A LINEAR TIME EXACT HIDDEN SURFACE ALGORITHM *

Wm. Randolph Franklin

Electrical and Systems Engineering Dept.
Rensselaer Polytechnic Institute
Troy, NY, 12181

ABSTRACT

This paper presents a new hidden surface algorithm. Its output is the set of the visible pieces of edges and faces, and is as accurate as the arithmetic precision of the computer. Thus calculating the hidden surfaces for a higher resolution device takes no more time. If the faces are independently and identically distributed, then the execution time is linear in the number of faces. In particular, the execution time does not increase with the depth complexity.

This algorithm overlays a grid on the screen whose fineness depends on the number and size of the faces. Edges and faces are sorted into grid cells. Only objects in the same cell can intersect or hide each other. Also, if a face completely covers a cell then nothing behind it in the cell is relevant.

Three programs have tested this algorithm. The first verified the variable grid concept on 50,000 intersecting edges. The second verified the linear time, fast speed, and irrelevance of depth complexity for hidden lines on 10,000 spheres. This also tested depth complexities up to 30, and showed that perspective scenes with the farther objects smaller are even faster to calculate. The third verified this for hidden surfaces on 3000 squares.

Keywords: hidden surface, hidden line, computational geometry, geometric intersections, variable grid, computer graphics, molecular models, space-filling, algorithms analysis

CR categories: 8.2, 3.74, 5.31, 3.13.

---

* This material is based upon work supported by the National Science Foundation under grant no. ENG-7908139.

LIST OF SYMBOLS

1. $F(N) = \theta(G(N))$ means that for all $0<a<b$, there exists N_0 such that $N>N_0 \Rightarrow aG(N) \leq F(N) \leq bG(N)$.
2. $F(N)>\theta(G(N))$ and $G(N) <\theta(F(N))$ mean that for all $c > 0$, there exists N_0 such that $N > N_0 \Rightarrow F(N) >c\ G(N)$.
3. $F(N) \geq \theta(G(N))$, equivalently $G(N) \leq \theta(F(N))$, means $F(N) = \theta(G(N))$ or $F(N) \geq \theta(\overline{G}(N))$.

   Informally, 1 means F grows asymptotically as fast as G, 2 means F grows faster, and 3 means F grows at least as fast.
4. Let $\lfloor x \rfloor$ = the largest integer $\leq x$ (truncation).
5. Let B be the fineness of the variable grid, that is the number of cells along one side of the screen.
6. Let C be an unspecified constant.
7. Let D be the depth complexity of the scene.
8. Let E be an edge.
9. Let F be a face.
10. Let F_b be the blocking face of a cell.
11. Let G be a grid cell.
12. Let L be the length of a projected edge (assuming the screen is one-by-one).
13. Let N be the number of edges in the scene.
14. Let P be a point.
15. Let R be either a region of the screen corresponding to the visible part of a face, or the radius for a projected sphere.

INTRODUCTION

The hidden surface problem has been actively researched for 15 years. For an excellent summary as of 7 years ago, see Sutherland [13]. Lately,

attention has turned to photographic quality output, Blinn [3,4], Crow [5], Newell [4], and Whitted [3]. The algorithm presented here falls in both object and image space in Sutherland's classification. Although it does many calculations on the display screen, in contrast to image space algorithms, its output is exact, and its time does not increase with depth complexity (the average number of projected faces on each point on the screen). Algorithms which compare all faces that cross a scan line take at least quadratic time in the depth complexity. In contrast to other object space algorithms, it takes linear time on a large class of input scenes.

Intuitively, the reason that the depth complexity, which is related to the size of the faces, does not matter, is as follows: Either the faces are large or they are small. If they are large then they overlap so much that most of them are totally hidden. After these are quickly detected and deleted, the resulting problem is much simpler. On the other hand, if the faces are small, then each face overlaps few, if any, other faces. This, too, can be detected efficiently. Therefore, the size of the faces does not slow the algorithm.

This algorithm is similar to that of Weiler and Atherton [14], except that it is faster, but does not yet texture the faces. Since the output of this algorithm has meaning, and is not just a set of pixel intensities, it could also accommodate shadows and textures. This algorithm is similar in spirit to the computational geometry ideas of Bentley [12].

On the surface, this algorithm is similar to Warnock's. However, it has the following differences:

1. Here the one level grid is a temporary scaffolding and the visible pieces are returned whole, while in Warnock's algorithm the visible pieces are returned cut up arbitrarily by the hierarchical grid.
2. This algorithm produces exact results while Warnock's algorithm stops subdividing at about half a pixel, and doubling its resolution doubles its time.

This algorithm takes worse than linear time on scenes where the faces' positions are correlated so as to make them all intersect each other with each of them partly visible. However here the complexity of the output is $>\Theta(N)$, so the algorithm could not possibly be linear.

Knowlton [9] and Max [10] have a good hidden surface algorithm for shading with space filling molecular models.

An earlier version of this linear algorithm developed and implemented by the author in 1972-1976 is described in [6,7].

ASSUMPTIONS

The speed of this algorithm depends on several assumptions about how the nature of the scene changes as N increases.

a. The projected edges in any given scene are all the same length. This can be weakened to allow the ratio of the average edge length to the shortest length to be merely bounded above since then all edges could be cut into pieces as long as the shortest edge without changing N by more than a constant factor.

   It might seem that this assumption rules out perspectively projected scenes since the further edges are smaller. However, in practice these scenes are faster to calculate, not slower. This is because in this case a few big faces in front totally hide many small faces in the rear, and this algorithm can quickly eliminate totally hidden faces. On the other hand, if the closer edges were smaller, then this algorithm would slow down.

b. The faces' and edges' positions are independently and uniformly distributed. This is never exactly true since otherwise the probability of two faces meeting at an edge would be zero. However, these effects become relatively smaller as N grows. Even if the 3-D scene is highly correlated, as in figure 3, the projected scene is less correlated. Finally, correlation of distant objects is irrelevant since it doesn't affect their probability of intersecting, which remains zero.

c. The objects in big scenes are "similar" to those in small scenes in the sense that the same number of edges meet at a vertex, that faces do not get longer and thinner, and so on. This appears to hold for realistic scenes. A scene of N faces with length one and width 1/N would be very slow to calculate.

d. Border effects at the edge of the screen are ignored. They become progressively less important as the objects get smaller, unless the depth complexity is very large, say 100.

A NAIVE ALGORITHM

The new algorithm is a refinement on the following naive algorithm operating in time $T \geq \Theta(N^2)$:

1. Intersect all the projected edges pair by pair, to cut each edge into segments. Each segment is completely visible or else completely hidden.
2. Compare each segment against all the faces to see if it is visible.
3. Draw the visible segments and use them to divide the screen into regions, each corresponding to a visible part of a face.
4. Compare each region against all the faces to see which it corresponds to and shade it accordingly.

Note that not only any algorithm that compares all the edges will be too slow (i.e., worse than linear), but any algorithm that finds all the edge segments is too slow since the number of segments is $\Theta(N^2L^2 + N)$ which can be $> \Theta(N)$. For example, let the scene be a cubical array of cubes. If there are K^3 cubes, each of side 1/K, then $N=12K^3$ and $L=1/K$. so $L=\Theta(N^{-1/3})$. Then there are $\Theta(N^{4/3})$ line segments, and testing their visibility takes time $T=\Theta(N^{7/3})$, which is worse than quadratic.

DATA STRUCTURES

This algorithm has the following logical data structures:

1. The array of faces.
2. The array of edges.
3. A B x B grid of square cells overlaid on the projected scene. Each cell has the following three elements which are initially empty:

3a. The number of the closest blocking face, F_b, if any, of this cell. F_b's projection completely covers the cell so that nothing behind can be seen.

3b. The set of faces whose projections intersect the cell, and which are in front of F_b.

3c. The set of edges whose projections intersect the cell, and which are in front of F_b.

4. A set of pairs of intersecting edges. This set contains all the visible, and some of the hidden, intersections.
5. The set of visible line segments.
6. The set of visible regions, R. Each R is a connected visible part of one face.

THE LINEAR ALGORITHM

1. Project and scale the scene to fit a 1 by 1 square on the plotter screen. These techniques are well known. They may be found in the textbooks by Newman and Sproull [11], and Rogers and Adams [12].
2. Let $B = \lfloor c \min (\sqrt{N}, L^{-1}) \rfloor$, for $0 < c < 1$. c, which is an unspecified constant, is used to fine-tune the algorithm. Fine-tuning is discussed later.
3. Overlay a B by B grid of cells onto the scene
4. Initialize the grid cell data structures.
5. Repeat for each projected face F:

5a. Determine which cells, G, F falls partly or wholly in. This is not done by comparing each of the B^2 cells with F. Instead, the minimal box of cells enclosing F gives a good approximation. A more accurate determination can be made by considering F's edges. If extra cells are included then the algorithm will later run slower but will still give the correct output.

5b. Repeat the following for each G that contains F:

i. If G has a blocking face, F_b, determine whether, throughout G, F_b is always in front of F. False negatives are alright at this step. Thus a convenient way is to compare the distance of F from the viewpoint at the 4 corners of G with the distance of F_b at these points. Since the faces are convex and do not interpenetrate, if F is behind F_b at those 4 point, it is always behind F_b. (The converse is not always true.)

If F is behind F_b, do not consider this G further with F and go back to (i) to process the next G.

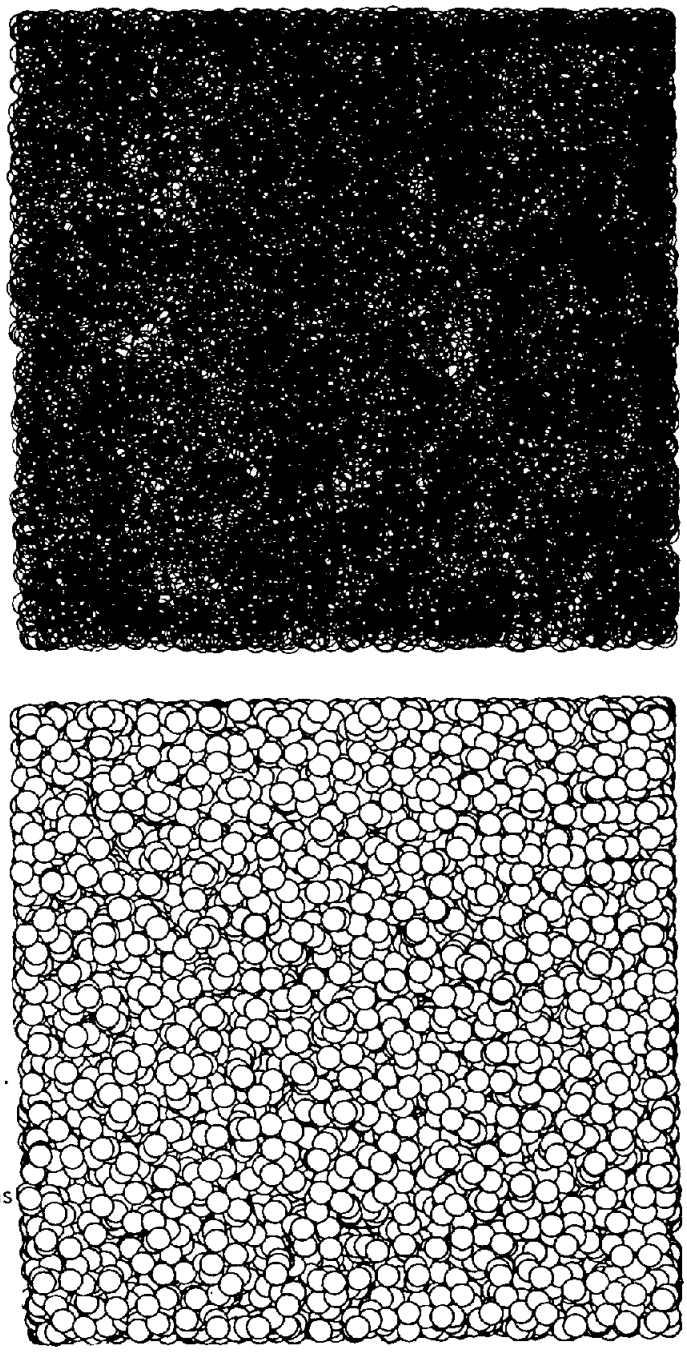

Figure 1: 10,000 spheres before and after removing the hidden lines

ii. Otherwise, determine whether F is itself a blocking face of G. This is true if and only if G's 4 corners are inside F. If F does cover G, then update F_b to be F.

iii. Otherwise add F to the set of faces in G.

5c. If F is not added to any G, then to save space delete it and its edges from the arrays since it is certainly invisible. If there are many big faces, most of them may get deleted at this point.

6. Repeat for each cell, G:

6a. Compare the blocking face, F_b, against all the faces on G's set. Delete from the set any that are behind F_b within G. The method of 5a(i) can be used. The reason that there may be faces to delete is that F_b may have been changed as the faces were processed. Thus a face, F, may have been added to G's list at a time when F was in front of the current F_b. Then later F_b was replaced with a closer face that was in front of F. F would be deleted in this step. Although we could clean out the list every time that we updated F_b, doing it all at once is faster. The initial check in 5a(i) was done to save space.

7. Repeat for each projected edge, E:

7a. Determine which cells, G, E passes through. Repeat for each G:

i. Test whether E is behind F_b. Do this by clipping E at the borders of G to E' and then testing whether the end-points of E' are both behind F_b.

ii. If not, then add E to the set of edges in G. Note that we add the complete E to the list, not its clipped version.

7b. If E is not added to any G, then delete it from the array of edges, since it is certainly invisible. However, note that a face may be partly visible even though none of its edges are.

8. Repeat for each cell, G:

8a. Repeat for each pair of edges, E_1 and E_2 in G's set of edges:

i. Test whether they intersect.

ii. If so, test whether the intersection is in G. (This catches duplications caused by the fact that the same pair of edges may pass through several cells).

iii. If so, then add E_1 to the list of edges intersecting E_2, and E_2 to the list of E_1.

However, if only the visible lines are wanted, and not the visible surfaces, do the following steps instead of (iii):

iv. Determine which one of E_1 and E_2 is in front of the other in 3-D at the point of intersection. Without loss of generality, let it be E_1.

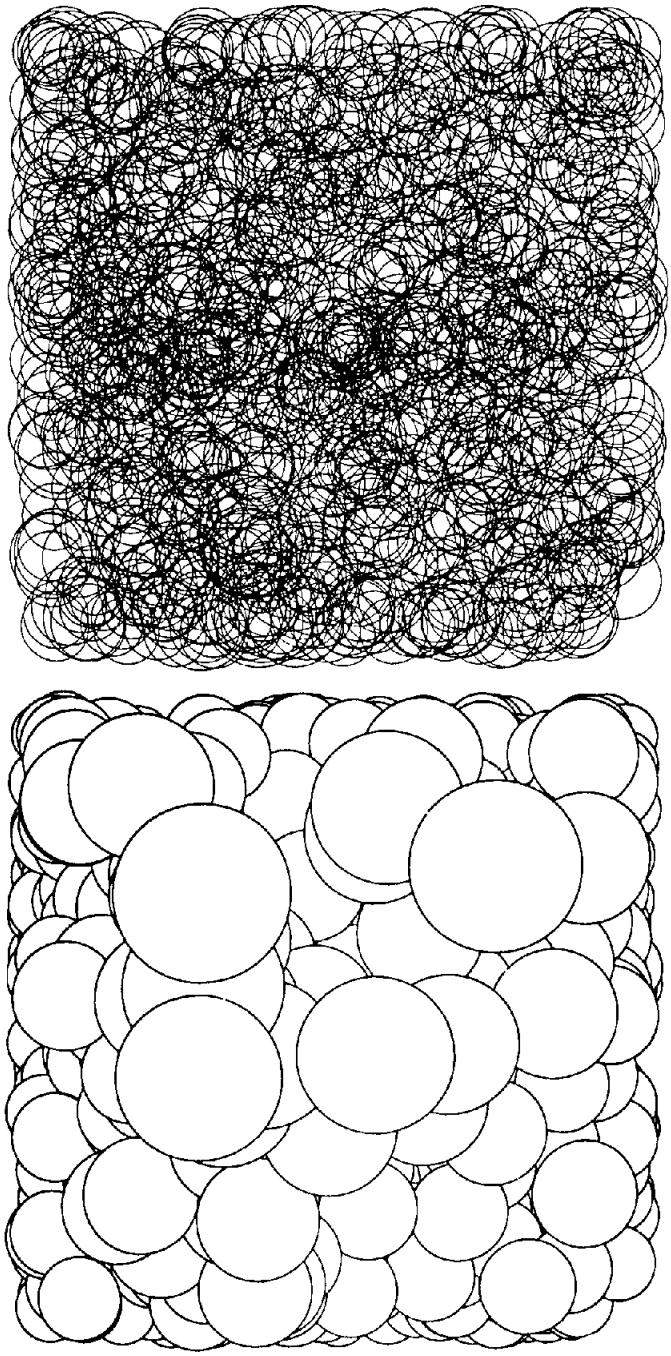

Figure 2: 1,000 spheres in perspective, before and after removing hidden lines

v. Add the intersection point to only E_2's list. This is because the visibility of E_1 is not changed by E_2's passing behind it. Nevertheless, we must still cut E_1 when we are shading if we want the visible regions to be properly defined.

9. Repeat for each edge, E:

9a. Sort the intersection points of E along E. A fast method that works regardless of E's slope and is stable against small floating-point errors in coordinates is this:

i. Let the intersections be (X_a, Y_a).

ii. Sort them by $ABS(X_a) + ABS(Y_a)$.

9b. Use the sorted points to split E into segments, S. An edge with no intersections forms one segment (unless it was previously deleted).

9c. Repeat for each segment, S:

i. Find its midpoint, P.

ii. Find which cell, G, contains P.

iii. Compare P against all the faces, F, in G to see if any hide P. F hides P if it satisfies 2 conditions: P is behind the 3-D plane of F (possibly extended) and the projected P is inside the projected F.

iv. If P is visible, then so is S, so draw S, and add it to the set of visible segments.

10. Determine the regions, R, of the planar graph formed by the visible segments, using standard algorithms.

11. Repeat for each region, R:

11a. Find a point, P, in it. The centroid will suffice except for some cases when R is concave.

11b. Find which cell, G, contains P.

11c. Compare P against the faces, F, in G. Find the closest of those faces whose projections contain P. The distance is measured by taking the length of a ray from the viewpoint towards P, possibly extended, until the 3-D plane of F. Note that in general the ordering of 2 faces in G will depend on where P is, so it is impossible to presort them by distance.

11d. R corresponds to a visible part of the closest face, F, so shade R accordingly. If P does not fall in any face, then R corresponds to background, and can be appropriately shaded. There may be more than one region per face.

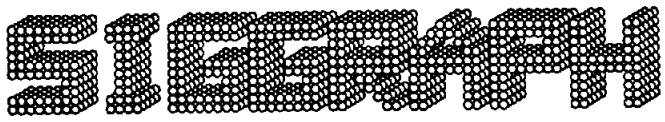

Figure 3: 1,470 spheres in a regular arrangement, with hidden lines removed

TIMING

There are 2 cases to be considered because of the definition of

$$B = \lfloor c \min (\sqrt{N}, L^{-1}) \rfloor .$$

$$\approx c \min (\sqrt{N}, L^{-1})$$

since the floor function is asymptotically unimportant.

Case 1: $L \geq \theta(N^{-1/2})$

so $B = \frac{c}{L}$.

Since the cell size is a constant fraction of the face size as N increases, the probability that a given face will completely cover a cell that it partly covers is a constant, say p. If the faces are square with sides parallel to the grid, then

$$p = \left(\frac{c-1}{c+1}\right)^2 \quad , c \geq 1 .$$

Now as the number of faces in the cell increases to infinity, the expected number of the first blocking face is

$$q = \sum_{i=1}^{\infty} i\, p\, (1-p)^{i-1}$$

$$= \frac{1}{p}$$

$$= \left(\frac{c+1}{c-1}\right)^2$$

Thus, and this is crucial to the algorithm, although the expected number of faces per cell may grow to infinity, the expected number after faces hidden by the blocking face have been removed is bounded above by q.

Let r = the expected number of cells a face falls in.

Then $r = (c+1)^2$.

Let $s = \frac{\text{number of faces}}{N}$

Then the number of faces per cell before deletions is

$$\frac{rsN}{B^2}$$

of which q faces per cell are not deleted. Thus the fraction of faces left is

$$\frac{qB^2}{rsN} = \left(\frac{c}{c-1}\right)^2 \frac{1}{sL^2N} .$$

Since the edges are distributed in 3-D as the faces are, this is also the fraction of edge segments that will not be deleted for falling behind a blocking face.

Let u = average number of cells an edge falls in

$$= c+1 .$$

Then the average number of edges per cell is

$$\frac{uN}{B^2}$$

Then the average number of edges per cell left after deletions is

$$\frac{qB^2}{rsN} \cdot \frac{uN}{B^2} = \frac{qu}{rs} = \frac{(c+1)}{(c-1)^2 s}$$

again a constant. Then intersecting the edges in the cell, pair by pair, takes constant time per cell. The total time is B^2.

Since each cell has a constant average number of edges in it, the number of intersections found is $\theta(B^2)$ so the number of segments the edges are cut into is $\theta(N + B^2)$. Testing a segment for visibility takes time proportional to the number of faces in the cell that that segment's center falls in, which is constant. Thus the total time for the hidden line part of the algorithm, obtained by summing times for the operations given above, and adding $\theta(N)$ for bookkeeping is

$$T = \theta(N + B^2)$$

$$= \theta(N)$$

Thus the hidden line algorithm is linear.

For the hidden surface algorithm, the number of visible segments is $\leq$ the number of segments = $\theta(N + B^2) = \theta(N)$. So the number of regions on the plotter screen is $\theta(N)$. Determining them takes an expected time of $\theta(N)$. Finding which face a region corresponds to takes time proportional to the number of faces in the cell being used for that region, which is constant. Thus, the total time to identify the regions is $\theta(N)$.

Case 2: $L < \theta(N^{-1/2})$

The analysis is similar to Case 1 and gives the same result. Therefore, the complete hidden surface algorithm takes time $T = \theta(N)$ in either case.

IMPLEMENTATION

Three different programs have been written to test various aspects of this algorithm. All the implementations were on a Prime 500 midicomputer with a 16 bit wordlength and 1 MB of memory. The Prime performs a single precision floating multiply in about 5 microseconds. The results were plotted either on an Imlac or an IBM 3277 graphics attachment.

The first was designed to test the concept of a variable grid. It calculated the intersections among random edges. It was tested with different numbers of edges, edge lengths and grid sizes. For example, with 10,000 edges of lengths about 0.01 (on a 1 by 1 square), and a 100 by 100 grid, it took only 77 seconds to find all 1814 intersections out of the 50 million possibilities. The number of (edge,cell) pairs was 19931. The optimal grid size was determined experimentally, and within a factor of 2 made little difference. The algorithm behaved as expected as N and L varied, except that the optimal grid size was hard to predict since it depends on the relative speeds of various parts of the program.

The second implementation, SPHERES, is described in detail in Franklin [8]. It uses the same concepts, but is a different algorithm designed to calculate hidden lines for scenes composed of spheres, as in a molecular model. SPHERES has been tested at depth complexities up to 30, and the time for fixed N even drops slightly as D increases at this point. For D = 10, SPHERES has been tested for N up to 10,000. This case took only 383 seconds to calculate. Figure 1 shows 10000 spheres packed 10 deep before and after hidden lines are removed. Figure 2 shows 1000 spheres with D = 10 in a perspective projection before and after. This case takes 1/3 less time than N = 1000, D = 10 with fixed R. Figure 3 shows the hidden lines removed from a regular array of 1970 spheres. Table 1 shows how T varies with N if D = 10. If N is fixed while D varies from .1 to 30 (by changing R), T is largest for D = 5, and declines slowly as D increases.

The third implementation removes hidden surfaces for random squares. Figure 4 shows 300 squares with the visible surfaces crosshatched at an angle proportional to their distance. This has been tested up to N = 3000 and table 2 shows that the time is still linear. This last case has D = 77.

This algorithm is now being extended to hierarchical or rotating scenes and is being incorporated into a general 3-D object manipulation package.

EXTENSIONS

It would be worthwhile to collect statistics on a scene before processing to determine the optimal B. If the scene is inhomogeneous, a hierarchical grid may be faster, although there exist sequences of scenes that still execute slowly.

ACKNOWLEDGEMENTS

Leong Shin Loong and Abel Shi Lo, while Mechanical Engineering students at RPI, helped with these implementations. Their assistance is gratefully appreciated.

REFERENCES

1. Bentley, J.L., Stanat, D.F., and Williams, E.H., Jr. The Complexity of finding fixed radius near neighbors. Info. Proc. Lett. 6, 6 (Dec. 1977), 209-212.

2. Bentley, J.L., Ottmann, T.A. Algorithms for reporting and counting geometric intersections. IEEE T. Comput. C-28, 9 (Sept. 1979), 643-647.

3. Blinn, J.F., Carpenter, L.C., Lane, J.M., and Whitted, T. Scan Line methods for displaying parametrically defined surfaces. Comm. ACM 23, 1 (Jan. 1980), 23-24.

4. Blinn, J.F., Newell, M.E. Texture and reflection in computer generated images. Comm. ACM 19, 10 (Oct. 1976), 542-547.

5. Crow, F.C. Shadow algorithms for computer graphics. Computer Graphics 11, 2 (Summer 1977), 242-248.

6. Franklin, W.R. VIEWPLOT summary, program logic manual, and user's manual. Harvard Lab for Computer Graphics, (July, Dec. 1976).

7. Franklin, W.R. Combinatorics of hidden surface algorithms. Harvard University Center for Research in Computing Technology, TR12-78, (June 1978).

8. Franklin, W.R. An Exact hidden sphere algorithm that operates in linear time. Computer Graphics and Image Processing, to appear.

9. Knowlton, K., and Cherry, L. ATOMS - a 3-D opaque molecular system. Computers and Chemistry 1, 3 (1977), 161-166.

10. Max, N.L. ATOMLLL - ATOMS with shading and highlights. Computer Graphics 13, 2 (Aug. 1979), 165-173.

11. Newman, W., and Sproull, R.F. Principles of Interactive Computer Graphics, 2nd edition. McGraw-Hill, 1979.

12. Rogers, D.F. and Adams, J.A. Mathematical Elements for Computer Graphics. McGraw-Hill, 1976.

13. Sutherland, I.E., Sproull, R.F., and Schumacker, R.A. A Characterization of ten hidden surface algorithms. Computing Surveys 6, 1 (Mar. 1974), 1-55.

14. Weiler, K., and Atherton, P. Hidden surface removal using polygon area sorting. Computer Graphics 11, 2 (Summer 1977), 214-222.

Table 1: Times for hidden spheres calculations for D = 10

N	R	B	T
30	0.326	7	4.1
100	0.178	13	5.6
300	0.03	23	10.3
1,000	0.0564	42	31.1
3,000	0.0326	72	97.5
10,000	0.0178	142	383.

Table 2: Times for hidden surface calculations for squares

N	L	B	D	T
1,000	0.16	12	25.6	527
3,000	0.16	12	76.8	1792

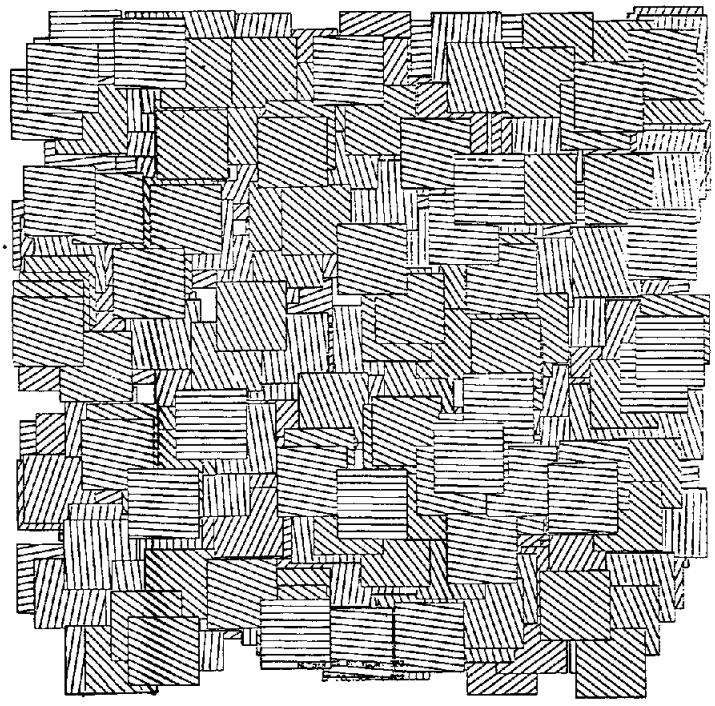

Figure 4: 300 squares with visible surfaces shaded