COMPUTER GRAPHICS AND IMAGE PROCESSING 15, 364–379 (1981)

An Exact Hidden Sphere Algorithm That Operates in Linear Time*

W. RANDOLPH FRANKLIN

Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, New York, 12181

Received March 12, 1980

This paper presents an exact hidden line algorithm, SPHERES, that operates in linear expected time. The algorithm is exact because it works in object space, and so calculates the visible lines accurately to within the floating point tolerance of the host machine. In contrast to all image space algorithms, SPHERES' calculations take no more time for a high-resolution display. SPHERES operates on a 3-D scene composed of spheres, as in a space-filling molecular model, but nothing in the underlying concepts depends on this. In contrast to all other hidden line algorithms, SPHERES' execution time does not depend much on the depth complexity of the scene, even if it is as high as 30. This has been verified experimentally, as has the linear time dependence on the number of input spheres, up to 10,000.

1. INTRODUCTION

This paper considers a central part of the problem of computer generated space filling molecular models. This is the problem of determining which parts of which atoms are visible. (Other important parts include the realistic shading, the chemistry knowledge, and the convenient user interface). Although current algorithms can take more time in the shading than in the hidden surface calculation, since the latter takes quadratic time in the number of atoms, it eventually dominates the total time as the molecules get more complex.

To see what this algorithm does, consider Fig. 1 which shows 10,000 random spheres overlaid to an average depth of 10. There are so many lines that they run together. Figure 2 shows the same 10,000 spheres with the hidden lines removed.

This paper abstracts the problem by considering a 3-D scene of spheres that is projected onto a plane. Since the spheres project into circles, the problem reduces to that of calculating overlapping circles on a plane. The original spheres are assumed to be the same size, for simplicity. Nevertheless, since the projection may be perspective, the circles in the back will be smaller. See Figs. 3 and 4 which show 1000 random spheres in perspective, overlaid to an average depth of 10, with and without the hidden lines.

Various complications, such as intersections in 3-space between the spheres and cylindrical bonds between them, are not considered since the techniques are known [5, 10, 12, 15, 16, 18, 19] and conceptually easy to integrate, but messy. They would not change the algorithm's linear time behavior.

SPHERES was developed from an investigation into the theoretical foundations of hidden surface algorithms [6], of which molecular models are only a special case, but are important enough to consider separately. These same techniques, when applied to another special case, the 3-D prism map in cartography, led the author

*This material is based upon work supported by the National Science Foundation under Grants ENG-7908139 and ISP-7920240.

0146-664X/81/040364-16$02.00/0

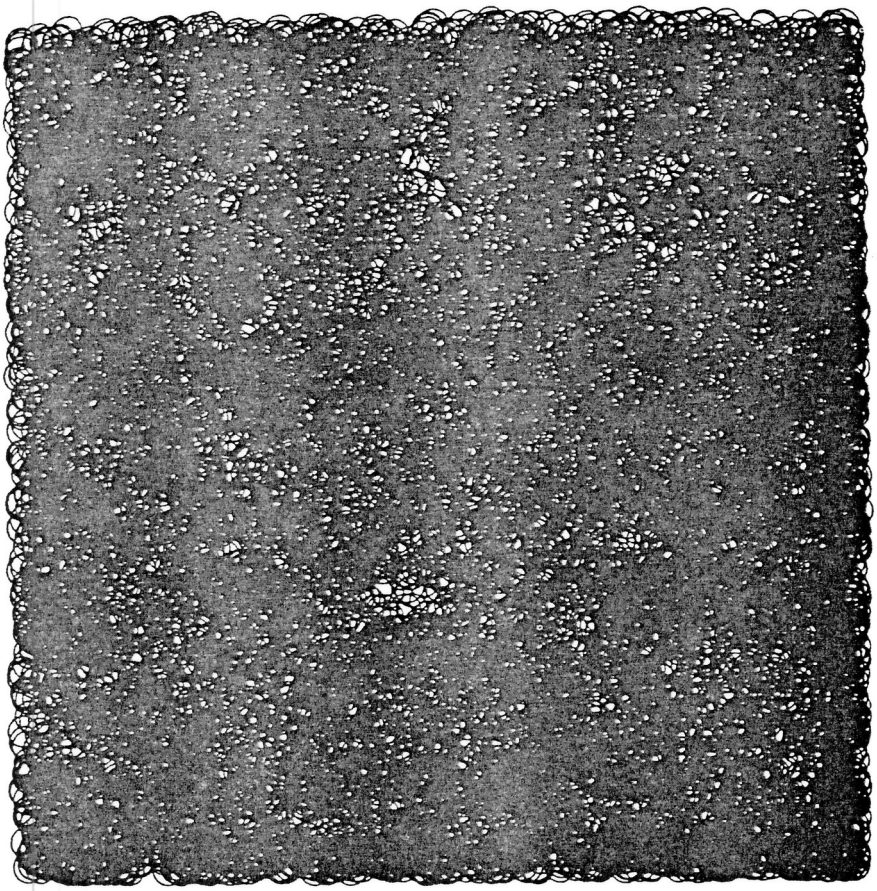

FIG. 1. Ten thousand random spheres, overlaid 10 deep.

to a new algorithm and implementation which is now being distributed by the Harvard University Laboratory for Computer Graphics and Spatial Analysis as the program PRISM [1, 7]. These algorithms draw on new discoveries in concrete algorithms analysis and database retrieval. (For example, if the circles are considered records in a database, then finding the circle intersections is analogous to finding all records with duplicate information.)

A hidden line version of SPHERES has been implemented at Rensselaer Polytechnic Institute. It handles 10,000 spheres in 383 sec on a Prime 500. The purpose of the program is to prove that the algorithm actually works on big examples, and to provide some timing statistics. Hence, no effort was made to include any knowledge of chemistry. The spheres were created with a pseudo-random number generator. There are already many fine systems containing knowledge of the atom positions in actual molecules, and it would be redundant and foolish to try to duplicate them. The regularly positioned atoms in real molecules would have fewer intersections, and hence be faster to calculate than the random case. The examples that SPHERES used look like a dense gas. The average depth of atoms behind a random point on the screen ranged up to 30. Nonetheless, SPHERES will also

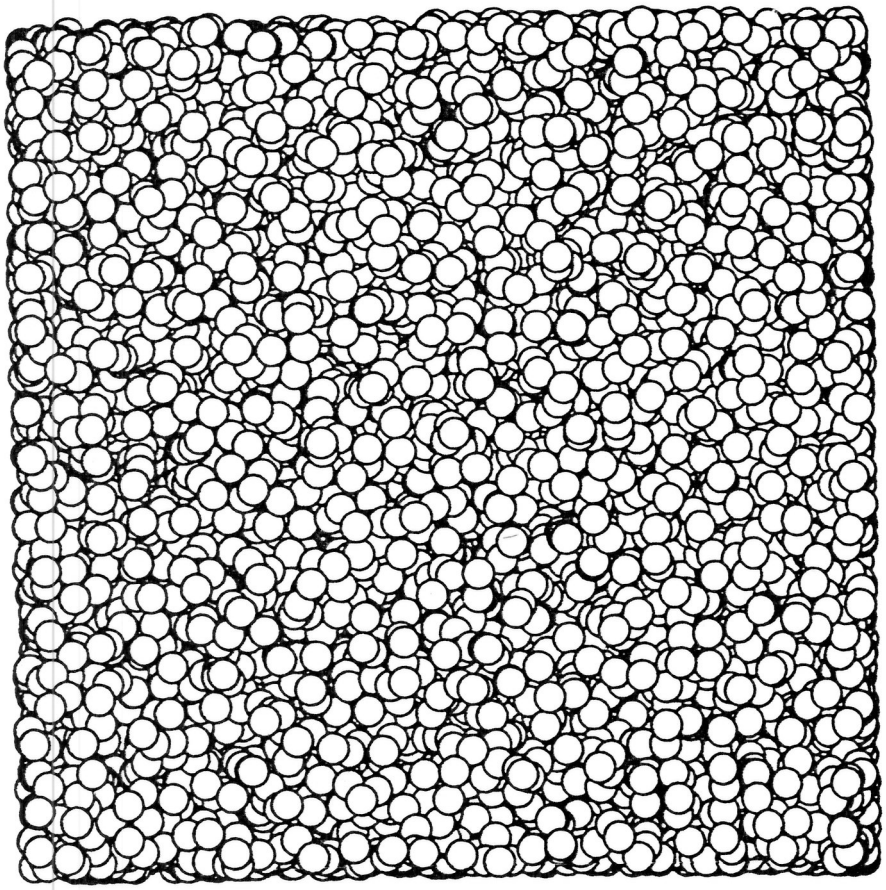

FIG. 2. The same 10,000 spheres with hidden lines removed.

handle regular arrangements as is shown in Fig. 5, which has 1500 spheres arranged in 10 layers to spell the initials of RPI's Computer and Systems Engineering curriculum. No real molecule other than a crystal is this regular.

Although SPHERES calculates only hidden lines, i.e., what parts of the perimeters of the circles are visible, the algorithm tells how to calculate the hidden surfaces also. Doing this would less than double the execution time. The hidden surface part of the algorithm has been implemented in another program using overlapping squares at Rensselaer Polytechnic Institute. It works as predicted.

Many existing hidden surface algorithms are what are called image space algorithms. They produce output on a raster CRT by calculating the color of each pixel. To avoid the jaggies and have the boundaries between different objects smooth, the intensities must be calculated at a subpixel level, which can mean calculating 4,000,000 intensities. It also costs 4 times the CPU time to double the resolution. In contrast, SPHERES is what is called an object space algorithm, which means that it calculates the output exactly to within the floating point arithmetic accuracy of the computer. Since the output is a list of visible arcs, each tagged with its origin, it can be written to a file, and displayed on a variety of pen

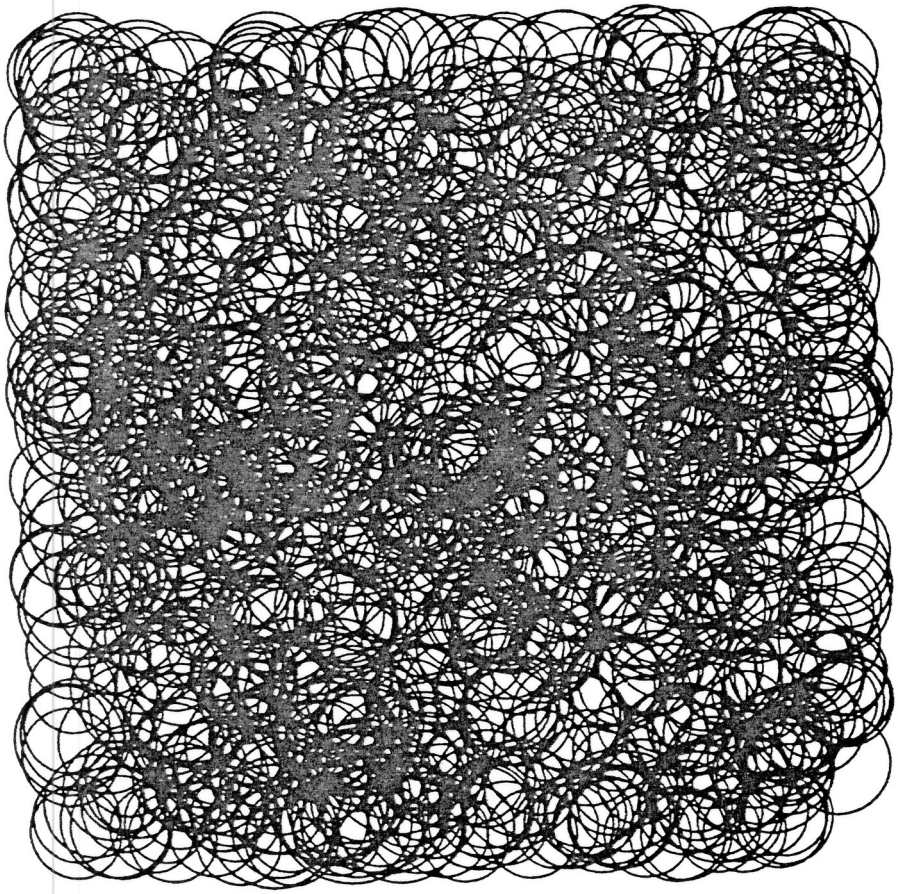

FIG. 3. One thousand random spheres overlaid 10 deep, in perspective.

plotters or raster CRT's without any more hidden surface calculations. So the output of SPHERES can be directly put onto a pen plotter, or can be easily converted to a raster CRT [8]. However, while the output from an image space algorithm can be directly displayed on a raster CRT, it cannot be easily converted to a form suitable for a pen plotter. Although the color microfilm plotters produce spectacular output, pen plotters remain much more common since they are a factor of 100 cheaper. Finally, the list of arcs and regions from an object space algorithm has meaning: you know the sphere that each line and region came from, and you know what came from each sphere. Thus this output is suitable as input for a future processing step, such as calculating shadows cast by one atom on another. In contrast, extracting the meanings from the pixel intensities produced by an image space algorithm is a difficult pattern recognition problem.

2. ASSUMPTIONS

1. Assume that our scene fills a one by one square on the perspective plane, and this is our plotter screen.

2. Let N = the number of spheres or circles.

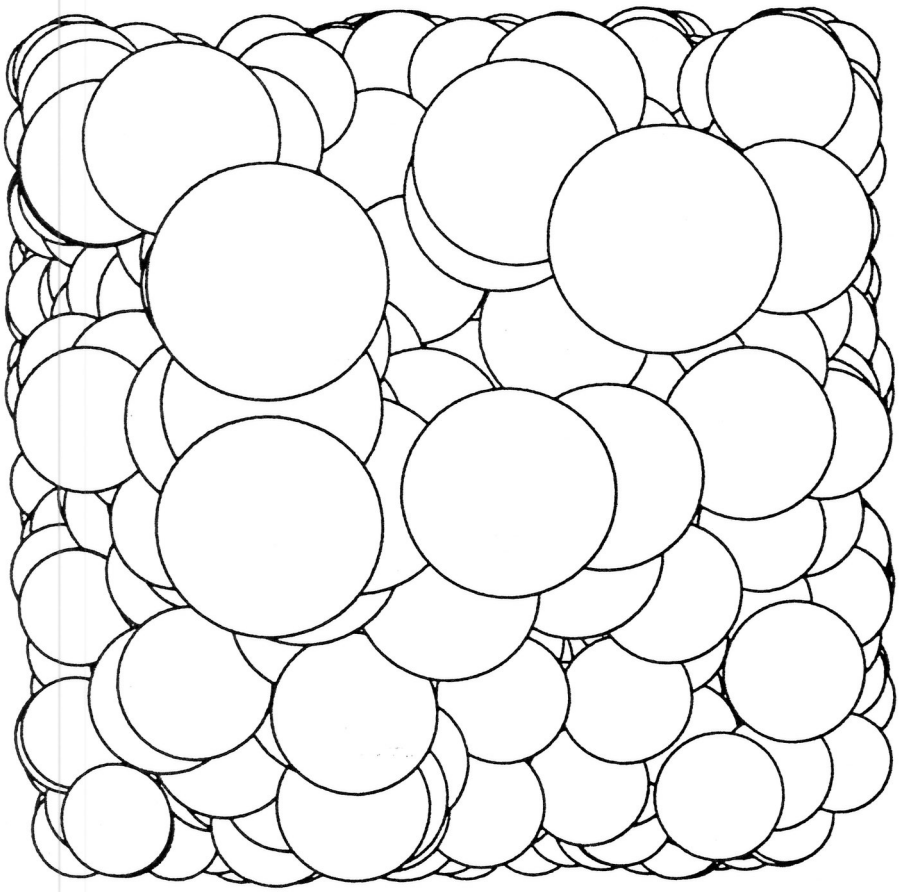

FIG. 4. The same 1000 spheres with hidden lines removed.

3. Let R = the radius of a sphere or circle.

4. Let D = the depth complexity, the average number of circles covering a point. $D = \pi * N * R.^2$

Later we will overlay a square grid on the plotter screen:

5. Let L = the length of the side of one cell of the grid.

6. Let $G = 1/L$ = the number of grid cells on one side of the square.

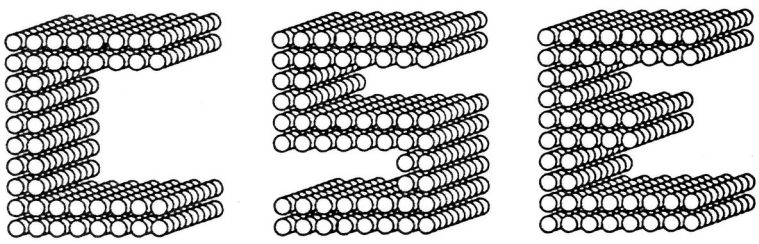

FIG. 5. Fifteen hundred spheres spelling "CSE" with hidden lines removed.

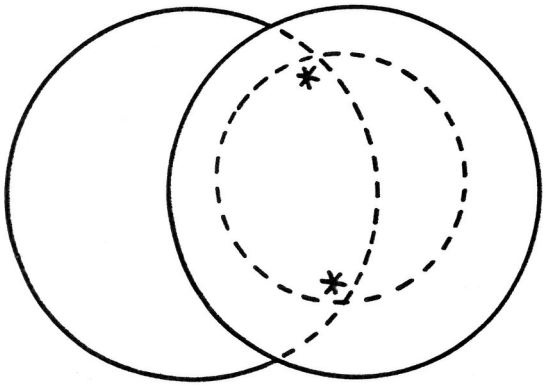

FIG. 6. Some intersections that need not be calculated.

3. MOTIVATION

This section explains some of the problems that the algorithm must solve. This is necessary to understand the underlying principles, but not to use the algorithm.

The central problem is that of determining which of the circles that the spheres project into intersect with which others. The N circles combine in $C(N, 2) = N(N-1)/2$ pairs, and each of these pairs possibly intersects. If $N = 10{,}000$, then there are 49,995,000 possible intersections. However, if $R = 0.0178$ (which gives $D = 10$), then the expected number of intersections (assuming the circles are independently and uniformly distributed in the square) is only about 200,000. Any algorithm that tests all 50 million possibilities will be too slow, regardless of how fast each test is. Even if we only test pairs of circles that are intersected by the same scan line, we will still have about 3,500,000 possibilities to check.

Moreover, there is another factor: we want the intersections only because it is only at its intersections where a circle changes its status from visible to hidden or back again. If at least one of the circles involved in this intersection is already hidden by other circles then this intersection is irrelevant, and its existence need never be discovered (see Fig. 6). In the above example, where the circles are piled an average of 10 deep throughout the square, only a few thousand of the intersections are actually relevant. The figures in this section are for an actual scene of 10,000 random spheres. There were only 32,000 relevant intersections, and SPHERES had to test only 71,000 pairs of circles to find them.

The problem that SPHERES solves is how to determine and calculate these relevant intersections without calculating the vast majority of the useless ones.

4. DATA STRUCTURES

Before we describe the algorithm itself, we will give the significant data structures used, since they are a crucial part of it. We omit any details that have no intrinsic interest, but are merely a concession to the low-level, obsolete language, Fortran. For example, this includes splitting an array $A(N, 3)$ into three arrays $A1(N)$, $A2(N)$, and $A3(N)$ so that each array is smaller than 64K bytes. As computer science advances, mechanical transformations such as this are gradually being done mechanically. On the other hand, factors that motivated the design of the algorithm are mentioned, even though they are not evident in the result.

SPHERES has the following arrays and other data structures:

1. An array of N spheres. Each sphere has this information:

(a) (CX, CY, CZ), the coordinates of it its center. CX and CY range from R to $1 - R$, and CZ is positive. The larger the CZ, the farther away the sphere is. Since the spheres are being projected orthogonally, the center of the projected circle is (CX, CY).

(b) R, the radius of the projected circle. R may range from 0 to 0.5, although if it is too large the plot is rather unrealistic.

(c) *ARCS*, a list of the invisible arcs on the circle's perimeter. Initially, a circle is considered to be completely visible, so this is empty. As closer circles cut pieces out of this circle, the pieces are added to *ARCS*. As hidden arcs are added, they may coalesce with existing hidden arcs. Eventually the circle's perimeter may be totally hidden so *VCIR*, below, is used. *ARCS* stores each arc by the angles of its endpoints, from 0 to 360°. The arcs are sorted. If an arc wraps around through 360°, it is stored as two arcs.

Since the number of arcs per circle is different for different circles, cannot be predicted accurately except on the average, and changes as the algorithm progresses, these arc lists are stored as linked lists allocated from a common free space. Linked lists, a standard technique useful throughout computer science, are described in [2, 3, 17].

(d) *VCIR*, a flag that is TRUE if the perimeter of this circle is known to be completely invisible. Note that some of the area inside this circle may still be

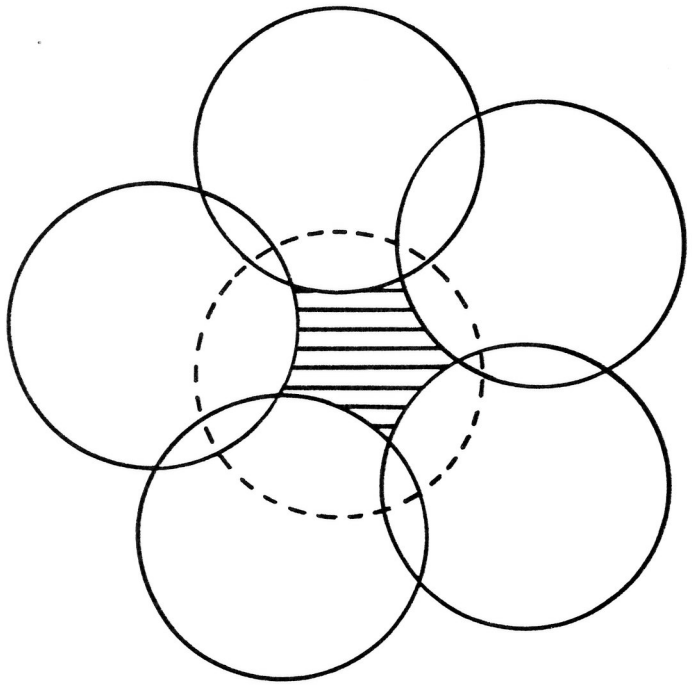

FIG. 7. A circle with some visible area whose perimeter is completely hidden.

visible, as shown in Fig. 7. This would affect hidden surface calculations, but not hidden line calculations.

2. A square grid of $G \times G$ cells, overlaid on the plotter screen. This grid is a scaffolding to help the hidden surface calculations. At the end of the algorithm, the results will be the same regardless of whether the grid is 1×1 or 1000×1000. The differences will be in the amount of storage and time used. Contrary to some other hidden surface algorithms, such as Warnock's [20], the circles are not cut where they cross a grid line. Neither does SPHERES need to proceed in a recursive descent down a tree of grid cells. Each cell, B, has the following information:

(a) *GCRLST*, a list of circles whose perimeters may pass through B. This list may contain a few circles that do not, in fact, pass through B, as long as it does not leave any out. This is because the list is used to test for possible intersections. Two circles that intersect must pass through the same grid cell at some point, so they are on a common list. Extraneous circles on the list will just cause extra possible intersections to be tested.

(b) *GRDZ*, the Z value of the closest sphere whose projected circle is known to completely cover B, if any. Initially, this is a big number (10^{30}).

3. *CTABLE*, a hash table containing pairs of circles that are known to intersect. SPHERES sometimes tests the same pair of circles more than once for intersection. *CFLAG* speeds up duplicate tests, at the expense of more memory. Hash tables are described in [2, 3, 17].

4. Extensive statistics were gathered to optimize the algorithm. They are an essential part of the data structure, not only during development, but during production use, since well-used programs do not remain static, but are modified as they are used. (Throughout a large program's life cycle, these maintenance costs can be double the program's development costs [13, p. 12]. The statistics are needed to maintain the program intelligently. For an example of the statistical output from such a run, see Fig. 8. It lists every parameter of the algorithm that is not easily predicted for the run with 10,000 spheres overlaid 10 deep.

5. ALGORITHM

Finally we get to the algorithm itself. It has the following steps:

1. Project and scale the spheres so that their projected circles fill a 1×1 plotter screen. The techniques can be found in [14] and other texts.

2. Given N and R, calculate L, the length of the size of one grid cell. L has a broad optimum; varying it by a factor of 1.5 from its best value does not slow SPHERES much, though performance drops off quickly after this. The optimal L depends on the relative speed of various parts of SPHERES so it cannot be predicted a priori. Various values were tried and the following proved satisfactory:

$$\text{IF } D \leq 2.4, \quad \text{THEN } L = R * (5.3 - 0.375 * \ln(D))$$
$$\text{ELSE } L = R * (0.6 - 0.078 * \ln(D)).$$

For a discussion of this formula, see the section on implementation.

```
[HIDSPH] SUN, FEB 17 1980, 00:16:00.
                                # Circles = 10000
                              Max Radius = 0.018
                          Average Radius = 0.018
                            Grid resolution =142
                                     ITEST = 2
          Perspectively shrink farther circles? F
                                Plotter type = 3

Time:   7.23 136.29 235.61  56.98; Total = 436.11
# Circle entries on cell lists =                    39560
# Later found to be blocked =                          0
Depth complexity =                                    9.95
Average # circles / cell (excl. blocked circles) =    1.93
Maximum # circles / cell (excl. blocked circles) =      22
# Cells with blocking circles =                      19337
# Times a blocking circle was found =                19337
# Times a circle blocked in a cell =                298911
# Pairs of circles tested for intersection =         71136
# Concentric pairs =                                    0
# Pairs that intersected =                          60084
# Unique pairs that intersected =                   32115
# Pairs that didn't intersect =                     11052
# Circles =                                         10000
# Completely visible =                                250
# Completely hidden =                                4316
# Blocked in all cells =                             3550
# Extra all hidden circles found by DRWCIR =         2319
# Partially visible with K arcs, K=1,10
  2247   754   105     9     0     0     0     0     0     0
Total # visible arcs =                                4356
# arcs found hidden by DRWCIR =                       2917
Total length of all visible arcs =                  1037.48 circles.
# Calls to PCFXN =           = 20000
# Circle pairs in hash table = 32115
# Hash table wrap-arounds =        0
Longest overflow =                14
Average overflow =             0.231
Storage usage:
              GCRLST used 39560 of 80000 @2
              CTABLE used 32115 of100000 @2
             Spheres used 10000 of 10000 @10
               Grids used 20164 of 22801 @3
```

FIG. 8. Statistical output from a run of SPHERES with $N = 10{,}000$.

3. Iterate through the circles. For each circle, C, do the following:

(a) Determine the smallest rectangle of grid cells that covers C.

(b) Test each cell, B, in the rectangle to see whether it is wholly inside C, it is wholly outside C, or C's perimeter passes through it. This can be done by testing whether B's four corners are inside C. If they all are inside, then B is inside C. C is called a BLOCKING CIRCLE (see Fig. 9) of B in this case, since it blocks everything in B that is farther away from view. (Blocking circles are the key to SPHERES' speed since none of the intersections of the circles in B farther away need to be calculated. As the scenes get more complicated, more and more of the scene is hidden by blocking circles, so that the visible complexity stays linear). If B's corners are all outside and the center of B is at least

$$R + L/\mathrm{SQRT}(2)$$

from the center of C, then B is certainly outside C. $L/\mathrm{SQRT}(2)$ is half the diagonal

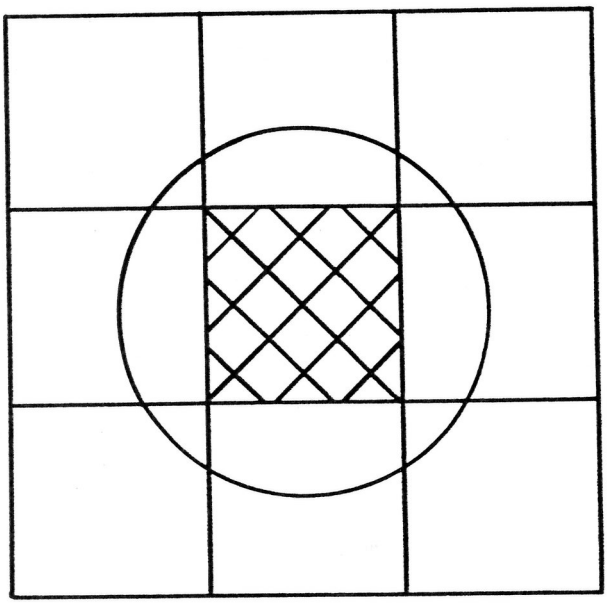

FIG. 9. A circle blocking a grid cell.

of B. If neither of the above is true, then B probably intersects C's perimeter. This is not always true, but a few false positives do nothing but slow SPHERES slightly.

(c) If C's perimeter passes through B (or probably does), then check whether C's Z-value, CZ, is greater than $GRDZ$ for B. If so, then the part of C in B is certainly hidden, and so we go to the next B. Otherwise:

(i) If C is a blocking circle for B, then set $GRDZ$ to CZ.

(ii) Otherwise add C to B's $GCRLST$.

4. Next, iterate through the grid cells. For each cell, B, do the following:

(a) Iterate through the circles in B's $GCRLST$. (Remember that there is a separate $GCRLST$ for each cell). For each circle, C:

(i) If $CZ > GRDZ$, then delete C from $GCRLST$. (Since $GRDZ$ was being lowered as more circles were added to $GCRLST$, a circle that was added early, but had a larger CZ than a later blocking circle, would be deleted now. A check could have been made for newly blocked circles everytime that $GRDZ$ was lowered, but that would have been slower, since the actual list of circles is not needed until now. The reason for the initial check when the circles were added is that this takes a small constant time and saves some space.)

(ii) If $VCIR$ is TRUE, then also delete C from $GCRLST$.

Note that C probably resides in several $GCRLST$'s, and deleting it from one does not delete it from the others also.

(b) Take all possible combinations of the remaining circles in $GCRLST$. For each pair, $C1$ and $C2$:

(i) If *CTABLE* shows that $C1$ and $C2$ have been found to intersect, then go to the next pair.

(ii) Otherwise, test whether or not $C1$ and $C2$ intersect. They intersect if their centers are closer than $2 * R$.

(iii) If they intersect, assume without loss of generality that $C1$ is closer than $C2$. Add the pair ($C1$, $C2$) to *CTABLE*. Determine the arc of $C2$ that $C1$ hides. Add it to $C2$'s *ARCS*, its list of invisible arcs. This new arc may merge with existing arcs, reducing their number. If $C2$'s perimeter is now totally invisible, then clear its *ARCS* and set its *VCIR* to TRUE.

The reason for *CTABLE* is that two circles that are nearly coincident will have several cells in common. Calculating their hidden arcs repeatedly for each such cell would waste time and possibly cause errors. *CTABLE* prevents this.

5. Iterate through the circles a second time. For each circle, C:

(a) If *VCIR* is TRUE, then go to the next circle.

(b) Determine C's possibly visible arcs by taking the complement of *ARCLST*. Some of these arcs may be hidden, but at this point, each arc is a unit that is either completely hidden, or else completely visible. For each arc, A, do:

(i) Take A's midpoint, P say. (P is hidden or visible depending on whether or not A is.)

(ii) Determine which cell, B, that P falls in. (If P is hidden, then it is by a circle in B.)

(iii) Test P against all the circles in B's *GCRLST* to see if any hide it.

(iv) If none do, then A is visible, so plot it.

6. Finally, retrieve the plot.

This algorithm represents a tradeoff between complexity and speed. Although there are several loops, SPHERES is quite fast. For proper values of G, the number of executions of the inner loops is bounded, and the number of executions of the outer loops is linear in N, for random scenes. A more formal mathematical analysis may be found in [9]. Someone who knows how SPHERES works can construct unrealistic examples that would make it run very slowly. However, this is true for many programs. SPHERES runs fast on most reasonable scenes, even though they may be very complicated with thousands of overlapping circles. In particular, it handles the case where the depth complexity (the number of spheres deep) is very large. In fact the random examples that it was tested on are probably much more difficult than actual examples would be.

6. IMPLEMENTATION

SPHERES has been implemented as a Flecs program at RPI. Flecs [4] is a Fortran preprocessor that adds block structure. It is in the public domain, and is available on several different computers such as IBM 3033, PDP 10, and Prime. It may be obtained from the author, Terry Beyer. The graphic output is onto Imlac vector refresh terminals, although the figures in this paper were done on a Tektronix. SPHERES is 1400 lines long, and is well documented.

SPHERES runs on a Prime 500. This is a minicomputer with 500K 16-bit words of memory. Some typical instruction times are 0.56 μsec for an add to memory and 4.02 μsec for a single precision floating multiply. The Fortran compiler does not know what optimization is. All this makes SPHERES 10 to 20 times slower than it would be on an IBM 3033.

Since SPHERES is designed to exhibit the algorithm's speed on thousands of spheres, it generates random spheres as input. However, the linear congruential generator that all manufacturers use is unsuitable. If successive pseudo-random values from a linear congruential generator are paired and plotted, then they fall on a small number of straight lines. This is true of all of them regardless of the initial seed and the multiplier used. This may not usually matter, but is totally unsuitable for a geometric program. Instead, a cubic modular generator is used. For example, for the Ith circle,

$$CX = \frac{\text{MOD}(5281 + 13513 * I + 22343 * I^2 + 16823 * I^3, 32003)}{32003}.$$

These numbers are all prime. This generator is similar to, though much simpler than, the one-way functions that can be used in public key cryptosystems. CY and CZ are calculated similarly. Nevertheless, SPHERES also worked well with the built-in generator.

Although a formal mathematical analysis [9] shows that SPHERES takes execution time linear in N, the exact time and the optimal values of L cannot be easily predicted a priori. For a constant N and R, as L varies, some steps of SPHERES run faster and some slower, and the exact time depends in a complicated way on the number of visible arcs, the number of circles in a cell before the blocking circle, the variances of these probability distributions, and so on. So extensive experiments were made to determine heuristically the optimal L.

Table 1 shows the optimal grid size for depth complexities varying from 0.09 to 31.4, a range of 300 to 1. There are several things to note about this table:

1. These values were determined experimentally by varying L over a wide range for each value of N and R to obtain the minimum T.

2. The times (which are seconds of CPU time on a Prime 500) vary a few percent as SPHERES is run with different loads on the computer.

3. The time for a given case is a little larger if it is run on a version of SPHERES that has been compiled with larger arrays so that it can handle larger cases.

TABLE 1
How Time and Optimal Grid Depend on Depth

N	R	best L	L/R	D	T	Tn
300	0.01	0.0625	6.25	0.09	2.6	2.6
300	0.03	0.167	5.6	0.85	5.5	5.5
300	0.05	0.25	5.0	2.36	11.6	11.6
300	0.07	0.033	0.47	4.6	13.4	13.4
300	0.1	0.04	0.4	9.0	9.0	9.0
500	0.1	0.04	0.4	15.7	15.0	9.0
1000	0.1	0.033	0.33	31.4	27.8	8.3

4. This implementation has not been extensively optimized so that in a production version the times would be smaller.

5. The times do not include the time to actually plot the arcs after they have been calculated since this is not a property of the algorithm and is dependent on the plotting package used.

6. For $N = 300$, $R = 0.5$, there was a very broad optimal value of L. For $0.03 \leq L \leq 0.3$, T varied less than 20%.

7. It is impossible to have depth complexities greater than 10 for $N = 300$ since this would make R so large that edge effects would affect T. So a larger N was used, and the time was extrapolated to a normalized time, Tn, that would be expected to hold for $N = 300$ if there were no edge effects. This was done by assuming that in the plot, the time of any subsquare with a fixed N depended only on D. This makes the time for a fixed D linear in N. This gives

$$Tn = T * 300/N.$$

This formula was only used for the last two lines of the table; even ignoring them, the depth complexity has been tested over a 100 to 1 range.

8. The worst D falls about 4.6. That T should have a maximum as D varies (for fixed N) can be explained: If D is small the plot is sparse and there are few interactions, and if D is very large, most of the circles are completely hidden. The messiest case is somewhere in the middle, where most of the circles are partly visible.

9. However, T is reasonably independent of D, unless D is very small. In particular, T does not increase as D gets extremely large. This is where SPHERES outperforms other hidden surface algorithms that have times proportional to D.

10. The small values of T when D is small show that the complicated parts of the algorithm adapt easily, without causing any overhead, to the case where the input is so simple that they are not needed.

11. The step function in the optimal L/R was a complete surprise to the author. It happens at the point where a grid cell has a reasonable chance of having a blocking circle. If there are so few circles that the average cell is unlikely to have a blocking circle regardless of its size, then the cells should be larger, since this minimizes the cost due to the fixed overhead of each cell. However, once D is large enough (around 2) that blocking circles become reasonable, then the cells should be made smaller so that they are more likely to be blocked. This is because it is the blocking circles that are the final step in making SPHERES linear in time.

As a result of these tests, the following formula was used thereafter:

$$\begin{aligned} \text{IF } D \leq 2.4, \quad & \text{THEN } L = R * (5.3 - 0.375 * \ln(D)) \\ & \text{ELSE } L = R * (0.6 - 0.078 * \ln(D)). \end{aligned}$$

Other formulae might be slightly better.

Given the optimal L, tests were next run on varying values of N over a range of 100 to 1. D was kept constant at 10 since this creates scenes of suitable difficulty that are not ridiculously complicated. The results are shown in Table 2. This shows

TABLE 2
Variation of T on N with D Fixed

N	R	G	L	T
30	0.326	7	.143	4.1
100	0.178	13	.077	5.6
300	0.103	23	.043	10.3
1,000	.0564	42	.024	31.1
3,000	.0326	72	.014	97.5
10,000	.0178	142	.007	383.

that SPHERES actually does take linear time in N, except for a slight increase for the largest scenes. The cause of this nonlinearity is not known, but it is possibly due to page swapping. For the largest case, SPHERES runs at a virtual to real memory ratio of about two.

SPHERES processes the circles in the order that they happen to be in. It does not sort them by CZ. Since most sorting algorithms take time that grows at least with $N * \log(N)$, using one of these would make SPHERES no longer linear. Now the proof that SPHERES is linear assumes that the input spheres are distributed independently and identically. Under these assumptions, an address calculation sort that takes only linear time [11] could have been used, but this is unnecessary. SPHERES was tested on various pairs of cases where N, R, and L were the same, but CZ was sorted in one case and random in another. The times were identical within a few percent.

Often we wish to use a perspective projection so that the variable sizes of the circles aid in depth perception. There was a suggestion that, since the theoretical proof of the linear time was for equal sized circles, SPHERES might run much slower on this case. This was tested by letting CR shrink as CZ increased, according to the formula

$$Ri = R1 * \log(11)/\log(i + 10),$$

where Ri is the radius of the ith sphere from the front. $R1$ is the radius of the closest sphere. For example, if $N = 1000$ and $R1 = 0.135$, this has the radii decrease in such a way that $D = 10.25$. This is the case used in Figs. 3 and 4. (To get a depth of 10 for fixed R requires $R = 0.0564$.) In this variable case, the optimal grid size was determined heuristically to be $L = 0.025$. This gave $T = 19.3$ sec, which is 38% faster than the same N and D with R fixed. The fixed and variable cases are summarized in Table 3. The reason is that although the scene before the hidden lines are removed, as shown in Fig. 3, may be very messy, the scene after, as shown in Fig. 4, is quite simple because the few large spheres in front easily hide

TABLE 3
Comparison of Times for Fixed and Variable R

	N	max R	D	opt L	T
Fixed R	1000	0.0564	10.0	0.024	31.1
Variable R	1000	0.135	10.25	0.025	19.3

the many small spheres in behind, and the strong point of SPHERES is its ability to efficiently delete irrelevant complexity.

This is the storage that SPHERES needs, in 16-bit words. The numbers in parentheses are the amounts that were allocated for the $N = 10{,}000$ case.

$10 * N$	(100K)
$+ 2 *$ size of *CTABLE*	(200K)
$+ 5 *$ total number of *ARCS*	(75K)
$+ 3 * G^2$	(30K)
$+ 2 *$ total size of *GCRLST*	(100K)
Total =	505K

Several of these arrays are padded since the exact amount needed, for example in *ARCS*, cannot be predicted in advance. Only 375K words were actually used. This figure could be further reduced to 293K words by using *INTEGER* * 2 instead of *REAL* * 4 for arrays such as *CX*.

7. APPLICATIONS

Besides molecular models, SPHERES can be used for any other application where the objects can be modelled with noninterfering spheres. For example, it is being used by W. R. Spillers in the Department of Civil Engineering at R. P. I. to model twisting electrical cables in a conduit. Each cable is represented by a string of spheres. Simple applications such as this, with only a few hundred spheres, can be calculated in a few seconds.

8. CONCLUSION

This paper has shown how algorithm analysis and data structure techniques can be useful in computer graphics. In particular, it presents an exact hidden line algorithm operating on spheres whose execution time, provided that the input is distributed statistically independently and identically, is linear in the number of input spheres. This is not a restriction in real scenes, since as was shown by experiment, even if the input is as regular as a crystal, the projected circles still have enough randomness that the hidden lines can be calculated rapidly. The time of this algorithm, SPHERES, in contrast to all other hidden surface algorithms, is independent of the depth complexity of the scene. Further, the time of SPHERES, in contrast to that of all image space algorithms, is independent of the resolution of the display, since SPHERES operates in object space. Nothing in the underlying concepts is peculiar to spheres or to hidden lines only, and the techniques are of general value and can be extended.

ACKNOWLEDGMENTS

The assistance of Leong Shin Loong, Abel Shi Lo and Neeraj Sangal, who implemented some of the results while M. Eng. students at R.P.I., is gratefully acknowledged.

REFERENCES

1. Map programs, data bases listed, *Computerworld* Sept. 8, 1978, 29.
2. A. V. Aho, E. Hopcraft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, Mass., 1974.
3. S. Baase, *Computer Algorithms: Introduction to Design and Analysis*, Addison–Wesley, Reading, Mass., 1978.
4. T. Beyer, *Flecs: User's Manual*, Computing Center, University of Oregon, Eugene, Oct. 1975.
5. A. S. French, Computer simulation of space-filling molecular models, *IEEE Trans. Computers* **C-26**, No. 10, 1977, 1026–1028.
6. W. R. Franklin, *Combinatorics of Hidden Surface Algorithms*, Harvard University Center for Research in Computing Technology TR-12-78, May 1978.
7. W. R. Franklin, 3-D graphic display of spatial data by prism maps, *Comput. Graphics (ACM–SIGGRAPH)* **12**, No. 3, 1978, 70–75.
8. W. R. Franklin, Evaluation of algorithms to display vector plots on raster devices, *Computer Graphics and Image Processing* **11**, 1979, 377–379.
9. W. R. Franklin, A linear time exact hidden surface algorithm, *Comput. Graphics (ACM—SIGGRAPH)* **14**, No. 3, July 1980, 117–123.
10. K. Knowlton and L. Cherry, ATOMS—A three-D opaque molecular system, *Comput. Chem.* **1**, 1977, 161–166.
11. D. E. Knuth, *The Art of Computer Programming, Vol 3: Sorting and Searching*, Addison–Wesley, Reading, Mass., 1973.
12. N. L. Max, *AtomLLL—A Three-D Opaque Molecular System (Lawrence Livermore Labs version)*, Jan. 11, 1979. UCRL-52645, Lawrence Livermore Labs, University of California, Livermore.
13. G. J. Myers, *Software Reliability: Principles and Practices*, Wiley–Interscience, New York, 1976.
14. W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics*, 2nd ed., McGraw–Hill, New York, 1979.
15. T. K. Porter, Spherical shading, *Comput. Graphics (ACM—SIGGRAPH)* **12**, No. 3, 1978, 282–285.
16. T. K. Porter, The shaded surface display of large molecules, *Comput. Graphics (ACM—SIGGRAPH)* **13**, No. 2, 1979, 234–236.
17. E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice–Hall, Englewood Cliffs, N.J., 1977.
18. J. T. Scott, Computer films for research, *Phys. Today* **32**, No. 1, 1979, 46–52.
19. G. M. Smith and P. Gund, *Computer Generated Space-Filling Molecular Models*, Mar. 5, 1978, Merck Sharp & Dohme Research Labs, Rahway, N.J.
20. I. E. Sutherland, R. F. Sproull, and R. Schumacker, A characterization of ten hidden surface algorithms, *Comput. Surveys* **6**, No. 1, 1974, 1–55.