and more dependent on the properties of the particular environment being drawn.

Comparison to Griffith's Original Results

Griffith's computer was much faster than the PDP-11/34 which was used for implementation and testing for this thesis. But the same trends in computation time were received. The FORTRAN timing function of a PDP-11/34 is not very accurate and thus the figures cannot be taken as exact data but they can be used as relative indicators of computing time. The final two test cases were not run because of lack

of memory for these large drawings. The results can be seen in Table V in Appendix C.

Griffith's test case of "a lattice of cubes" takes advantage of the grid cell structure of the data structure. Another test case was constructed which contained long rectangular boxes intermixed with the cubes. This would tend to make the algorithm "work" harder if the rectangles were at an angle such that they were linked into many cells of the grid of which they did not cover or intersect. This would increase the number of edge to face comparisons and consequently the amount of computer time. The results showed an approximately 100% increase in the time required for calculations if the angle of the rectangular boxes was from 30 to 45 degrees from the horizontal. (Refer to Table VI, Appendix C.) This shows so dramatically the importance of the environment in relation to the performance of the algorithm. Griffith's algorithm is based on an area coherence scheme and if the environment is not area coherent, the algorithm losses much of its efficiency. Even though the efficiency dropped drastically as the environment changed, the computer time still grew at a linear rate. This seems to prove that the algorithm grows at a linear rate regardless of the environment, assuming that the environment's coherence properties remain fairly uniform.

Comparisons with Improved Versions

Three versions of the algorithm with improvements were tested against Griffith's original algorithm:

1. New Z depth test for face to edge comparisons.
2. Back faces removed.

3. Both of the above together.

These were run on the original test case of "a lattice of cubes" and on the new test case with long rectangular boxes. The new version's performances have already been mentioned in the previous discussion. The test results can be seen in Table VII and Figures 9 and 10 of Appendix C.

CHAPTER V

CONCLUSIONS AND SUGGESTIONS

All of the goals of this thesis were accomplished and have been presented in the previous chapters. A few specifics on how they were accomplished follows, with comments on the implementation, problems encountered, and ideas for future work in this area.

After receiving a copy of Griffith's algorithm, it was first implemented on an IBM 370/168. All of the file processing was changed to make it compatible with the IBM file processing. No other changes were necessary and the program ran successfully. No timing was done because the multiprocessing environment made any timing functions almost meaningless.

To test the program on a small computer, it was implemented on a PDP-11/34. The major problems were the 16 bit word size and the small memory size. Many of the integer calculations had to become real calculations to prevent overflow, as discussed in chapter 4. As for the memory size, each program run on a PDP-11/34 has 32K of memory for all of the code and data. This is obviously too small for the array which holds all of the data structure. The PDP-11/34 allows access to other in-core memory through what is called "virtual" arrays. By this technique the data structure was stored in a 32K integer array outside of the memory partition which ran the program. While the program was executing, the amount of memory access for it essentially deleted the

multiprocessing capability of the PDP-11/34; there was no more memory for other programs. This makes the algorithm very detrimental to a multiprocessing environment if it is executing for long periods of time on complex pictures. A new algorithm design is needed such that only a small necessary part of the data structure is stored in core memory at any given time and the rest is on some kind of high speed peripheral storage device. Some breakdown of the data structure would be required to access the data efficiently.

Future work is needed in several areas.

1. The environment condition that all faces must intersect on face boundaries is restrictive. Some way is needed to extend the general masking line scheme to allow for penetrating faces.
2. Many times three-dimensional environments are created from a group of basic building blocks, such as a group of cubes to create a building. A method is needed to eliminate all overlapping edges so that a continuous shape is created when the hidden-lines are removed.
3. More work is needed in implementing such large data requirements on small computers, as discussed previously.

Many problems are still to be solved and more problems will arise in the future.

The original FORTRAN code was not structured in any readable format and it was not documented. To allow for easier study during this thesis work and for future work, the code was rewritten into a more readable format. All major loops and decision statements were documented. The resulting code is in Appendix D.

Griffith's algorithm is a very efficient scheme for hidden-line removal. Its use in the field of computer graphics will grow in the coming years. Small computer applications will also have many uses for this algorithm as they gain more power and memory.

A SELECTED BIBLIOGRAPHY

(1) Appel, A. "The Notion of Quantitative Invisibility and the Machine Rendering of Solids." Proc. 1967 ACM Nat. Conf., 22 (1967), 387-393.

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

(3) Griffith, J. G. "A Data Structure for the Elimination of Hidden Surfaces by Patch Subdivision." Computer Aided Design, 7, 3 (1975), 171-178.

(4) Griffith, J. G. "Bibliography of Hidden-line and Hidden-surface Algorithms." Computer Aided Design, 10, 3 (May, 1978), 203-206.

(5) Griffith, J. G. "A Surface Display Algorithm." Computer Aided Design, 10, 1 (January, 1978), 65-73.

(6) Griffith, J. G. "Eliminating Hidden Edges in Line Drawings." Computer Aided Design, 11, 2 (March, 1979), 71-78.

(7) Newell, M. E., Newell, R. G., and Sancha, T. L. "A Solution to the Hidden Surface Problem." Proc. ACM 1972 Nat. Conf., 1972, 443-450.

(8) Newman, W. M. and Sproull, R. F. Principles of Interactive Computer Graphics. 2nd Ed. New York: McGraw-Hill Book Co. Inc., 1979.

(9) Roberts, L. G. "Machine Perception of Three Dimensional Solids." In J. T. Tippet, et al (eds), Optical and Electro-optical Information Processing. Cambridge: MIT Press, 1964, 159-197.

(10) Sutherland, I. E., Sproul, R. F., and Schumacker, R. A. "A Characterization of Ten Hidden-surface Algorithms." ACM Computing Surveys, 6 (1974), 1-55.

(11) Watkins, G. S. "A Real-time Visible Surface Algorithm." University of Utah Dept. of Computer Science, UTEC-CSC-70-101. (June, 1970), NTIS AD-762 004.

(12) Williamson, H. "Hidden-line Plotting Problem." Communications of the ACM, 15 (February, 1972), 100-103.

APPENDIX A

EXAMPLE OF GRIFFITH'S ALGORITHM

INTRODUCTION

This appendix is a very simple example of the data structure and the masking line concept. In an attempt to make the example "readable", subscripted pointers are used instead of actual numbers. Refer to Chapter III for a more detailed description of the overall concepts.

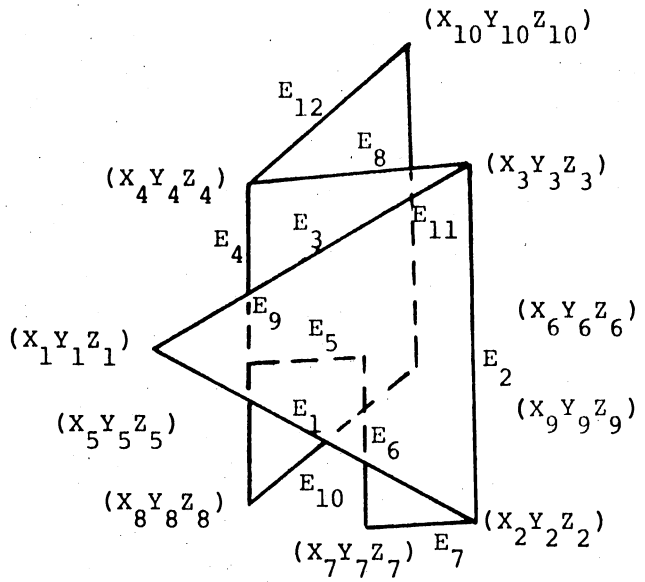


Figure 9. Example Picture

Face Nodes

F_1	Z_1		3	V_1	V_2	V_3		
F_2	Z_4		6	V_4	V_5	V_6	V_7	V_2	V_3
F_3	Z_4		4	V_4	V_8	V_9	V_{10}		

Vertex Nodes

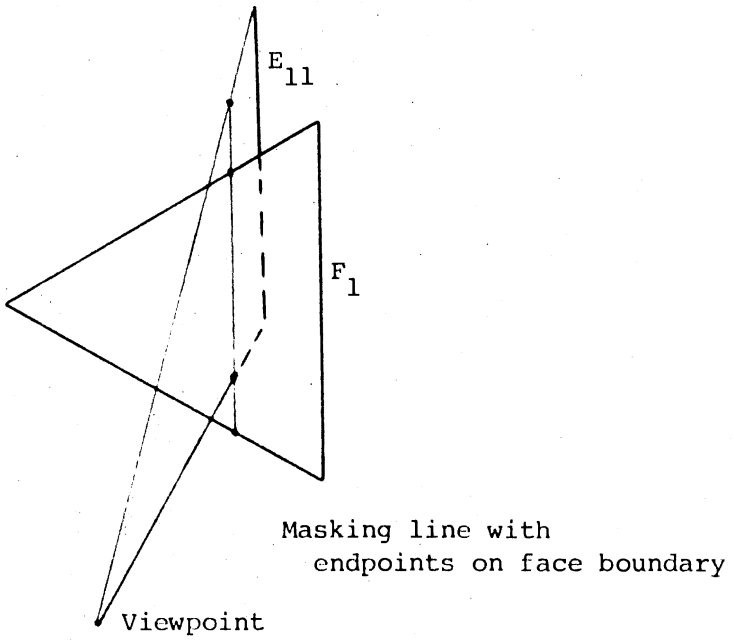
V_1	X_1	Y_1	Z_1		0
V_2	X_2	Y_2	Z_2		E_1
V_3	X_3	Y_3	Z_3		E_2
V_4	X_4	Y_4	Z_4		E_8
V_5	X_5	Y_5	Z_5		E_4
V_6	X_6	Y_6	Z_6		E_5
V_7	X_7	Y_7	Z_7		E_6
V_8	X_8	Y_8	Z_8		E_9
V_9	X_9	Y_9	Z_9		E_{10}
V_{10}	X_{10}	Y_{10}	Z_{10}		E_{11}

Edge Nodes

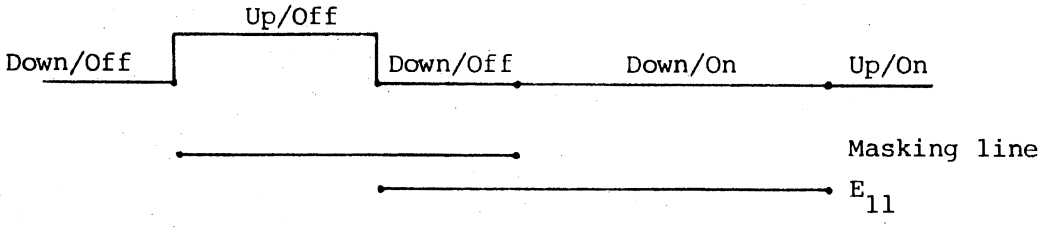
E_1	V_1	F_1	F_1	0
E_2	V_2	F_1	F_2	E_3
E_3	V_1	F_1	F_1	0
E_4	V_4	F_2	F_3	0
E_5	V_5	F_2	F_2	0
E_6	V_6	F_2	F_2	E_7
E_7	V_2	F_2	F_2	0
E_8	V_3	F_2	F_2	0
E_9	V_4	F_3	F_3	0
E_{10}	V_8	F_3	F_3	E_{11}
E_{11}	V_9	F_3	F_3	E_{12}
E_{12}	V_4	F_3	F_3	0

Figure 10. Example Data Structure

The following figures are an example of a masking line compared to an edge to eliminate hidden edge segments. Edge eleven (E_{11}) is compared to faces one (F_1) and two (F_2). It is not compared to face three (F_3) because a face cannot hide one of its own edges. Face one is compared first (Figure 11a). Note that the masking line lies on the face and is created by the face intersection with the imaginary plane formed by the edge and the viewpoint.



(a) Masking Line



(b) Masking Line Switches

Figure 11. Edge Eleven and Face One Comparison

The relationship of the masking line to the edge is shown in Figure 11b, as well as the switch positions for visible segment determination. Visible segments are created only from the down/on switch setting. The masking line segments control the up/down switch and the edge segments control the on/off switch. Only a portion of the edge segment is now visible and only this much is used for future testing against other faces.

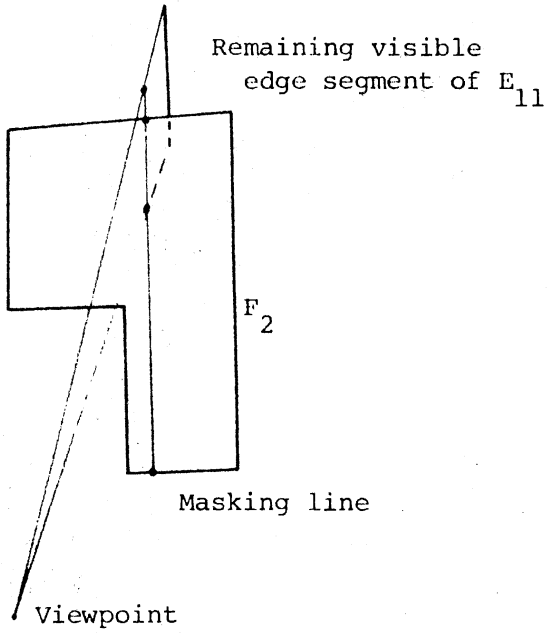


Figure 12. Edge Eleven and Face Two Comparison

The same relationship for the switches is true for the resulting masking line and edge segment in Figure 12. A portion of E_{11} is still visible and no other faces remain to be tested. The visible segment is output for later drawing. Comparisons on a new edge are started.

APPENDIX B

SUTHERLAND'S ALGORITHM COMPARISONS

Description of Faces

Appendix B cannot be understood without a thorough study of Sutherland's article "A Characterization of Ten Hidden-surface Algorithms". The explanation of this article would require more space than is available here.

Tables I through IV are copies of tables from Sutherland's (10) article with additions which refer to Griffith's (6) algorithm. Table I defines a set of variables used to describe an environment. As seen in Table II, some of these values are "given" as initial data to establish an environment. The other variables are defined in terms of these basic given variables. An understanding of each algorithm is needed to fully understand the formulas in Table II.

Three environments are set up, each one being twenty-five times more complex than the previous one. The "Roberts' House" has only 100 faces, while "Big Harbor" has 60,000 faces. Using the variables and their values for each environment, the "cost" of computing a hidden-surface picture is calculated. This cost is a relative value based on the number of comparisons an algorithm makes, the kind of mathematics involved in those comparisons, and the types of sorting done. These costs give some insight into the efficiency of each algorithm. Table III shows the equations which are used to obtain the final costs for each algorithm in each environment. The totals are in Table IV. Note that because of the relative nature of the costs assigned to each algorithm, numbers in Table IV which are within a magnitude of 10 of each other are considered to be close in efficiency. When the magnitudes approach 100, then those algorithms with lower costs are definitely more efficient.

TABLE I

ENVIRONMENT STATISTICS

F_t	Total number of faces in the environment.
F^r	Number of relevant faces in the environment.
D^c_t	Depth complexity of the environment (average).
C^c	Number of relevant clusters in the environment.
F^c_t	Number of faces per cluster (average).
E^t	Total number of edges in the environment.
E^r	Number of relevant edges in the environment.
E^s	Number of relevant edges if sharing is allowed.
E^c	Number of contour edges in the environment.
X^r	Total number of edge crossings in the viewing plane.
X^v_f	Number of intersections of visible edges.
H^f	Number of face intersections.
S^r	Height of a face in resolution units (average).
S^v_l	Total number of segments, visible or not.
L^v	Number of segments on a scan line, visible or not (average).
n^v	Number of visible segments on a scan line (average).
m	Total length of visible edges (measured in resolution units).
	Vertical resolution of screen (number of scan lines).
	Horizontal resolution of screen.

New Definitions*

E	Total number of edges in a cluster. ($4F_c$)
X^p	Total number of edge intersections in a cluster.
X^p_m	Number of edge intersections on a masking line.
S^m	Number of segments on a masking line.
S^m_v	Number of visible segments on a masking line.
V^{mv}_t	Total number of vertices.

*Author's additions

Source: Sutherland (10, p. 47)

TABLE II

STATISTICS FOR THREE ENVIRONMENTS

Statistic	Rule of Thumb	Roberts' House (1/25)	Harbor (1)	Big Harbor (25)
n	given	500	500	500
m	given	500	500	500
F_r	given	100	2500	60000
F_c	given	10	25	200
D_c	given	3	3	3
F_t	$2F_r$	200	5000	120000
C_t	F_t/F_c	20	200	600
E_t	$4F_t$	800	20000	480000
E_r	$E_t/2$	400	10000	240000
E_c	$E_r/(F_c/2)^{1/2}$	180	2800	24000
E_s	$(E_r-E_c)/2+E_c$	290	6400	130000
X_r	$(D_c-1)E_r/4$	200	5000	120000
X_v	X_r/D_c	70	1700	40000
H_f	$(nmD_c/F_r)^{1/2}$	86	17	4
S_l	$(D_cF_rm/n)^{1/2}$	17	87	420
S_v	S_l/D_c	5	29	140
L_v	$2nS_v$	5000	29000	140000
<u>New Definitions</u>*				
E_p	$4F_c$	40	100	800
X_p	$(D_c-1)E_c/4$	20	50	400
X_m	X_p/E_p	1/2	1/2	1/2
S_m	$2X_m$	1	1	1
S_{mv}	S_m/D_c	1/3	1/3	1/3
V_t	$3/2E_t$	1200	30000	720000

*Author's additions

Source: Sutherland (10, p. 47)

TABLE III

COSTS FOR THREE ENVIRONMENTS

Roberts			
1. Back-facing edges cull			
E_t	800	20K	480K
2. Clipping cull			
100 E_s	29K	640K	13M
3. Edge/volume test			
100 $E_s f C_t$	2.3M	510M	31B
f = 4; split edges, and C_t should be higher			
Appel, Loutrel, Galimberti and Montanari			
1. Back (and contour edge) cull			
E_t	800	20K	480K
2. Initial visibility search			
100 $C_t F_r$	200K	50M	3.6B
3. Edge intersection			
30 $E_s E_c$	1.6M	540M	93B
4. Invisibility correction			
30 ($2E_s$3)	52K	1.2M	23M
5. Sort along edge			
$E_s(X_t/E_s)$ $\log_2$ (X_t/Es)	290	6.4K	130K
Griffith*			
1. Vertex links for drawing			
V_t	1.2K	30K	720K
2. Z sort			
F_t	200	5K	120K
3. Area covered sort			
$4F_t$	800	20K	480K
4. Edge intersection			
$30E_t(48-8)$	960K	24.6M	590.4M
5. Intersection sort			
$E_t(X_m+3)$	2.8K	70K	1.68M
6. Masking edge test			
$50E_t S_m$	40K	1M	24M
7. merge			
$E_t S_m$	800	20K	480K
8. Output			
$E_t(S_m v)$	270	6.7K	160K
Schumacker et al			
1. Intra-cluster priority			
100 $F_c^2 C_t$	200K	12M	2.4B
2. Inter-cluster priority			
10 C_t	200	2K	6K
3. Back-face cull			
F_r	100	2.5K	60K
4. Y cull			
n E_s	150K	3.2M	65M
5. X sort and priority search			
nmS_1	4.2M	22M	100M
Newell et al			
1. Z sort			
$2F_r$	200	5K	120K
2. Newell special			
$F_r^2(f+f^2+100f^3)$ f = 2 H_f/n	45K	650K	60M
3. Segment generator and Y sort			
10 $F_r H_f$	86K	420K	2.4M
4. X merge			
$F_r H_f S_v/4$	11K	310K	8.4M
Warnock			
1. Z sort			
2 F_r	200	5K	120K
2. Warnock special cull			
100 $L_v D_c$	1.5M	8.7M	42M
3. Depth search			
$L_v D_c$	15K	87K	420K
Romney et al			
1. Y sort			
2 F_r	200	5K	120K
2. X sort			
nS_1	8.5K	43K	210K
3. X priority search			
nm	250K	250K	250K
4. Depth search			
20 n $2S_1 D_c f$ f = 1/2; due to depth coherence	510K	2.6M	13M
Watkins, Bouknight			
1. Y sort			
E_r	400	10K	240K
2. X merge			
$E_r S_1/2$	3.4K	430K	50M
3. X sort			
$n(S_1+10X_r/(nS_1))$	8.5K	43K	210K
4. Span cull			
n S_1	8.5K	43K	210K
5. Depth search			
$30nD_c$min(m.fS_v) f = 2; spans include not only visible segments	450K	2.6M	13M
Brute-force image space			
No memory:			
$100nmF_r$	2.5B	62B	1500B
Large memory:			
$10H_f^2 F_r$	7.5M	7.5M	7.5M

(Note: K=1,000; M=1,000,000)

*Author's Additions

Source: Sutherland (10, p. 50)

TABLE IV

COST SUMMARY: THREE ENVIRONMENTS

Roberts	Appel, Loutrel, Galimberti and Montanari	Griffith*	Schumacker et al	Newell et al	Warnock	Romney et al	Watkins, Bouknight	Brute force
2.4M	1.8M	1.0M	4.2M	140K	1.5M	770K	470K	2.4B or 7.5M
510M	590M	25M	25M	1.4M	9M	2.9M	3M	62B or 7.5M
31B	97B	618M	170M	71M	43M	14M	64M	1500B or 7.5M

(Note: K=1,000; M=1,000,000)

*Author's additions

Source: Sutherland (10, p. 54)

APPENDIX C

ALGORITHM COMPARISON RESULTS

TABLE V

PROGRAM PERFORMANCE FOR LATTICES OF CUBES

		1904S Computer		PDP-11/34	
	Number of Cubes	Time Taken	Time per Cube	Time Taken	Time per Cube
1.	1	0.257	0.257	3.33	3.33
2.	8	2.28	0.285	22.04	2.75
3.	27	8.84	0.328	76.9	2.85
4.	64	23.6	0.369	163.1	2.54
5.	125	51.5	0.412	375.5	3.00
6.	216	92.6	0.429	-	-
7.	343	161.0	0.469	-	-

TABLE VI

PROGRAM PERFORMANCE FOR LATTICE OF CUBES AND RECTANGLES AT DIFFERENT ANGLES OF ROTATION

Number of Cubes	Using Griffith's Original Algorithm		
	Drawing at 5^{o}	Drawing at 30^{o}	Drawing at 45^{o}
8	23.0	30.8	30.6
27	76.9	117.0	111.6
60	163.1	355.9	313.5
120	375.5	705.5	704.9

TABLE VII

PROGRAM PERFORMANCE FOR LATTICE OF CUBES

Number of Cubes	Griffith Original Algorithm (Sec)	New Z Depth Test (Sec)	Back Faces Removed (Sec)	Both Improvements Together (Sec)
1	3.33	5.13	2.73	2.73
8	24.5	22.9	13.88	15.84
27	101.9	101.6	53.5	56.5
64	275.	274.	147.9	139.2
125	620.	617.	298.	313.
216	-	-	573.	-

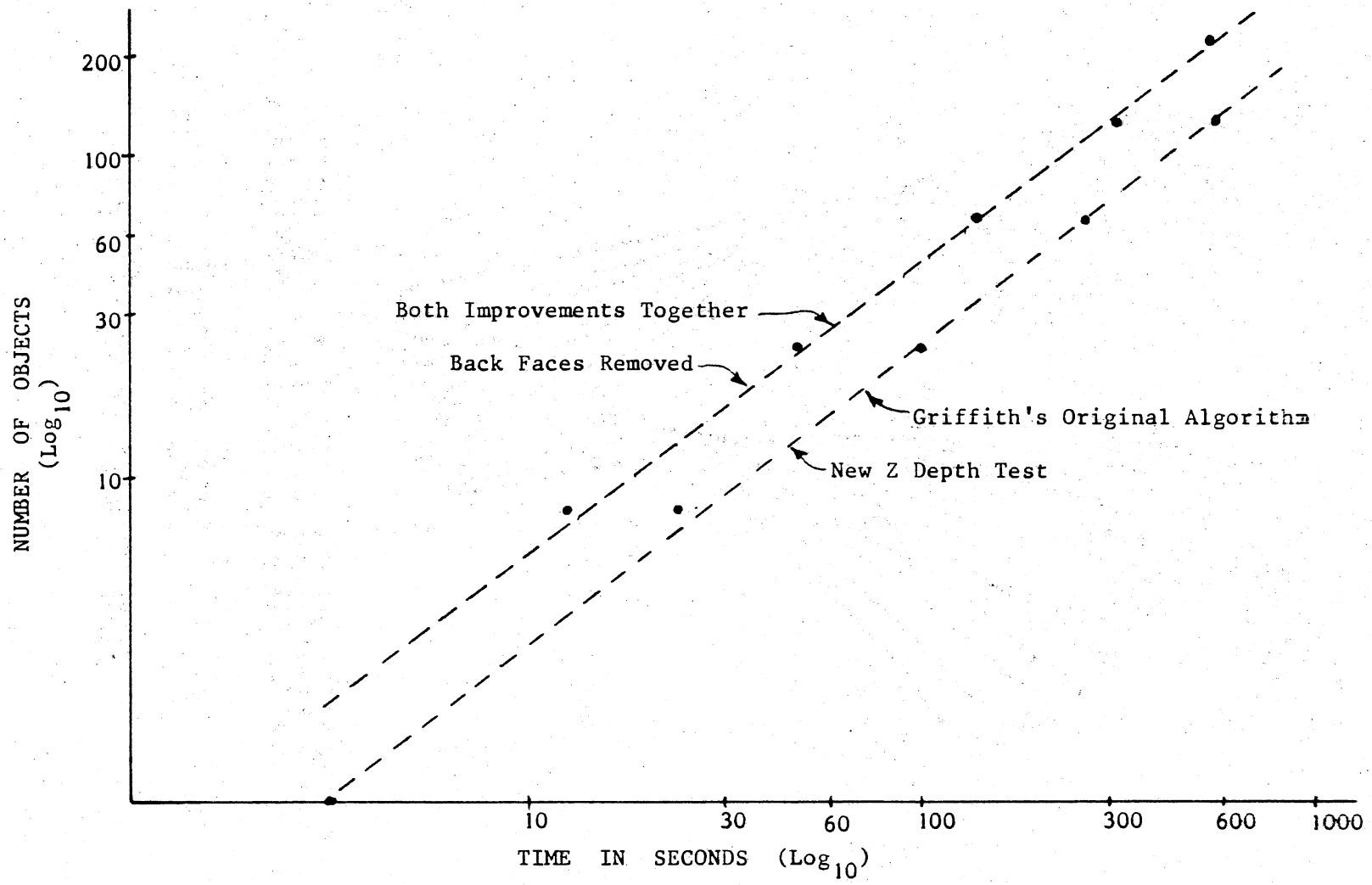


Figure 13. Program Performance for Lattice of Cubes

SYSTEM
PDP 11/34

TIME TAKEN
2.7344

WORDS USED
144

SIZE OF
RESOLUTION
4095

ORIGINAL NUMBER
OF VERTICES
8

ORIGINAL NUMBER
OF FACES
6

VERTICES/CELL
4

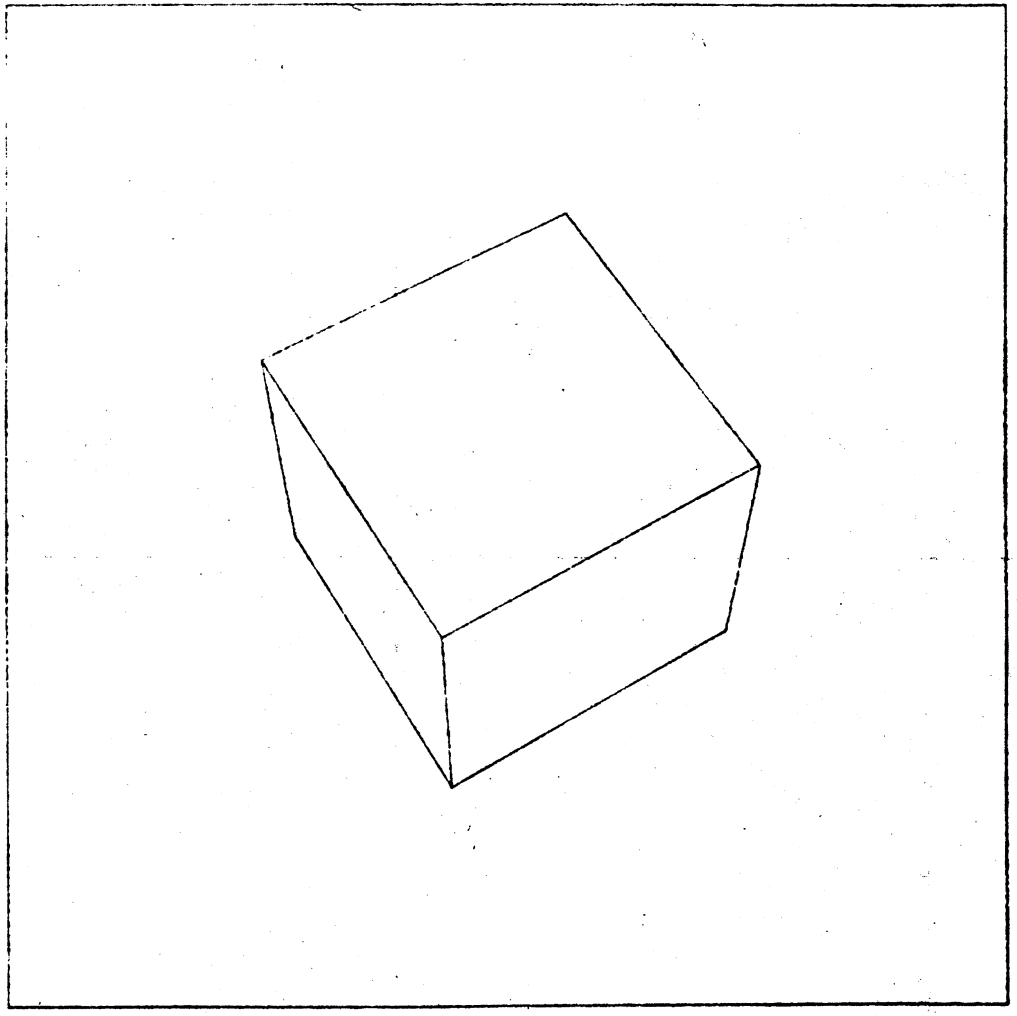


Figure 14. One Cube

SYSTEM
PDP 11/34

TIME TAKEN
13.8828

WORDS USED
1057

SIZE OF
RESOLUTION
4095

ORIGINAL NUMBER
OF VERTICES
64

ORIGINAL NUMBER
OF FACES
48

VERTICES/CELL
4

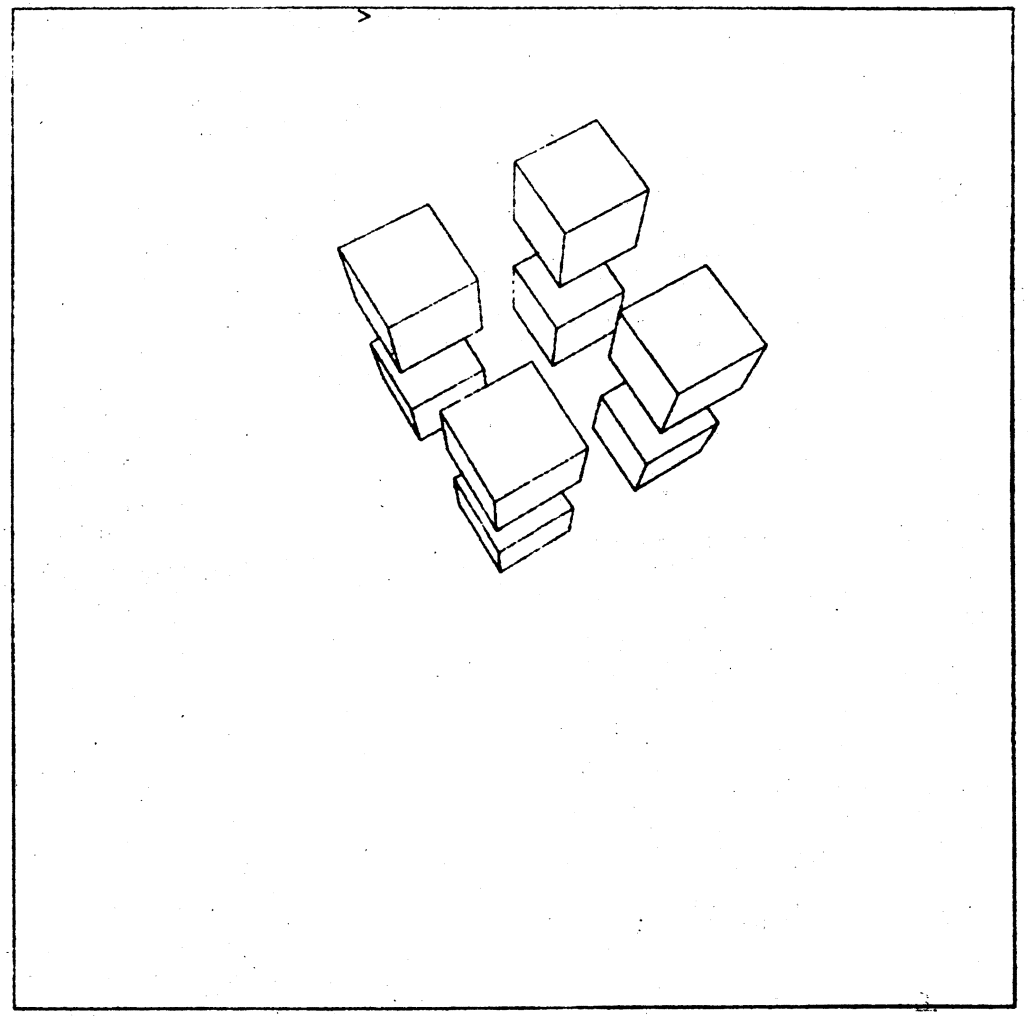


Figure 15. Eight Cubes

SYSTEM
PDP 11/34

TIME TAKEN
101.9141

WORDS USED
5287

SIZE OF
RESOLUTION
4095

ORIGINAL NUMBER
OF VERTICES
216

ORIGINAL NUMBER
OF FACES
162

VERTICES/CELL
4

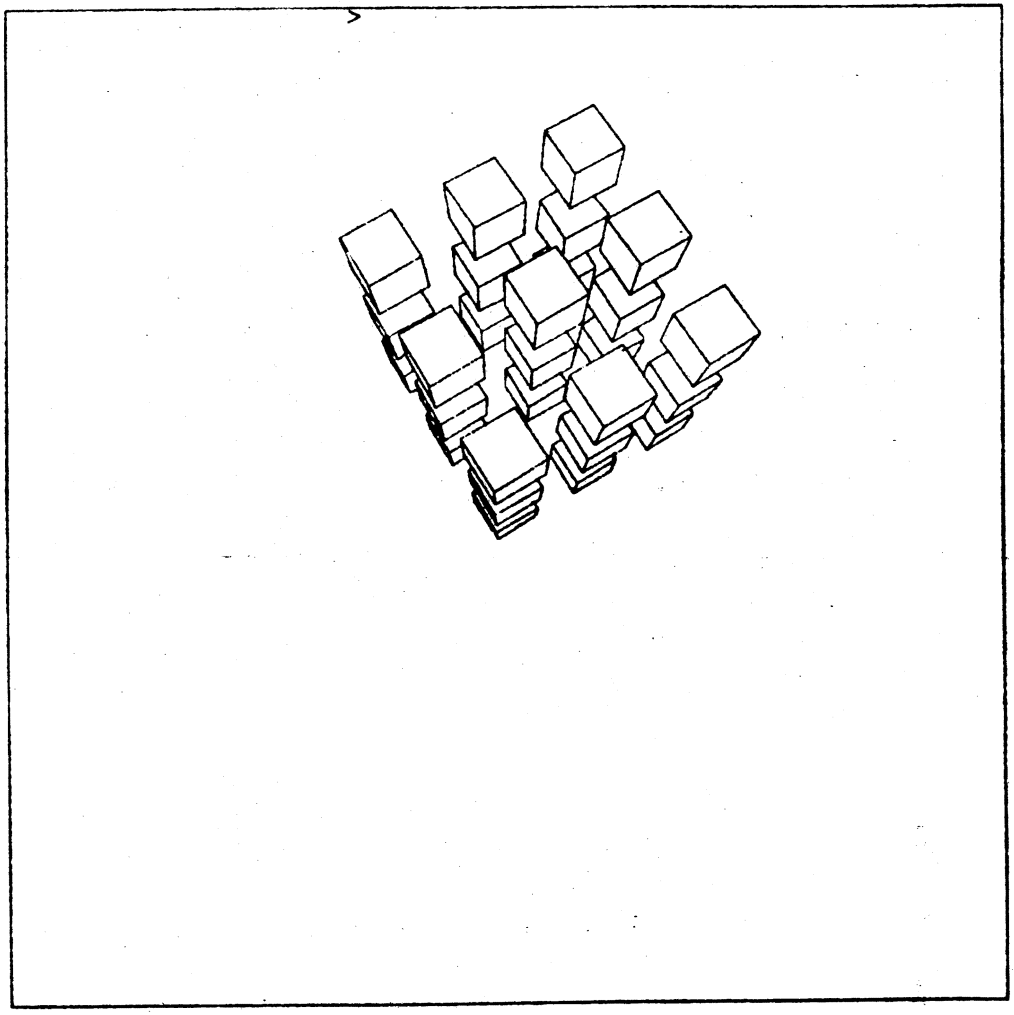


Figure 16. Twenty-seven Cubes

SYSTEM
PDP 11/34

TIME TAKEN
. 275.0703

WORDS USED
12593

SIZE OF
RESOLUTION
4095

ORIGINAL NUMBER
OF VERTICES
512

ORIGINAL NUMBER
OF FACES
384

VERTICES/CELL
4

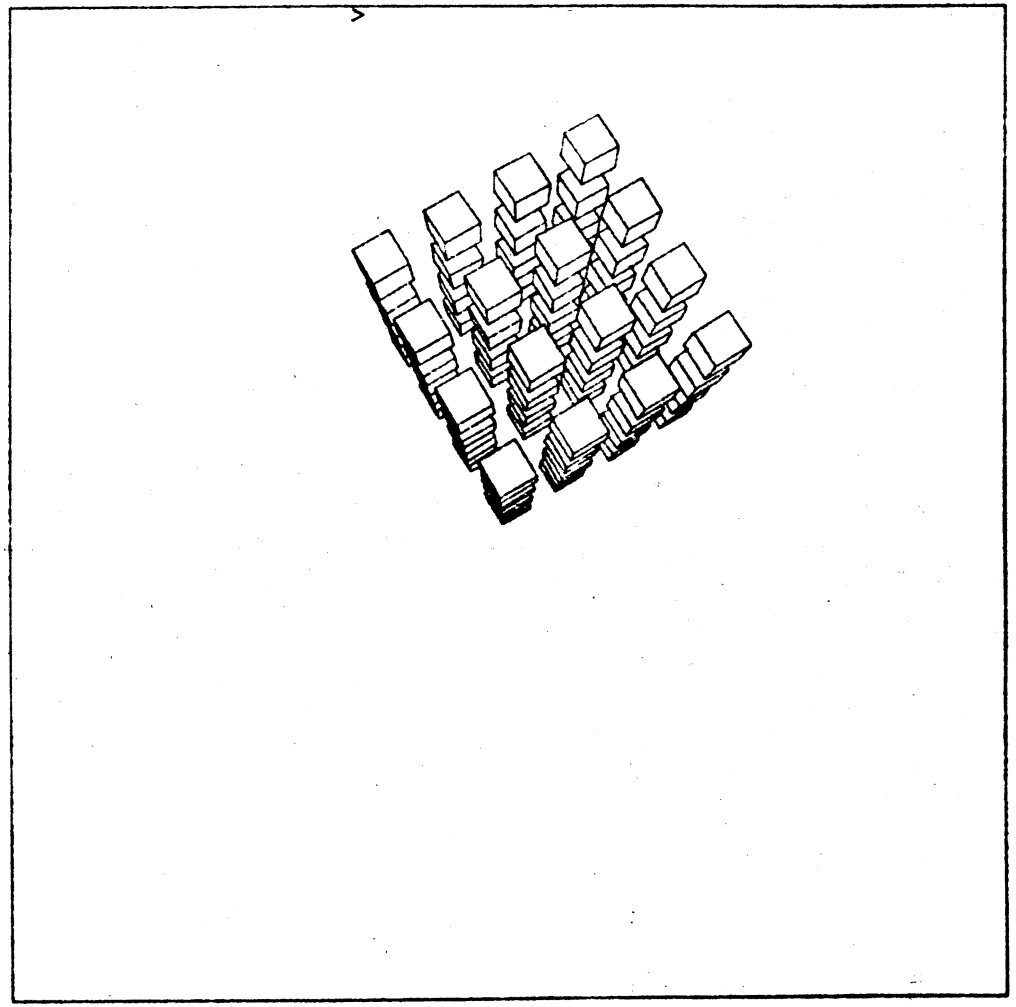


Figure 17. Sixty-four Cubes

SYSTEM
PDP 11/34

TIME TAKEN
620.3359

WORDS USED
24975

SIZE OF
RESOLUTION
4095

ORIGINAL NUMBER
OF VERTICES
1000

ORIGINAL NUMBER
OF FACES
750

VERTICES/CELL
4

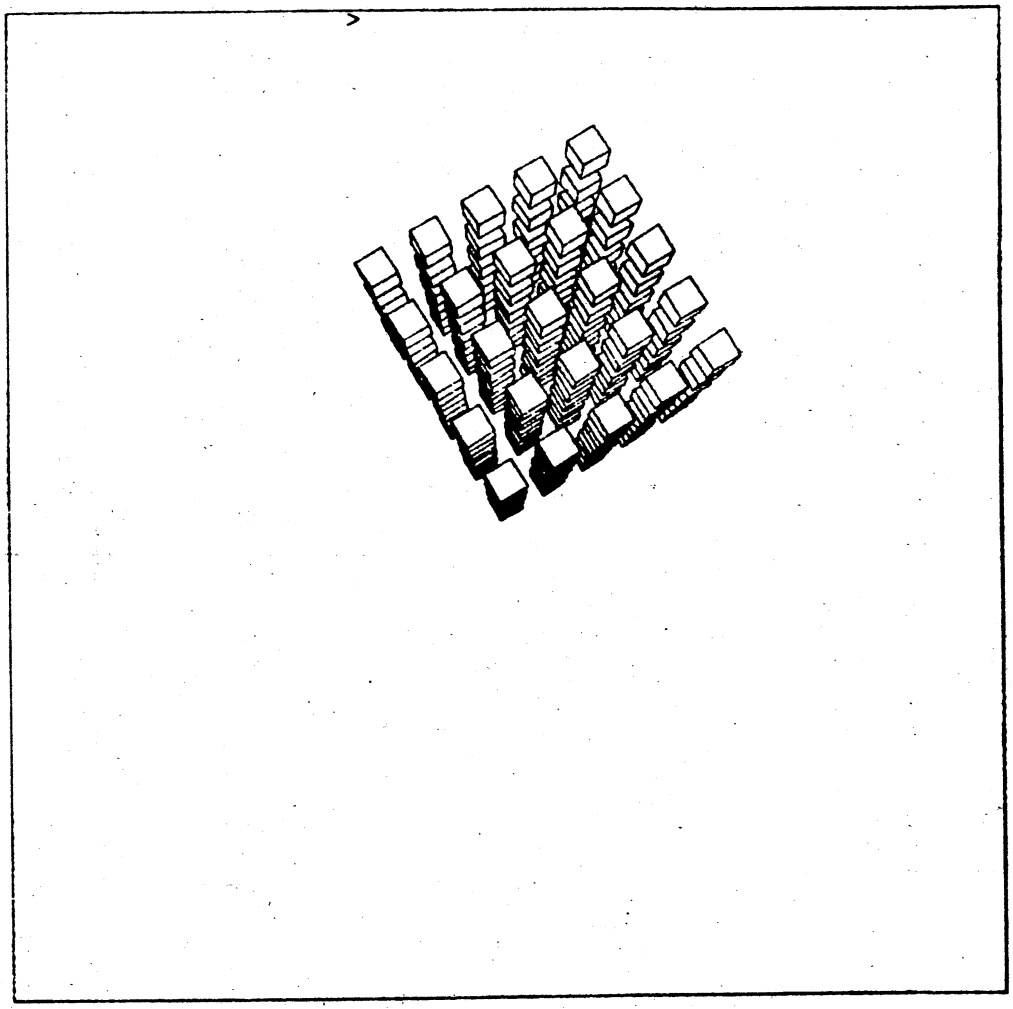


Figure 18. One Hundred Twenty-five Cubes

TABLE VIII

PROGRAM PERFORMANCE FOR LATTICE OF CUBES AND RECTANGLES

Number of Cubes	Griffith's Original Algorithm (sec)	New Z Depth Test (sec)	Back Faces Removed (sec)	Both Improvements Together (sec)
1	3.33	5.13	2.73	2.73
8	30.8	29.0	15.9	16.0
27	117.0	114.8	64.4	59.0
60	355.9	355.7	188.2	190.0
120	705.5	704.1	351.5	359.7
216	-	-	-	-

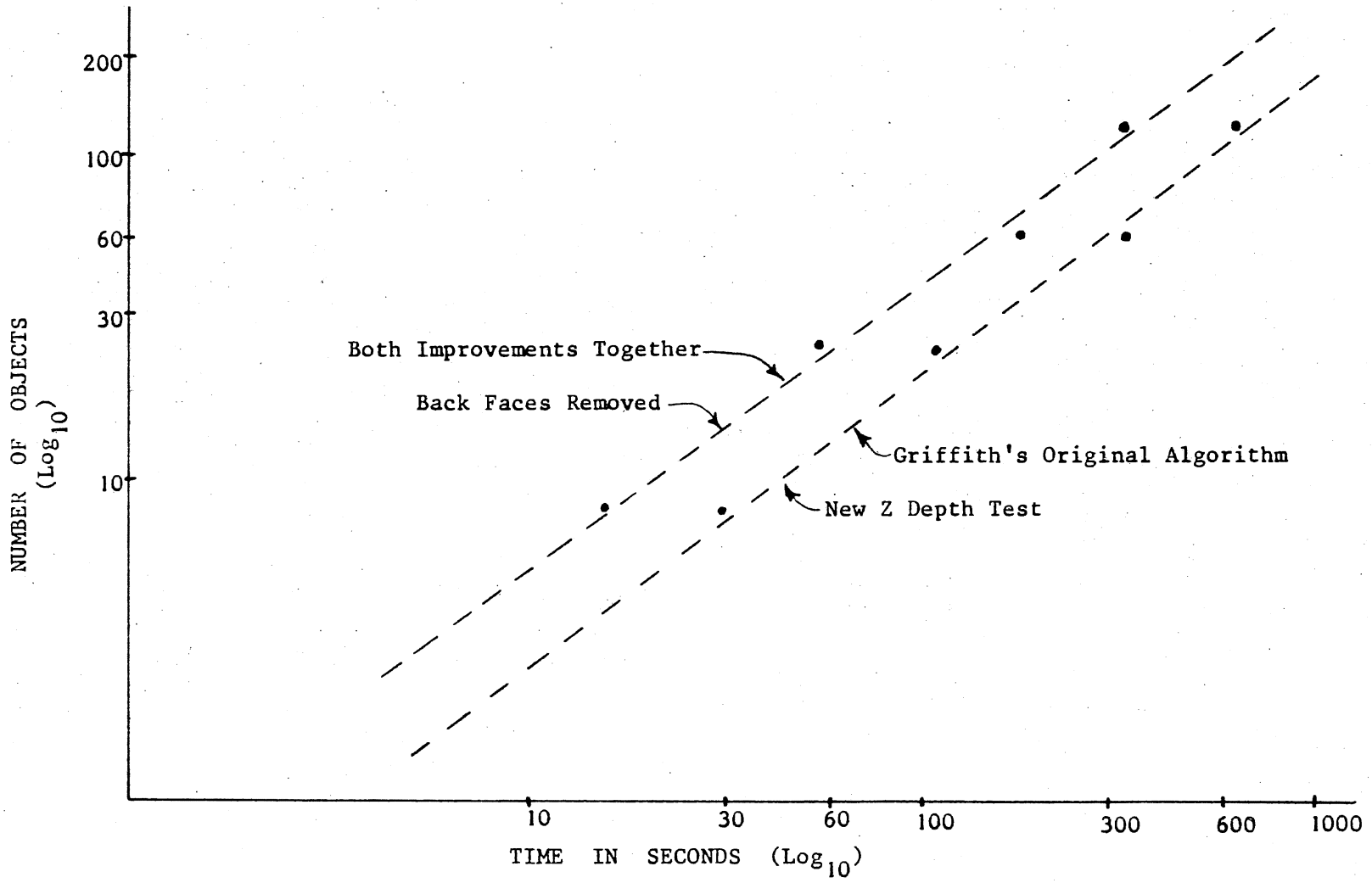


Figure 19. Program Performance on Lattice of Cubes and Rectangles

SYSTEM
PDP 11/34

TIME TAKEN
30.7734

WORDS USED
1599

SIZE OF
RESOLUTION
4095

ORIGINAL NUMBER
OF VERTICES
64

ORIGINAL NUMBER
OF FACES
48

VERTICES/CELL
4

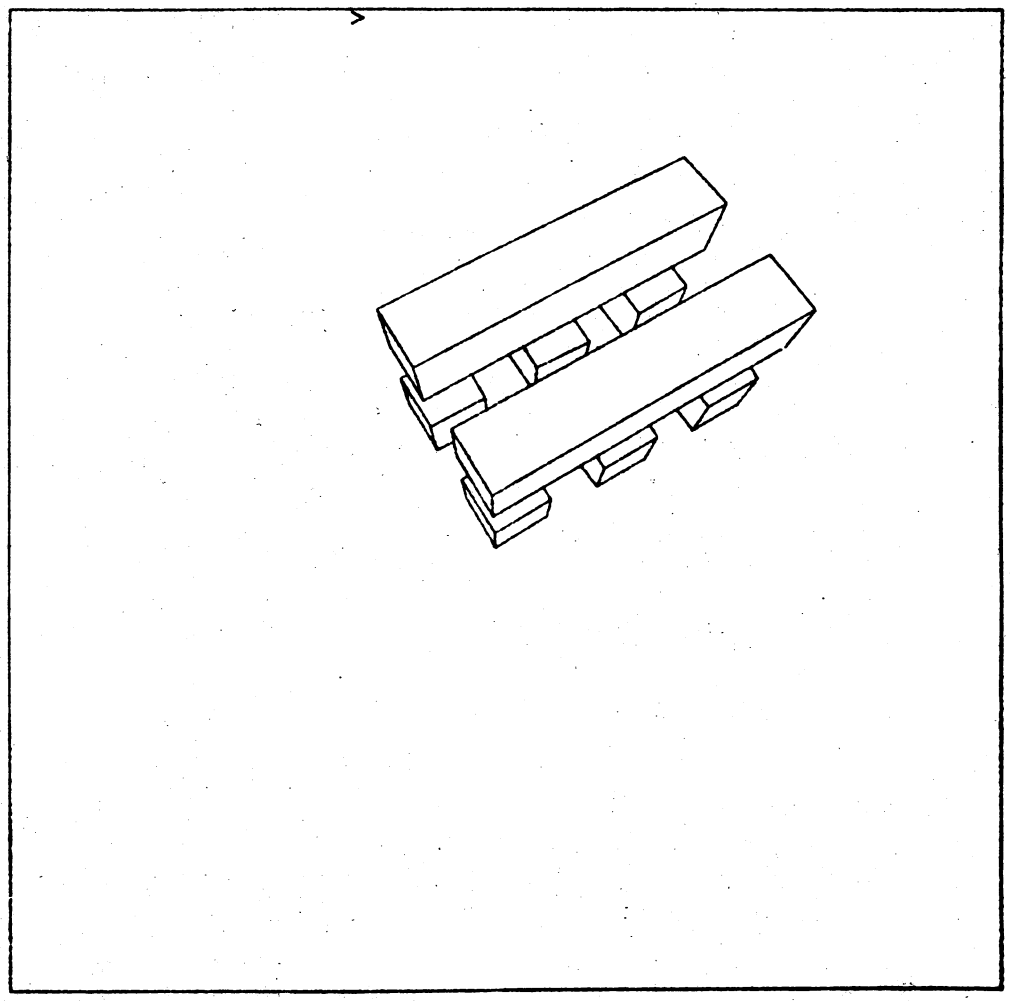


Figure 20. Eight Cubes and Rectangles

SYSTEM
PDP 11/34

TIME TAKEN
116.9688

WORDS USED
5599

SIZE OF
RESOLUTION
4095

ORIGINAL NUMBER
OF VERTICES
216

ORIGINAL NUMBER
OF FACES
162

VERTICES/CELL
4

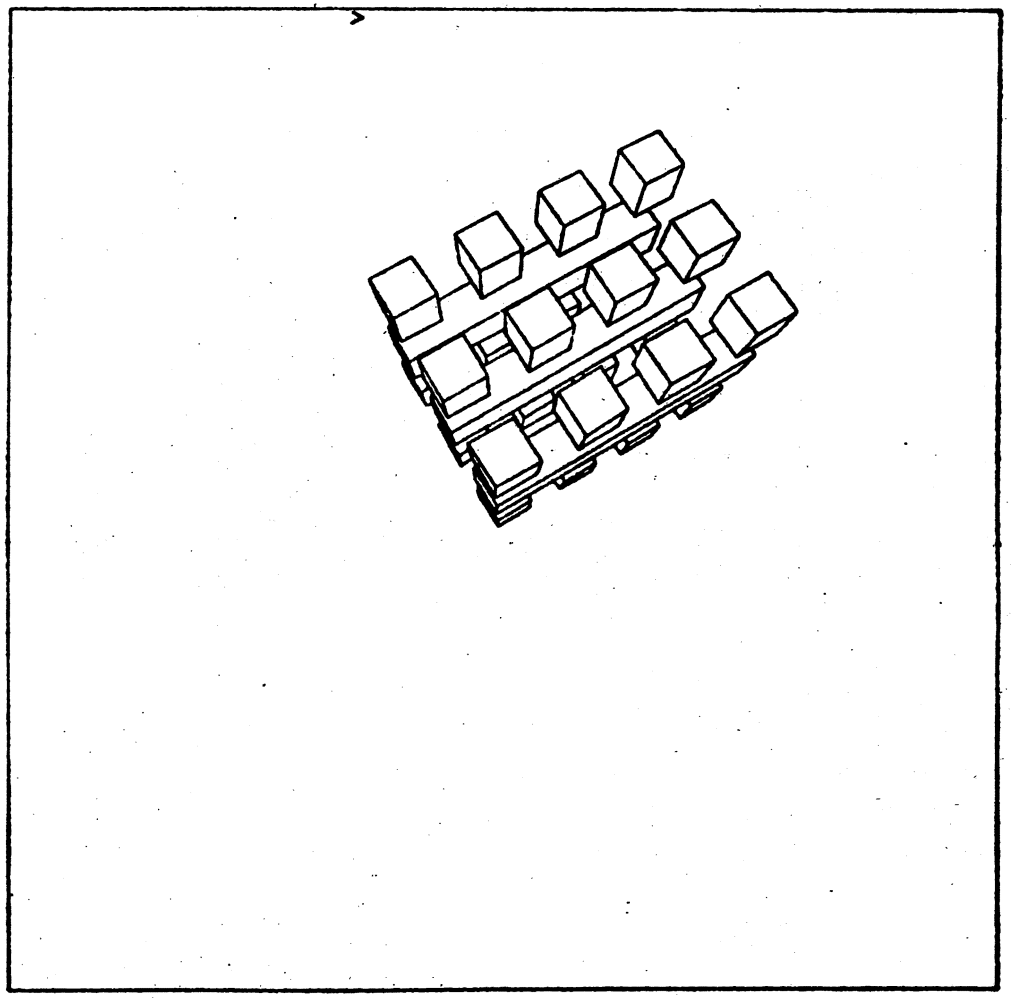


Figure 21. Twenty-seven Cubes and Rectangles

SYSTEM
PDP 11/34

TIME TAKEN
355.8667

WORDS USED
13083

SIZE OF
RESOLUTION
4095

ORIGINAL NUMBER
OF VERTICES
480

ORIGINAL NUMBER
OF FACES
360

VERTICES/CELL
4

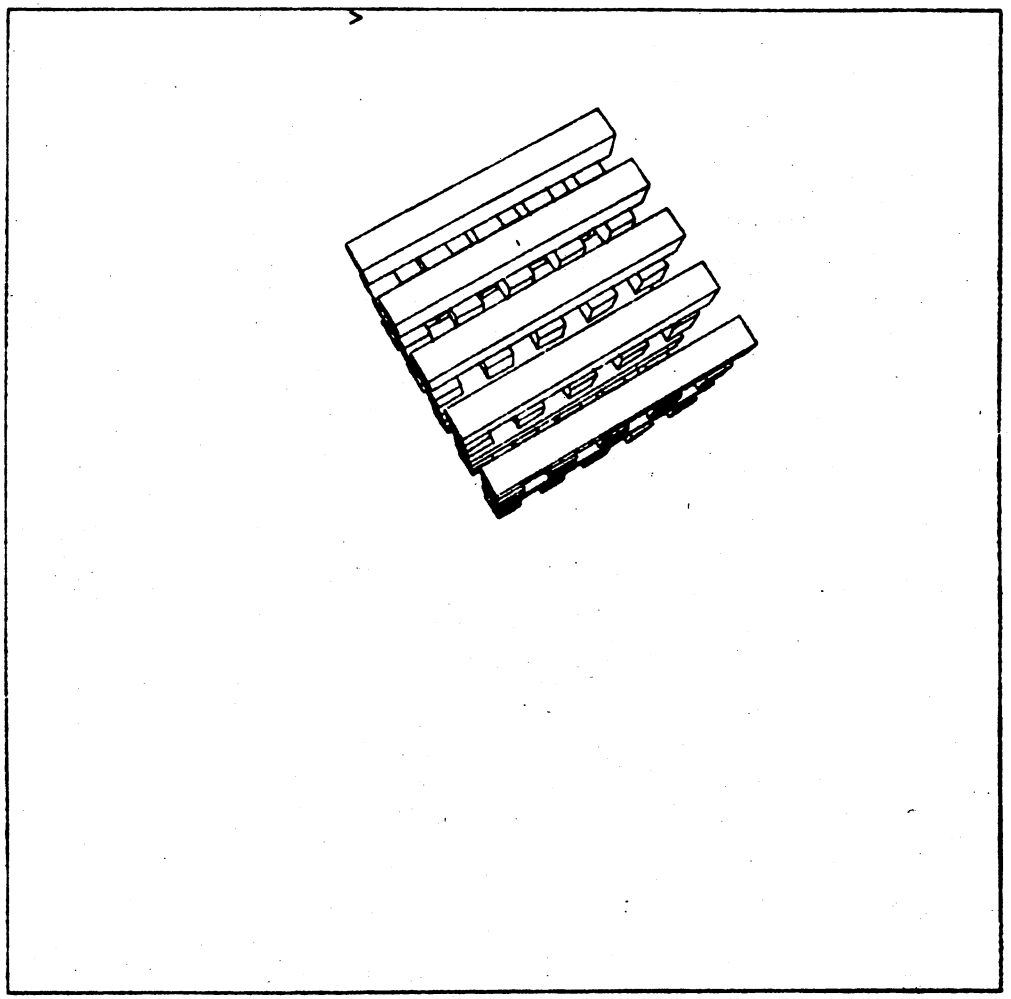


Figure 22. Sixty Cubes and Rectangles

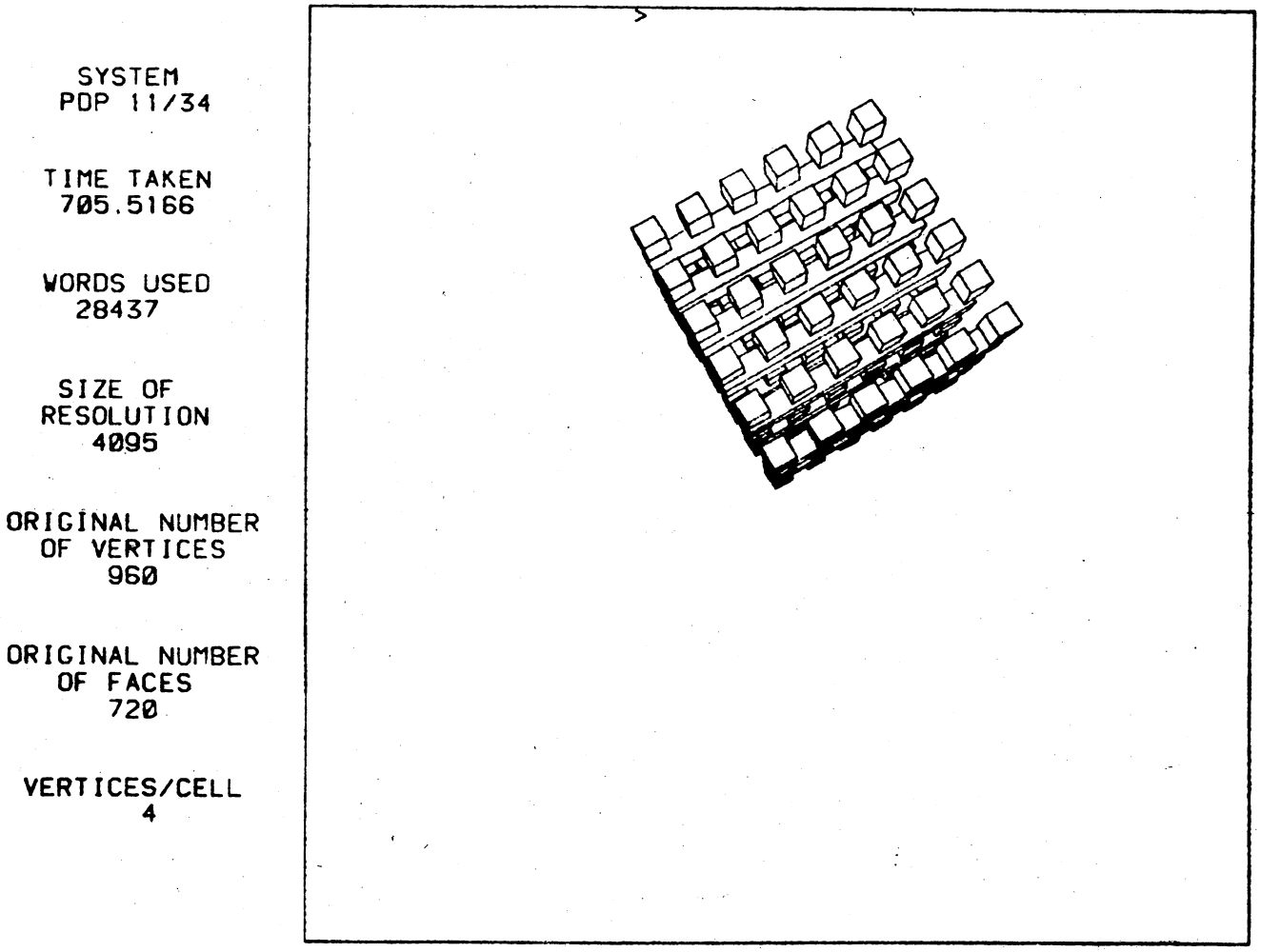


Figure 23. One Hundred Twenty Cubes and Rectangles

TABLE IX

PROGRAM PERFORMANCE FOR LATTICE OF CUBES AND RECTANGLES AT DIFFERENT ANGLES OF ROTATION

View	Griffith's Original Algorithm	New Z Depth Test
1	619.7	585.2
2	658.4	619.8

SYSTEM
PDP 11/34

TIME TAKEN
585.1816

WORDS USED
27397

SIZE OF
RESOLUTION
4095

ORIGINAL NUMBER
OF VERTICES
960

ORIGINAL NUMBER
OF FACES
720

VERTICES/CELL
4

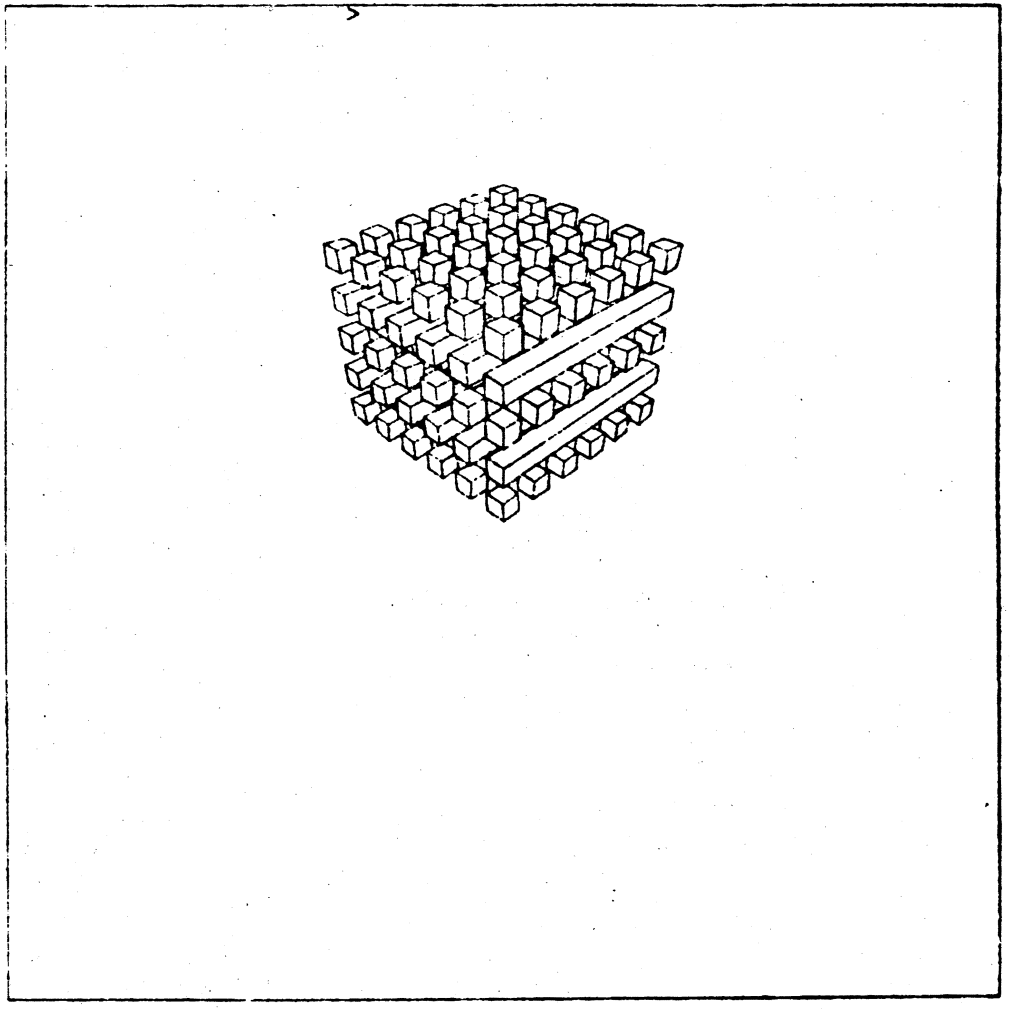


Figure 24. View One

SYSTEM
PDP 11/34

TIME TAKEN
619.8320

WORDS USED
27859

SIZE OF
RESOLUTION
4095

ORIGINAL NUMBER
OF VERTICES
960

ORIGINAL NUMBER
OF FACES
720

VERTICES/CELL
4

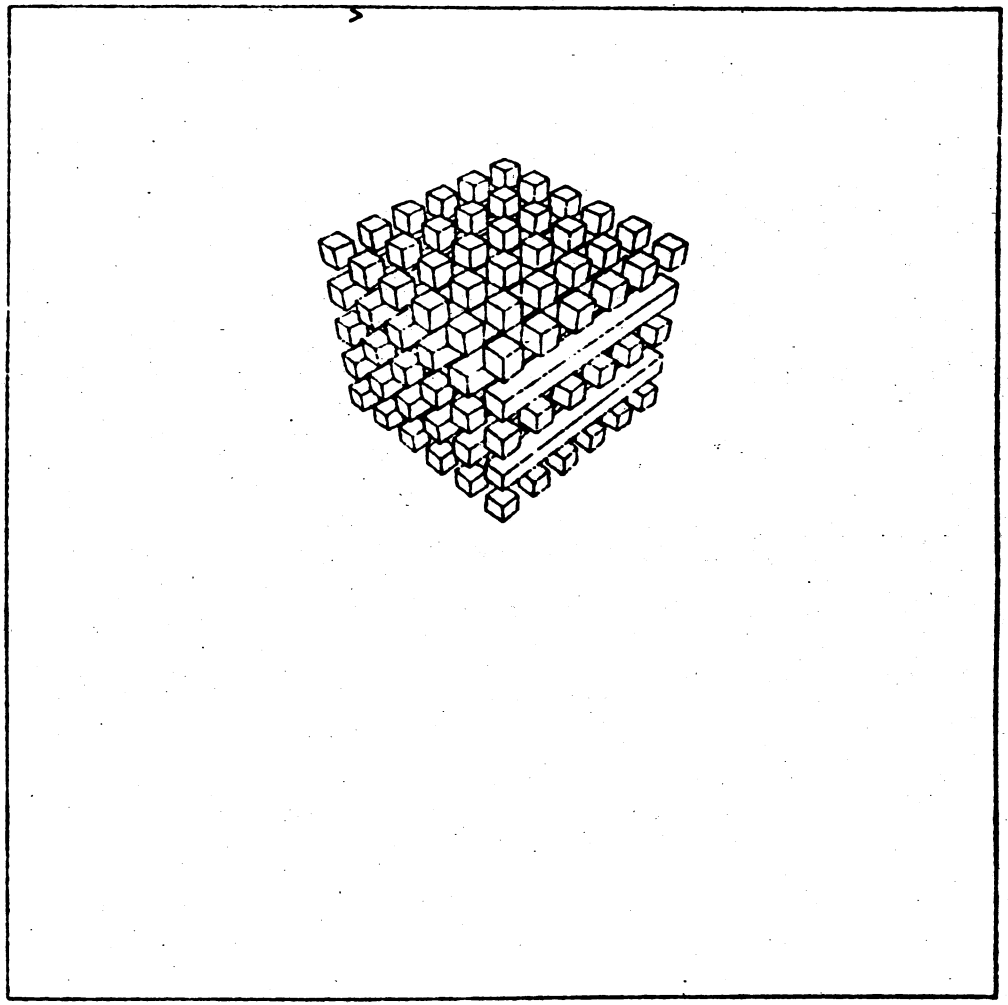


Figure 25. View Two

APPENDIX D

LISTING OF COMPUTER PROGRAM

```
C*************************************************************************
C                  HIDDEN LINE REMOVABLE
C
C          THE ORIGINAL VERSION OF THIS ALGORITHM WAS
C          DEVELOPED BY J.G. GRIFFITH AND PRESENTED IN THE ARTICLE
C          "ELIMINATING HIDDEN EDGES IN LINE DRAWINGS",COMPUTER-AIDED
C          DESIGN,VOLUME 11 NUMBER 2,MARCH 1979.
C
C          THIS VERSION, WITH ALGORITHM REVISIONS AND DOCUMENTATION, WAS
C          DEVELOPED FOR GRADUATE THESIS WORK AT OKLAHOMA STATE UNIVERSITY
C          BY WAYNE BROWN.
C          DATE:SPRING 1980
C
C*******************                                        *************
C                    ALGORITHM DESCRIPTION
C
C          PHASE 1:SET UP THE DATA STRUCTURE.
C                  PART 1:TRANSLATION, ROTATION, AND PERSPECTIVE.
C                  PART 2:VERTEX NODES.
C                  PART 3:FACE NODES.
C                  PART 4:EDGE NODES.
C                  PART 5:TEST GRID SET UP.
C
C          PHASE 2:HIDDEN LINE REMOVABLE.
C                  PART 1:SET UP AN EDGE.
C                  PART 2:COMPARE EDGE TO EACH INDIVIDUAL EDGE OF A FACE.
C                  PART 3:SORT INTERSECTIONS IN ASCENDING ORDER.
C                  PART 4:DEPTH ANALYSIS.
C                  PART 5:MERGE TWO LISTS OF INTERSECTION POINTS.
C                  PART 6:FROM MASKING LINE SAVE VISIBLE LINE SEGMENTS.
C                  PART 7:REPETE FOR EACH FACE IN EACH CELL THAT
C                         THE EDGE PASSES THRU.
C                  PART 8:OUTPUT VISIBLE LINE SEGMENTS-IF ANY.
C                  PART 9:REPEAT 1 THRU 8 FOR EACH EDGE.
C
C*******************                                        *************
C
C                    MAJOR VARIABLE LIST
C
C           S      - ARRAY FOR ALL DATA STRUCTURE AND CALULATIONS.
C                    (MAKE THIS AS BIG AS POSSIBLE)
C           NVSIZE- SIZE OF VERTEX NODE
C           NFSIZE- SIZE OF FACE NODE
C           NESIZE- SIZE OF EDGE NODE
C           BITS   - LARGEST ABSOLUTE VALUE OF INTEGERS FOR MAPPING
C           LIMIT  - RANGE OF INTERGER VALUES
C           HX     - TRANSLATION VALUE FOR THE X VALUES
C           HY     - TRANSLATION VALUE FOR THE Y VALUES
C           HZ     - TRANSLATION VALUE FOR THE Z VALUES
C           R      - RADIUS OF SPHERE THAT ENVELOPES THE OBJECT VIEWED
C           NV     - NUMBER OF VERTICES
C           NF     - NUMBER OF FACES
C           A1     - ANGLE OF ROTATION ABOUT Z AXIS
C           G1     - ANGLE OF ROTATION ABOUT X AXIS
C           A      - SCALE FOR X MAPPING
C           B      - BIAS FOR MAPPING(HALF POSITIVE - HALF NEGATIVE)
C           C      - SCALE FOR Y MAPPING
C           D      - BIAS FOR MAPPING Y(HALF POSITIVE - HALF NEGATIVE)
C           NBUCK  - NUMBER OF BUCKETS(NUMBER OF ROWS,COLUMNS IN GRID)
C           STEP   - SIZE OF EACH BUCKET(GRID SQUARE)
C           START  - BIAS FOR STARTING POINT IN THE GRID
C
C*********************                                    ***************
C
C
      IMPLICIT INTEGER*4 (I,J,K,L,M,N)
      VIRTUAL N(32767)
      DIMENSION BUFF(60)
      DATA NTI/5/,NF2/2/,NF3/3/,NF4/4/
C                          ****************************************
C                          PART 1:TRANSLATION,ROTATION,PERSPECTIVE.
C                                 FIND THE ANGLES OF ROTATION.
      WRITE(NTI,1000)
1000  FORMAT(' INPUT POINT BEING OBSERVED - F10'/
     $       ' I          I          I          I')
      READ(NTI,1001)X,Y,Z
1001  FORMAT(3F10.0)
      WRITE(NTI,1002)
1002  FORMAT(' INPUT POSITON OF OBSERVER - F10')
      READ(NTI,1001)XP,YP,ZP
      X1=XP-X
      Y1=YP-Y
      Z1=ZP-Z
      HX=-X
      HY=-Y
      HZ=-Z
      RD=ATAN(1.0)/45.0
      RXY=SQRT(X1**2 + Y1**2)
      IF(RXY.EQ.0.0) GO TO 50
      IF(X1.EQ.0.0) GO TO 10
      IF(Y1.EQ.0.0) GO TO 20
        A=ATAN(Y1/X1)/RD
        A1=90.0*(2.0-X1/ABS(X1)) + A
      GO TO 30
10      A1=90*(1.0+Y1/ABS(Y1))
      GO TO 30
20      A1=90*(2.0-X1/ABS(X1))
30    IF(Z1.NE.0.0) GO TO 40
        G1=90.0
      GO TO 60
40      G=ATAN(Z1/RXY)/RD
        G1=90-G
      GO TO 60
50      A1=0.0
        G1=0.0
60    WRITE(NTI,1003) A1,G1
1003  FORMAT(' A1',F10.5,'     G1',F10.5)
```

```
      A1=A1*RD
      G1=G1*RD
C                                  INPUT POWER OF 2 FOR RESOLUTION SIZE.
C                                  EXAMPLE: 6 - 254, 13(MAX) - 32767.
      WRITE(NTI,1004)
1004  FORMAT(' INPUT SIZE OF INTEGERS - I2')
      READ(NTI,1005) NBITS
1005  FORMAT(I2)
      WRITE(NTI,1006)
1006  FORMAT(' INPUT NUMBER OF VERTICES PER CELL - I2')
      READ(NTI,1005)NC
C                                  INITIALIZE CONSTANTS.
       BF=1.0/FLOAT(NC)
       TIME=SECNDS(0.)
       MAX=0
       NVSIZE=5
       NFSIZE=3
       NESIZE=5
       KBITS=2**NBITS-1
       LBITS=2*KBITS
       BITS=FLOAT(LBITS)
       LIMIT=2*LBITS+3
       ZF=1.0E20
       ZN=-1.0E20
       R=0.0
      OPEN (UNIT=2,NAME='OBJECT.DAT;1',TYPE='OLD',
     &       ACCESS='SEQUENTIAL',FORM='FORMATTED')
      OPEN (UNIT=3,NAME='SCRATCH1.DAT;1',TYPE='NEW',
     &       ACCESS='SEQUENTIAL',FORM='UNFORMATTED')
      OPEN (UNIT=4,NAME='SCRATCH2.DAT;1',TYPE='NEW',
     &       ACCESS='SEQUENTIAL',FORM='UNFORMATTED')
C                                  ROTATION MATRIX.
      SINA=SIN(A1)
      COSA=COS(A1)
      SING=SIN(G1)
      COSG=COS(G1)
      R1=COSA
      R2=SINA
      R3=0.0
      R4=COSG*(-SINA)
      R5=COSG*COSA
      R6=SING
      R7=(-SING)*(-SINA)
      R8=(-SING)*COSA
      R9=COSG
C                                  READ THE NUMBER OF VERTICES.
       READ(NF2,1007)NV
1007   FORMAT(20I4)
       DO 70 J1=1,NV
         READ(NF2,1008)X,Y,Z
 1008    FORMAT(5F16.0)
C                                  TRANSLATION.
         X=X+HX
         Y=Y+HY
         Z=Z+HZ
         RM=X**2 + Y**2 + Z**2
         IF(RM.GT.R) R=RM
C                                  ROTATION.
         TX=R1*X + R2*Y + R3*Z
         TY=R4*X + R5*Y + R6*Z
         TZ=R7*X + R8*Y + R9*Z
         WRITE (NF3) TX,TY,TZ
         IF(TZ.GT.ZN)ZN=TZ
70       IF(TZ.LT.ZF)ZF=TZ
       REWIND 3
       R=SQRT(R)
C                                  HALF ANGLE OF VIEW FOR PERSPECTIVE.
      ANG=R/SQRT(X1**2 + Y1**2 + Z1**2)
      IF(ANG.LT.1.0) GO TO 80
        WRITE(NTI,1009) ANG
1009    FORMAT(' ANGLE=',F10.6,'   TOO CLOSE TO THE OBJECT')
        STOP
80    H=R*(1.0-ANG)
C                                  LINEAR MAPPING FOR THE Z.
       A=BITS/(ZN-ZF)
       B=-A*(ZN+ZF)*0.5+0.5
       XL=R
       XR=-R
       YU=-R
       YD=R
       DO 90 J1=1 , NV
         READ (NF3)   R1,R2,R3
C                                  PERSPECTIVE.
         R4=H / (R-R3*ANG)
         R1=R1*R4
         R2=R2*R4
         R3=A*R3+B
         WRITE (NF4) R1,R2,R3
C                                  FIND RANGE OF X AND Y FOR MAPPING.
         IF(R1.LT.XL) XL=R1
         IF(R1.GT.XR) XR=R1
         IF(R2.GT.YU) YU=R2
         IF(R2.LT.YD) YD=R2
90     CONTINUE
       H=H/SQRT(1.0-ANG*ANG)
C                                  SCALE FOR FINAL DRAWING.
       BUFF(1)= 2.0*H+(Z1-R)*0.25
       NBP=5
       REWIND 3
       REWIND 4
C                                  LINEAR MAPPING FOR X AND Y.
       A=BITS/(XR-XL)
       B=-A*(XL+XR)*0.5+0.5
       C=BITS/(YU-YD)
       D=-C*(YU+YD)*0.5+0.5
C                          ****************************
C                          PART 2:SET UP VERTEX NODES.
       J1=1-NVSIZE
```

```
      J2=2-NVSIZE
      J3=3-NVSIZE
      J4=4-NVSIZE
      J5=0
      DO 100 J9=1,NV
        J1=J1+NVSIZE
        J2=J2+NVSIZE
        J3=J3+NVSIZE
        J4=J4+NVSIZE
        J5=J5+NVSIZE
        READ (NF4)  R1,R2,R3
        N(J1)=A*R1+B
        N(J2)=C*R2+D
        N(J3)=R3
        N(J4)=0
        N(J5)=0
100   CONTINUE
C                                  REVERSE MAPPING CONSTANTS.
      A=1.0/A
      B=-(B-0.5)*A
      C=1.0/C
      D=-(D-0.5)*C
C                                  SET UP TEST GRID.
      NBUCK=SQRT(BF*FLOAT(NV))
      IF(MOD(NBUCK,2).NE.0)NBUCK=NBUCK+1
      STEP=FLOAT(NBUCK)/BITS
      START=-0.5*BITS*(1.0+1.0/FLOAT(NBUCK))
      NBUCK=NBUCK+1
      K1=NV*NVSIZE+1
      K2=K1+NBUCK*NBUCK-1
C                                  INITIALIZE GRID TO ZEROES.
      DO 110 J1=K1,K2
110     N(J1)=0
C                                  ASSIGN EACH VERTEX TO AN APPROPRIATE
C                                  CELL LIST.
      J1=K2-1
      J2=K2
      J3=K1+1-NVSIZE
      DO 120 J4=1,J3,NVSIZE
        J5=(FLOAT(N(J4))-START)*STEP
        J6=(FLOAT(N(J4+1))-START)*STEP
        IF(J6/2*2.NE.J6) J5=NBUCK-1-J5
        J7=J6*NBUCK+J5+K1
        J1=J1+2
        J2=J2+2
        N(J2)=N(J7)
        N(J7)=J1
120     N(J1)=J4
C
C                                    MINIMIZE FINAL DRAWING PEN
C                                    MOVEMENT - LINK VERTICES.
      J32=0
      DO 140 J1=K1,K2
        J2=N(J1)
        IF(J2.EQ.0) GOTO 140
130       J3=N(J2)
          N(J3+3)=J32
          J32=J3
          J2=N(J2+1)
          IF(J2.NE.0) GOTO 130
140   CONTINUE
C                                    PREHAVE SCREEN AREA FOR SORTING FACES
C                                    BY DEPTH.
      NFS=K2+1
      J1=(NFS-K1)/2
      J2=K1+2*(J1-1)
      DO 150 J3=K1,J2,2
        N(J3)=-LIMIT
150     N(J3+1)=J3
      R1=FLOAT(1-J1)/BITS
      R2=FLOAT(J1)*0.5
C                          ****************
C                          PART 3:FACE NODE.
C                                READ NUMBER OF FACES.
      READ(NF2,1007)NF
      DO 240 J3=1,NF
        J4=NFS
        J6=NFS+NFSIZE
        J8=J6+20
        J5=J6-1
        J7=J8-1
        READ(NF2,1007)(N(J9),J9=J6,J7)
        JV1=(N(J6)-1)   * NVSIZE +1
        JV2=(N(J6+1)-1) * NVSIZE +1
        JV3=(N(J6+2)-1) * NVSIZE +1
C                                REMOVE BACK FACES.
        IF(FLOAT(N(JV1)-N(JV2)) * FLOAT(N(JV2+1)-N(JV3+1)) -
     &    FLOAT(N(JV2)-N(JV3)) * FLOAT(N(JV1+1)-N(JV2+1))
     &    .GE. 0.0) GO TO 10
        N(J8)=0
        J7=J6
        IZN=-LIMIT
        J8=N(J7)
C                                FIND THE NEAREST POINT ON THE FACE.
160       J8=(J8-1)*NVSIZE+1
          N(J7)=J8
          J8=N(J8+2)
          IF(J8.GT.IZN) IZN=J8
          J7=J7+1
          J8=N(J7)
          IF(J8.NE.0) GOTO 160
        N(J4)=IZN
        N(J4+2)=J7-J6
        NFS=J7
        J7=J7-1
C                                BUCKET ADDRESS FOR FACE.
        J9=R1*FLOAT(IZN)+R2
        J9=2*J9+K1
```

```
C                                   SEARCH BUCKET TO PLACE THE FACE.
170       J8=J9
          J9=N(J9+1)
          IF(N(J9).GT.IZN) GOTO 170
        N(J8+1)=J4
        N(J4+1)=J9
        J9=N(J7)
C                                   EDGES.
        DO 230 J10=J6,J7
          J8=J9
          J9=N(J10)
          IF(J8.GT.J9) GOTO 180
            J11=J8
            J12=J9
          GO TO 190
180         J11=J9
            J12=J8
190       J13=N(J12+4)
          IF(J13.EQ.0) GOTO 220
200         IF(N(J13).EQ.J11) GOTO 210
              J13=N(J13+3)
              IF(J13.NE.0) GOTO 200
              GOTO 220
210         N(J13+2)=J4
            GOTO 230
220       N(NFS)=J11
C                           ********************
C                           PART 4:EDGE NODES.
          N(NFS+1)=J4
          N(NFS+2)=J4
          N(NFS+3)=N(J12+4)
          N(J12+4)=NFS
          NFS=NFS+NESIZE
230     CONTINUE
240     CONTINUE
        J6=0
C                                   TRACE THRU THE FACES AND
C                                   LINK TOGETHER.
        DO 270 J3=K1,J2,2
          J4=N(J3+1)
          IF(J4.EQ.J3) GOTO 260
250         J5=N(J4+1)
            N(J4+1)=J6
            J6=J4
            J4=J5
            IF(J4.NE.J3) GOTO 250
260       N(J3)=0
270       N(J3+1)=0
        N(K2)=0
        J1=J6
C                                   PLACE EACH FACE IN ITS APPROPRIATE
C                                   CELL LIST.
280       IXL=LIMIT
          IXR=-LIMIT
          IYU=-LIMIT
          IYD=LIMIT
          J2=J1+NFSIZE
          J3=J2+N(J1+2)-1
C                                   FIND THE RANGE OF THE FACE.
          DO 290 J4=J2,J3
            J5=N(J4)
            J6=N(J5)
            J7=N(J5+1)
            IF(J6.LT.IXL) IXL=J6
            IF(J6.GT.IXR) IXR=J6
            IF(J7.LT.IYD) IYD=J7
290         IF(J7.GT.IYU) IYU=J7
C                                   COORDINATES OF GRID CELLS.
          IXL=(FLOAT(IXL)-START)*STEP
          IXR=(FLOAT(IXR)-START)*STEP
          IYU=(FLOAT(IYU)-START)*STEP
          IYD=(FLOAT(IYD)-START)*STEP
C                                   MAP INTO GRID.
          J6=IYD*NBUCK+1XL+K1
          J7=J6-IXL+1XR
C                                   PUT POINTER TO FACE IN EACH
C                                   CELL LIST.
          DO 310 J8=IYD,IYU
            DO 300 J9=J6,J7
              N(NFS)=J1
              N(NFS+1)=N(J9)
              N(J9)=NFS
300           NFS=NFS+2
            J6=J6+NBUCK
310         J7=J7+NBUCK
          J1=N(J1+1)
          IF(J1.NE.0) GOTO 280
        CLOSE (UNIT=3,DISP='SAVE')
        CLOSE (UNIT=4,DISP='DELETE')
C
C                       DATA STRUCTURE COMPLETELY SET UP
C
C************************                                 ************************
C
C                       BEGIN HIDDEN LINE COMPARSIONS.
C
        OPEN (UNIT=4,NAME='DRAW.DAT;1',TYPE='UNKNOWN',
     &        ACCESS='SEQUENTIAL',FORM='UNFORMATTED')
        K=NFS
        J1=0
C                                   J2 POINTS TO A STARTING VERTEX.
        J2=J32
C                       ***  LOOP 1  ***
C                            DO FOR EVERY STARTING VERTEX.
C                                   J3 POINTS TO A EDGE NODE.
320     J3=N(J2+4)
C                                   IF THERE ARE NO MORE EDGES WITH
C                                   THIS STARTING VERTEX THEN GO TO
```

```
C                                   THE NEXT STARTING VERTEX.
      IF(J3.EQ.0) GOTO 670
C                                   STARTING VERTEX X1,Y1,Z1.
      E1=N(J2)
      E2=N(J2+1)
      E3=N(J2+2)
C                         *** LOOP 2 ***
C                             DO FOR EVERY EDGE WITH THIS STARTING VERTEX.
C                                   EDGE MODE FOR CURRENT EDGE.
330   J4=N(J3)
      J5=N(J3+1)
      J6=N(J3+2)
C                                   MARKER TO MARK FACES TO AVOID
C                                   DUPLICATE COMPARISONS.
      J1=J1-1
      N(J5+1)=J1
      N(J6+1)=J1
      NFS=K
C                                   ENDING VERTEX X2,Y2,Z2.
      E4=N(J4)
      E5=N(J4+1)
      E6=N(J4+2)
      XF=E4
      YF=E5
      ZF=E6
C                                   CALCULATE THE CELL ADDRESS IN THE
C                                   GRID FOR THE ENDING VERTEX.
      J4=(E4-START)*STEP
      J7=(E5-START)*STEP
      J6=J7*NBUCK+J6+K1
      E4=E4-E1
      E5=E5-E2
      E6=E6-E3
      K3=NFS
      K4=NFS+2
      NFS=NFS+3
C                                   INITIALIZE MASKING LINE.
      N(K3)=0
      N(K3+1)=LBITS
      N(K4)=LIMIT
C                                   SET CONSTANTS FOR OBTAINING
C                                   CELLS ALONG THE EDGE.
      Z6=E3
      KYS=1
      KAS=1
      IF(E2.LE.YF) GO  TO 340
        KYS=0
        KAS=-1
340   KBS=1
      KXS=1
      IF(E1.LE.XF) GO TO 350
        KBS=-1
        KXS=0
350   JC7=(E1-START)*STEP
      JC8=(E2-START)*STEP
      XD=FLOAT(JC7+KXS)/STEP + START
      YD=FLOAT(JC8+KYS)/STEP + START
      IF(E4.NE.0.0) GO TO 360
        XL1=BITS
        YL1=(YD-E2)/(YF-E2)
        GO TO 370
360   IF(E5.NE.0.0) GO TO 365
        YL1=BITS
        XL1=(XD-E1)/E4
        GO TO 370
365   XL1=(XD-E1)/E4
      YL1=(YD-E2)/E5
370   R6=0.0
      KTX=KXS+KBS
      KTY=KYS+KAS
C                               *** LOOP 3 ***
C                                   DO FOR EVERY CELL ALONG AN EDGE.
380   R5=R6
      IF(XL1.LT.YL1) GO TO 390
        R6=YL1
        YD=FLOAT(JC8+KTY)/STEP + START
        YL1=(YD-E2)/E5
        KTY=KTY+KAS
        GO TO 400
390   R6=XL1
      XD=FLOAT(JC7+KTX)/STEP + START
      XL1=(XD-E1)/E4
      KTX=KTX+KBS
C                                       GET THE NEXT CELL ON THIS EDGE.
400   R7=(R5+R6)*0.5
      J7=(E1+R7*E4-START)*STEP
      J8=(E2+R7*E5-START)*STEP
      J7=J8*NBUCK+J7+K1
      J8=N(J7)
      Z5=Z6
      IF(R6.GT.1.0)R6=1.0
      Z6=E3+R6*E6
C                                       K5 IS THE MINIMUM Z ON THE EDGE.
      K5=Z5
      IF(Z6.LT.Z5)K5=Z6
C                               *** LOOP 4 ***
C                                   DO FOR EVERY FACE IN THIS CELL.
C                                       BEGIN COMPARING FACE EDGES TO THE
C                                       EDGE - ONE AT A TIME.
410   J9=N(J8)
      J10=N(J9)
C                                       IF THE MAXIMUM Z ON THIS FACE IS
C                                       FARTHER AWAY THAT THE SMALLLEST Z
C                                       ON THE EDGE THEN THIS FACE CANNOT
C                                       HIDE ANY PART OF THE EDGE - GO TO
C                                       THE NEXT CELL FOR MORE FACES.
      IF(J10.LE.K5) GOTO 630
C                                       IF FACE IS MARKED THEN NO COMPARISON.
```

```
      IF(N(J9+1).EQ.J1) GOTO 620
C                                  MARK THE FACE.
      N(J9+1)=J1
      J10=J9+3
      J11=J10+N(J9+2)-1
      J12=NFS
      J13=NFS-1
C                                  INITIALIZE SUMS FOR LEAST SQUARE FIT.
      R7=0.0
      R8=0.0
      R9=0.0
      R10=0.0
      J16=N(J11)
C                                  COMPARE EACH EDGE OF THIS FACE.
      DO 430 J17=J10,J11
        J15=J16
        J16=N(J17)
        T7=FLOAT(N(J15))
        T8=FLOAT(N(J15+1))
        T10=FLOAT(N(J16))-T7
        T11=FLOAT(N(J16+1))-T8
        T9=T7-E1
        T12=T8-E2
        T13=T10*T12-T9*T11
        T14=E4*T12-E5*T9
        T15=E5*T10-E4*T11
        IF(T15.GT.0.0)GO TO 420
          T13=-T13
          T14=-T14
          T15=-T15
420     IF(T14.LT.0.0 .OR. T14.GT.T15 .OR. T15.EQ.0.0) GO TO 430
C                                  SOLVE FOR INTERSECTION USING
C                                  CRAMER'S RULE.
        R11=T13/T15
        R12=T14/T15
        R13=FLOAT(N(J15+2)) + R12*FLOAT(N(J16+2)-N(J15+2))
C                                   LEAST SQUARE FIT SUMS.
        R7=R7+R11*R11
        R8=R8+R11
        R9=R9+R13*R11
        R10=R10+R13
        F20=FLOAT(N(K3))
        F21=FLOAT(N(K4-1))
        F22=R11*BITS+0.5
        IF(F22.LT.F20)F22=F20
        IF(F22.GT.F21)F22=F21
        J13=J13+1
        N(J13)=F22
430   CONTINUE
C
C                                  IF NO INTERSECTIONS OCCURED -
C                                  GO TO NEXT FACE IN THIS CELL.
      IF(J13.LE.J12) GOTO 620
      J14=J13-1
C                                  SORT INTERSECTION POINTS IN
C                                  ASCENDING ORDER - SELECTION SORT.
      DO 450 J15=J12,J14
        J16=J15
        J17=N(J15)
        J18=J15+1
        DO 440 J19=J18,J13
          IF(N(J19).GE.J17) GOTO 440
          J16=J19
          J17=N(J19)
440     CONTINUE
        N(J16)=N(J15)
450     N(J15)=J17
C
C                                  ARE VALUES OF INTERSECTIONS
C                                  WITHIN THE END POINTS OF THE
C                                  MASKING LINE? - IF NOT GO TO
C                                  THE NEXT FACE.
      IF(N(J13).LE.N(K3) .OR. N(J12).GE.N(K4-1)) GO TO 620
C                                  LEAST SQUARES FIT.
      R11=FLOAT(J13-J12+1)
      R12=R9*R11-R8*R10
      R13=R7*R10-R8*R9
      R14=R7*R11-R8*R8
      IF(R14.GE.0.0)GOTO 460
        R12=-R12
        R13=-R13
        R14=-R14
460   J14=J13+1
      J15=J14
      R15=0.0
      J19=N(J12)
      J12=J12+1
C                                  LOOK AT EACH EDGE SEGMENT -
C                                  COUNT NUMBER OF INTERSECTIONS TO
C                                  DETERMINE WHETHER EDGE SEGEMENT IS
C                                  POSSIBLY HIDDEN.
      DO 510 J20=J12,J13
        J18=J19
        J19=N(J20)
        IF(J18.EQ.J19) GOTO 510
        TEMP=FLOAT(J18+J19)*(R12-R14*E6)
     &   +2.0*BITS*(R13-R14*E3)
        IF(TEMP.LE.0.0)GO TO 510
          F21=(J18+J19)/2
          F22=E1+(F21*E4)/BITS
          F23=E2+(F21*E5)/BITS
470       R15=R15+1.0
          IF(R15.GT.6.5) GOTO 510
            F24=BITS*COS(R15)
            F25=BITS*SIN(R15)
            J26=0
            J28=FLOAT(N(J11))
            F29=FLOAT(N(J28))
            F30=N(J28+1)
            DO 490 J32=J10,J11
```

```
            F27=F29
            F28=F30
            J30=N(J32)
            F29=N(J30)
            F30=N(J30+1)
            F33=F29-F27
            F34=F30-F28
            F35=F27-F22
            F36=F28-F23
            J37=(F36*F33-F34*F35)/BITS + 0.5
            J38=(F24*F36-F25*F35)/BITS + 0.5
            J39=(F25*F33-F24*F34)/BITS + 0.5
            IF(J39.GE.0) GOTO 480
              J37=-J37
              J38=-J38
              J39=-J39
480         IF(J38.LT.0 .OR. J38.GT.J39) GOTO 490
            IF(J38.EQ.0 .OR. J38.EQ.J39)GO TO 470
          IF(J37.LT.0) GOTO 490
          IF(J37.EQ.0) GOTO 500
          J26=J26+1
490     CONTINUE
        IF(J26/2*2.EQ.J26) GO TO 510
500       N(J15)=J18
          N(J15+1)=J19
          J15=J15+2
510   CONTINUE
C                                 NO INTERACTION - GO TO NEXT FACE.
      IF(J15.EQ.J14) GOTO 620
      N(J15)=LIMIT
      NFS=J15+1
C                                 MERGE THE TWO INTERSECTION LISTS.
      J18=K3
      J19=J14
      J16=NFS
      J17=NFS-1
520   IF(N(J18)-N(J19)) 530,540,550
530     J17=J17+1
        N(J17)=-1-N(J18)
        J18=J18+1
      GOTO 520
540     IF(N(J18).EQ.LIMIT)GOTO 560
        N(J17+1)=-1-N(J18)
        J17=J17+2
        N(J17)=N(J19)
        J18=J18+1
        J19=J19+1
      GOTO 520
550     J17=J17+1
        N(J17)=N(J19)
        J19=J19+1
      GOTO 520
560   K3=J17+1
      N(K3)=-1
      K4=K3
C                                 SET PEN SWITCHES.
      J12=0
      J13=1
      J15=1
C                                 TRAVERSE THE MASKING LINE AND EDGES.
      DO 600 J18=J16,J17
        J14=J15
        J19=N(J18)
        IF(J19.LT.0) GOTO 570
          J12=1-J12
        GO TO 580
570       J19=-J19-1
          J13=1-J13
580     J15=J12+J13
        IF(J15.EQ.2) J15=1
        IF(J14.EQ.J15) GOTO 600
          IF(J19.NE.N(K4)) GOTO 590
          K4=K4-1
        GO TO 600
590     K4=K4+1
        N(K4)=J19
600   CONTINUE
      K3=K3+1
      K4=K4+1
      N(K4)=LIMIT
C                                 MAXIMUM NUMBER OF ARRAY N USED.
      IF(K4.GT.MAX) MAX=K4
C                                 IF TRUE - EDGE HIDDEN ENTIRELY.
C                                 GO TO NEXT FACE.
      JTEST=K4-K3-2
      IF(K4-K3-2.LT.0) GOTO 660
      J14=K-1
C                                 MOVE THE FRAGMENTED EDGE BACK TO
C                                 THE POSITION OF THE ORIGINAL
C                                 EDGE FOR MORE COMPARISONS.
      DO 610 J15=K3,K4
        J14=J14+1
610     N(J14)=N(J15)
      K3=K
      K4=J14
      NFS=K4+1
620   J8=N(J8+1)
C                                 ARE THERE ANY MORE FACES IN THE CELL?
      IF(J8.NE.0) GOTO 410
C                   *** END LOOP 4 ***
C                                 ANY MORE CELLS ALONG EDGE?
630   IF(J7.NE.J6) GOTO 380
C                   *** END LOOP 3 ***
C                                 DRAW WHAT IS LEFT.
640   J12=(K4-K3)/2
      J13=K3-2
      J14=K3-1
      R12=A*E1+B
```

```
      R13=A*E4/BITS
      R14=C*E2+D
      R15=C*E5/BITS
C                                      OUTPUT EACH LINE SEGMENT.
      DO 650 J15=1,J12
        J13=J13+2
        J14=J14+2
        R16=FLOAT(N(J13))
        BUFF(NBP)=R12+R13*R16
        BUFF(NBP+1)=R14+R15*R16
        R16=FLOAT(N(J14))
        BUFF(NBP+2)=R12+R13*R16
        BUFF(NBP+3)=R14+R15*R16
        NBP=NBP+4
        IF(NBP.NE.61) GOTO 650
          WRITE (NF4)  BUFF
          NBP=1
650   CONTINUE
C                                      ARE THERE ANY MORE EDGES WITH THIS
C                                      STARTING POINT?
660   J3=N(J3+3)
      IF(J3.NE.0) GOTO 330
C                        *** END LOOP 2 ***
C                                      NEXT STARTING VERTEX.
670   J2=N(J2+3)
C                                      ARE THERE ANY MORE STARTING VERTEX?
      IF(J2.NE.0) GOTO 320
C                        *** END LOOP 1 ***
C                                      MARK END OF DRAWING.
      BUFF(NBP)=0
      BUFF(NBP+1)=0
      BUFF(NBP+2)=0
      BUFF(NBP+3)=0
      WRITE (NF4)  BUFF
      DTIME=SECNDS(TIME)
      BUFF(1)=DTIME
      BUFF(2)=MAX
      BUFF(3)=LIMIT
      BUFF(4)=NV
      BUFF(5)=NF
      BUFF(6)=NC
      WRITE(NF4) (BUFF(J),J=1,6)
      CLOSE (UNIT=4,DISP='SAVE')
      CLOSE (UNIT=2,DISP='SAVE')
      NCUBE=NF/6
      STOP
      END
```

VITA

Charles Wayne Brown

Candidate for the Degree of

Master of Science

Thesis: ANALYSIS OF A HIDDEN-LINE ALGORITHM

Major Field: Computing and Information Science

Biographical:

Personal Data: Born in Paris, Texas, October 14, 1954, the son of Mr. and Mrs. Lynn A. Brown.

Education: Graduated from Arlington High School, Arlington, Texas, in May, 1973; received Bachelor of Science degree in Architecture from the University of Texas at Arlington in December, 1977; completed requirements for Master of Science at Oklahoma State University in December, 1980.

Professional Experience: Self-employed Professional Model Builder, 1974-1977; self-employed Building and Remolding Contractor and Designer, 1976-1977; Graduate Teaching Assistant, Oklahoma State University, Department of Computer Science, 1978-1980; Computer Programmer (part time), Graphics Constructions, Inc., Tulsa, Oklahoma, 1980.