REVISITING WATKINS ALGORITHM

J.C. Beatty, K.S. Booth, and L.H. Matthies

Department of Computer Science
University of Waterloo, Ontario

ABSTRACT

In the eleven years since Watkins' algorithm first appeared it has not been subjected to the rigorous complexity analysis applied to other problems such as is typified by the vast literature on searching and sorting. This neglect has resulted in the use of terms such as "visual complexity" to describe the running times of the algorithms, with no precise formulation of what that complexity is.

This paper discusses some optimizations to Watkins' algorithm and provides a complexity analysis of our particular version of the algorithm. We also emphasize the wide applicability of the basic scanline approach as evidenced by related algorithms for cross-hatching, haloing and VLSI layout.

RÉSUMÉ

Au cours de ses onze années d'existence, l'algorithme de Watkins n'a pas été soumis à l'analyse de complexité rigoureuse comme l'ont été d'autres problèmes si on en juge par la documentation importante sur les opérations de recherche et de tri. Cette négligence a abouti à l'utilisation de termes comme "complexité visuelle" pour décrire les temps d'utilisation des algorithmes sans que cette complexité soit définie de façon précise.

Dans le présent document, nous traitons de quelques méthodes d'optimisation de l'algorithme de Watkins et nous présentons une analyse de complexité de notre propre version de l'algorithme. Nous soulignons également les vastes possibilités d'application de l'approche fondamentale d'analyse par ligne comme le montrent des algorithmes rattachés pour la présentation par hachures, par halos et par très grande intégration (VLSI).

Extended Abstract

1. Introduction

The visible surface problem is to efficiently determine which portions of a scene are visible from some viewpoint. The problem is difficult: it has a long history, a variety of approaches have been explored, and it is still an active area of research. We are interested in applying techniques from computational geometry, theory of data structures and analysis of algorithms to the family of scan line algorithms, especially to the algorithm developed by Gary Watkins which has been in use for over ten years. Our work stems from the following three observations.

* Watkins' algorithm has not been subjected to a rigorous complexity analysis, in contrast to problems such as sorting. Terms like "visible complexity" have been used to describe the running time but no precise definition has appeared in the literature.

* Watkins' algorithm can be sped up by using additional coherence information and by employing more sophisticated data structures which allow many scan lines to be processed with much less work, especially for the hidden line version of the algorithm.

* Watkins' algorithm is very similar to a number of algorithms used in graphics and computational geometry. The scan line approach in these algorithms is a technique which deserves to be in the standard programmer's toolkit.

In section 2 we describe the general structure of Watkins' algorithm and argue that scan line algorithms have significant advantages in terms of efficiency of execution and quality of image when compared with some other algorithms. Section 3 addresses our first observation, suggesting a uniform set of parameters for measuring the complexity of visible surface algorithms. This is then applied to Watkins' algorithm in Section 4 where we describe a number of improvements which have been made since Watkins' original description of the algorithm. Section 5 continues this discussion with a case study of a scan line algorithm for computing the haloed line effect. Section 6 describes two other problems for which similar solutions have been proposed. We conclude in Section 7 with a short summary of the work we have been doing and an indication of further work to be done.

Our goal here is to provide a thorough complexity analysis of our haloing algorithm and to stress the similarity between related algorithms by using a uniform framework for discussion.

2. Watkins' Algorithm

In his Ph.D. thesis [W], Gary Watkins presented an elegant algorithm for determining the visible surfaces of complex 3-dimensional scenes. Watkins generated output for a raster display in which a picture is a rectangular array of pixels. The visible surface problem is equivalent to deciding, for each pixel, which of the polygons is visible at that pixel. A realization that the problem need only be solved to screen resolution is one of the key ideas in Watkins' algorithm. A second is the notion that each scan line (a horizontal row of pixels) can be processed separately, but that if the scan lines are processed in order from top to bottom, consecutive scan lines will usually be quite similar. This is called scan line coherence and is one instance of a more general observation: within a scene visible areas or objects tend to change infrequently and only at discrete places.

We will assume that a scene is made up of objects consisting of planar polygons in three dimensions and that no polygon penetrates another. A scan line algorithm is one which sorts polygons according to the scan lines on which they first appear in the picture, then processes each scan line in turn, adding polygons as they enter the picture and deleting them as they exit.

Figure 1 shows two examples of scenes in which (a) all edges have been drawn, (b) only edges of front-facing polygons have been drawn, and (c) only visible edges have been drawn. The

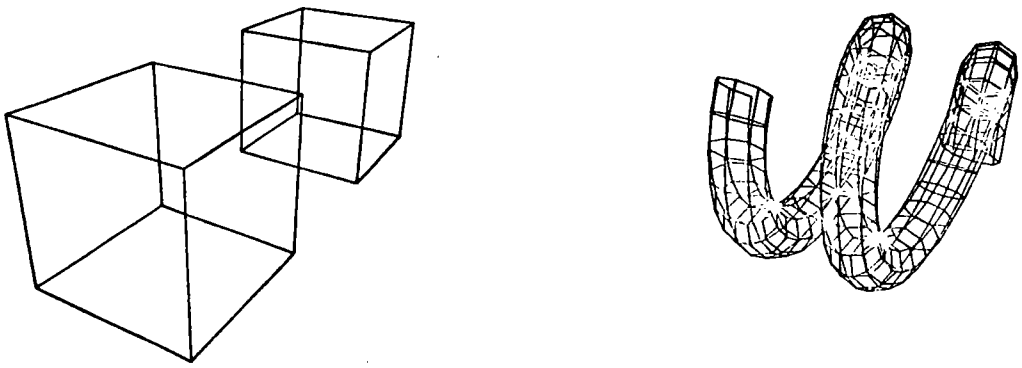

Figure 1(a). Wire-frame with all edges drawn.

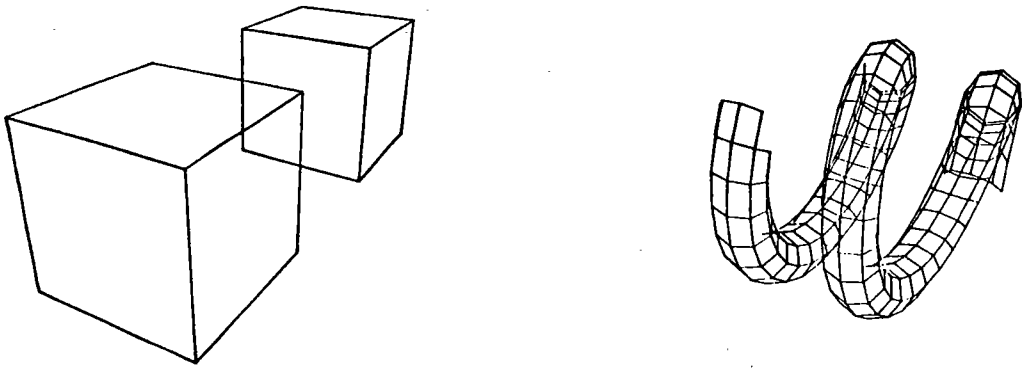

Figure 1(b). Backfacing polygons removed.

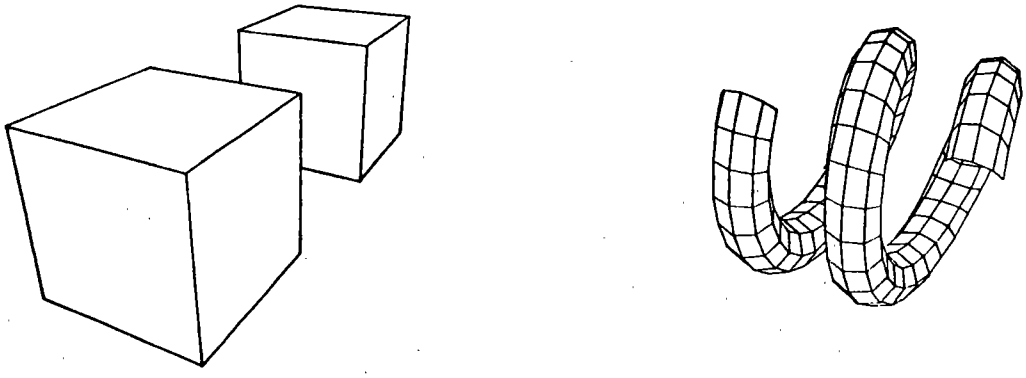

Figure 1(c). Only visible edges drawn.

purpose of using a visible surface algorithm is to add realism to the rendering. The advantage of displaying only the visible edges is readily apparent in the second set of examples.

Watkins' algorithm processes each scan line maintaining a list of active edges sorted along x. As the list is traversed the active polygons are compared in z and the frontmost polygon is displayed. Figure 2 illustrates the top-to-bottom order in which polygons enter a scene after being bucket sorted according to their highest y coordinates.

Efficiency is achieved by maintaining a sample list of spans, which are segments of a scan line along which the same polygon is visible. This simplifies the z sort within a scan line. The algorithm capitalizes on scan line coherence by using the same sample list on each scan line and by assuming that the polygons appear in the same x and z relationship on successive scan lines. When these assumptions are incorrect, the algorithm performs additional work to reestablish the appropriate lists.

Using the scan line approach means that many complicated calculations, such as determining the intersection of an edge with a particular scan line, can be replaced by incremental calculations, in which the intersection is computed from the previous intersection by addition of the inverse slope. Depth and shading information are also easily updated in this manner.

Scan line algorithms are not the only means for solving the visible surface problem. Declining costs for computer memory have greatly increased the availability of raster terminals and frame buffers. Techniques such as the z-buffer algorithm are computationally less expensive when implemented directly in hardware or firmware and have supplanted the more sophisticated visible surface algorithms for some applications [C, NS2]. Older algorithms based on priority orderings of the input polygons have been revised to handle a larger class of problems [Y]. In both cases there remains a major stumbling block: the problem of anti-aliasing pictures, especially for full color renderings, is still largely unsolved.

Aliasing is the term applied to the jagged lines which result when a line is drawn on a raster terminal. If the line is not exactly vertical or horizontal then it must be approximated by a sequence of pixels which will not fall precisely on the line. Figure 3 shows an exaggerated example of aliasing for the first scene of Figure 1.

There are well-known techniques for anti-aliasing a picture by averaging all of the visible polygons for a given pixel, but these do not work well for z-buffer or priority algorithms [Cr]. The reason is that these algorithms overwrite previously computed areas of the screen; they are unable to correctly compute the intensity of a single pixel because there is never a time at which all of the polygons visible at that pixel are simultaneously under consideration. Scan line algorithms do not suffer from this drawback and can be easily modified to produce very high quality anti-aliased renderings of complex scenes.

Scenes may be described using more complicated surface patches such as B-splines or Bezier surfaces [NS2]. Recently Lane, Carpenter, Whitted and Blinn have described a family of visible surface algorithms for dealing with these parametric definitions of 3-dimensional objects [LCWB]. The techniques we propose for improving the polygon-based algorithms should also apply to their patch-based scan line algorithms. The algorithms are necessarily more complex, so the percentage improvement may be less.

3. Analyzing Complexity

In the appendices to their survey of ten hidden surface algorithms Sutherland, Sproull and Schumacker propose a list of twenty environment statistics for measuring the complexity of a scene [SSS]. They provide estimates of the running times for each of the algorithms in terms of these statistics.

These environment statistics are one way of parametrizing a scene. We think that a slightly different viewpoint provides more insight into the

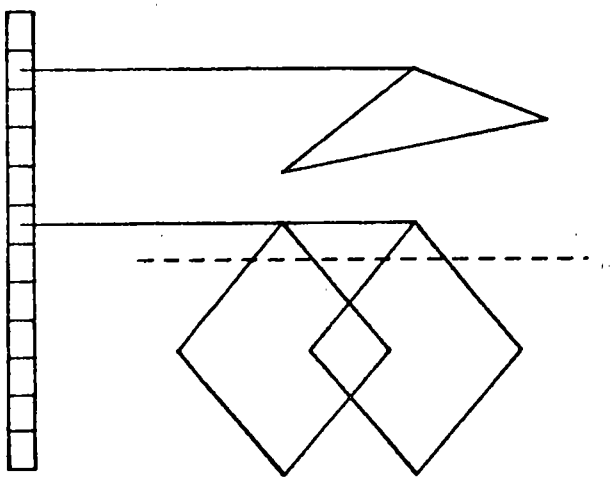

Figure 2. Edges of polygons enter the picture as the scan lines on which they first appear are processed. Active edges are kept in x-sorted order along the scan line.

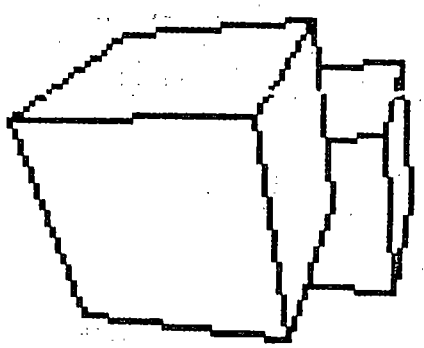

Figure 3. The jagged lines characteristic of aliasing on raster terminals have been exaggerated in this view of the two cubes.

efficiency of visible surface algorithms. Examining Watkins' algorithm reveals that the following operations are being performed.

* Work proportional to the number of edges in a scene during the x and y bucket sorts at the start of each frame and on various scan lines as each edge enters and leaves the scene.

* Work proportional to the number of polygons during initialization of the scene.

* Work proportional to the number of edges on scan lines as x and z values for each edge are incremented from scan line to scan line.

* Work proportinal to the number of total edge crossings keeping the active edge list in x-sorted order.

* Work proportional to the number of visible edge crossings determining a new visible polygon.

* Work proportional to the screen resolution initializing the bucket sorts and performing the outer loop of the top-to-bottom scan of the scene.

All of these measures are included in the environment statistics, or can be easily derived from measures in that list. We choose these for our study because they allow us to focus on some of the significant costs which can be reduced.

A useful definition of "visual complexity" can be made in terms of these parameters. Any algorithm which solves the visible surface problem must examine every polygon, in fact every edge, at least once. Moreover it seems reasonable to allow time at least linear in the screen resolution. Finally, since each visible edge crossing requires some output action the running time must allow for their detection. We thus propose that the visual complexity of a picture be defined as the sum of the number of edges in a scene, the screen resolution and the number of visible edge crossings.

Our view of Watkins' algorithm indicates that its running time is not proportional to the visual complexity, contrary to some claims in the literature [NS1]. Instead, there is a term linear in the total number of edge crossings (whether visible or not) and also to the total number of edges on scan lines, since each edge is processed on each incident scan line. The latter term might reasonably be included in the visual complexity, but not the former. A term proportional to the total number of edge crossings arises in Watkins' algorithm precisely because the active edge list is kept in x sorted order. This is accomplished with a bubble sort. A bubble sort always does work proportional to the number of inversions within the data. Each edge crossing is an inversion. The extra work spent looking at invisible crossings is a prime candidate for elimination.

It is worth noting that for n objects there can be, in the worst case, n squared edged crossings. This leads to quadratic behaviour, precisely the objection raised by Sutherland, Sproull and Schumacker when they compare Roberts' and other algorithms to Watkins'.

4. Suggested Improvements

Watkins' algorithm is designed for hidden surface processing. It is easily modified to handle hidden line elimination by keeping track of when edges become visible and invisible. On a raster display this is probably best performed by simply drawing scan segments with the "first" and "last" pixels "on" and intermediate pixels "off" [NS2]. On line-drawing systems one would remember the endpoints of the edges and issue simple drawing commands to generate the visible scene.

The basic algorithm can be sped up by observing that multiple scan-line coherence occurs in many pictures. This means that not only are adjacent scan lines similar, but in fact a whole group of scan lines will be similar. An algorithm for a raster display must specify the image at each pixel of each scan line; a line-drawing algorithm need not be concerned with scan lines on which no "critical events" occur.

We define a critical event to be the visible entry, exit or crossing of edges in the scene. We propose to ignore scan lines on which no critical events occur.

Sutherland, Sproull and Schumacker made a similar observation in their survey [SSS] and later Hamlin and Gear [HG] implemented a visible surface algorithm which did predict crossings. Their algorithm predicted all crossings, however, which still left a potentially quadratic behaviour, although they noted that there was no need to process crossings which were invisible.

As Watkins' algorithm processes a scan line it verifies that the edges are in x-sorted order along the scan line. As this check is made it is possible to predict the next scan line on which two edges will cross. This, together with readily-available information about when edges enter and leave the picture, enables the algorithm to predict the next scan line on which a critical event might occur. It is thus free to skip intervening scan lines.

A further improvement on this scheme fully realizes the potential for ignoring invisible crossings. Two edges which cross while obscured by some other surface do not affect the visible picture so the scan line on which the crossing occurs does not have to be processed (unless, of course, there is some critical event on that scan line). The final observation is that on a scan line which has a critical event, if the crossing is known to lie between two visible edges, there is no need to process the entire active edge list, only the spans (regions of the scan line) in which the event occurs need be updated.

In the first edition of their textbook on graphics, Newman and Sproull report that

> [Watkins' algorithm] is quite fast, although its dependence on the complexity of the scene is difficult to analyze. Watkins tabulated the performance of the algorithm for a variety of scenes and discovered that the computation grows roughly as the visible complexity increases [NS1, page 321].

This is not the case, at least for the standard version of Watkins' algorithm which appears in the literature. In the worst case the algorithm has a quadratic behaviour, due simply to the fact that the bubble sort of the active edge list requires that all of the inversions removed as edges cross.

A careful analysis of the running time of our proposed algorithm indictes that it is still not proportional to the visual complexity of the scene. An invisible edge which crosses diagonally through the scene can cause many scan lines to be processed unnecessarily. In the worst case it seems that quadratic behaviour is still possible. We have not completely analyzed the effect of more sophisticated data structures for maintaining z-sort information within a span. Perhaps with this information the quadratic behaviour can be avoided for scenes whose visual complexity is not quadratic in the number of objects.

The version of the algorithm proposed here does not handle penetrating polygons. Adding this to the algorithm is a straightforward adaptation of the solution used in Watkins' original algorithm. We have left it out here to provide a clear indication of the differences between our algorithm and the original. There are added complications in the z-sort which have not been fully explored. Again, more sophisticated data structures for maintaining information within a span may reduce the complexity.

5. Haloing

One interesting question which can be asked is whether the scan line algorithms can be used to provide an exact solution for the visible surface problem. It seems that the scan line algorithms derive some of their power from the fact that they only solve the problem to screen resolution. This is true. But an exact solution can be obtained if we use priority queues to keep track of the

y coordinates at which critical events occur. This allows us to process only those slices through the picture at which the scene changes. The cost is a logarithmic slowdown, replacing the y bucket sort with one of the standard priority queue algorithms.

A number of other problems can be solved with an exact scan line approach. The one which we will look at in detail is a variant of the visible surface problem. Appel, Rohlf and Stein [ARS] have pointed out that for some applications a rendering with hidden lines removed is inappropriate. They suggest a haloing effect in which hidden lines are removed only when they pass near or behind other lines. Their implementation is based on a hidden line algorithm of Appel, but they observe that the same techniques probably apply to other hidden line algorithms.

Figure 4 illustrates the use of haloing on three scenes showing (a) a wire-frame rendering in which all edges have been drawn, (b) a haloed rendering in which edges crossing behind other edges are clipped as they pass behind the front edge, and (c) a visible surface rendering in which Watkins' algorithm has been applied to eliminate all hidden line segments.

The scan line algorithm outlined in Section 4 can be adapted to haloing. The algorithm already keeps track of where edges cross so that just the visible portions are drawn. The only change required is to keep track of the distance from the crossing point at which the line should be terminated. The idea of skipping ahead to the next scan line on which a crossing occurs is still relevant, although now we have to revert to a version similar to Hamlin and Gear's algorithm since all crossings are visible, not just those crossings which involve edges of visible surfaces.

Our implementation of the haloing algorithm maintains two priority queues, each implemented as 2-3 trees. The event queue maintains the critical events in y-sorted order; the edge queue maintains the active edges in x-sorted order. The algorithm cyclically removes the minimum (highest scan line) critical event from the event queue and updates the edge queue by inserting, deleting or reordering the active edges. The edge queue makes it easy to find the edges to the left and right of a critical event, thus allowing further critical events to be predicted and inserted into the event queue.

An event is any edge entering or leaving the scene or any two edges which cross. For haloed pictures all events are visible, thus critical. As each event is processed, it is removed from the event queue (in scan line order because of the y-sort). The x-sorted edge queue permits the immediately adjacent active edges (left and right spans on the scan line) to be located quickly. These edges are checked for possible crossings, the edge list is updated to maintain the x-sort, and any detected crossings are inserted into the event queue for subsequent processing.

The event queue allows insertion of an arbitrary element and deletion of the minimum element. The edge queue admits insertion, deletion, predecessor and successor operations for an arbitrary element. All of these can be performed in at most logarithmic time in a 2-3 tree.

The running time of the algorithm is proportional to the number of crossings (all crossings are visible in a haloed rendering) times the logarithm of the number of elements in the queues. The queues are always linear in the number of edges, so the overall running time is proportional to the number of crossings times the logarithm of the number of edges. This is clearly not linear in the visible complexity (number of crossings) so there is still a margin for further improvement.

We have not investigated the possibility for parallel processing in this algorithm. Clearly "nonoverlapping" critical events can be

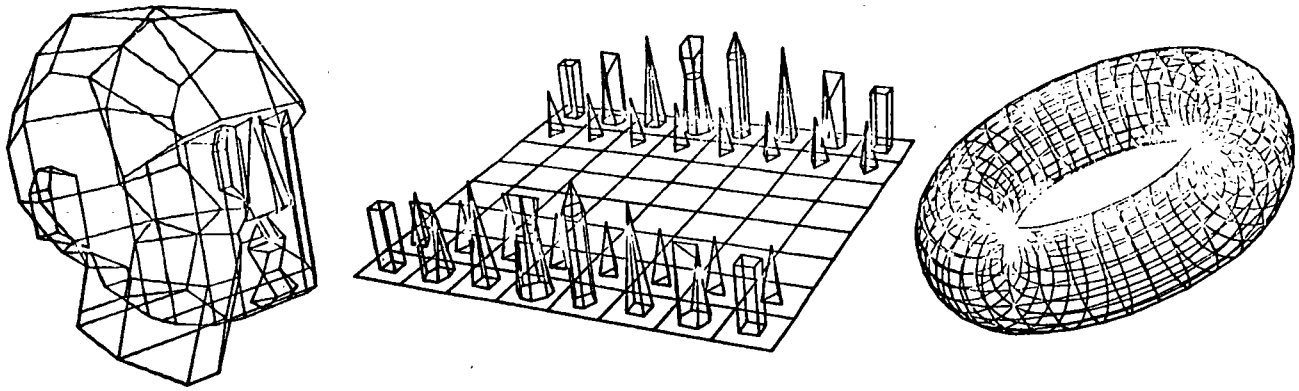

Figure 4(a). A wire-frame rendering.

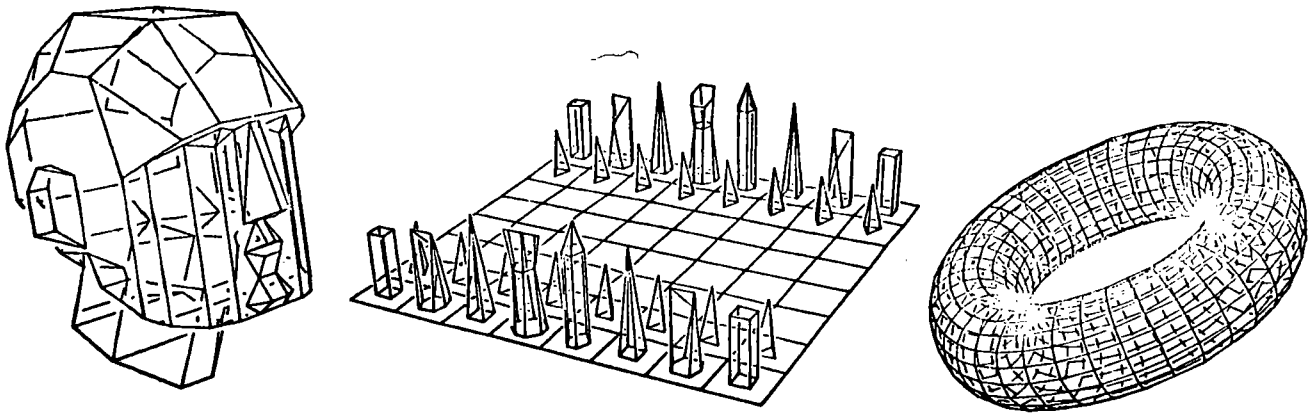

Figure 4(b). A haloed rendering.

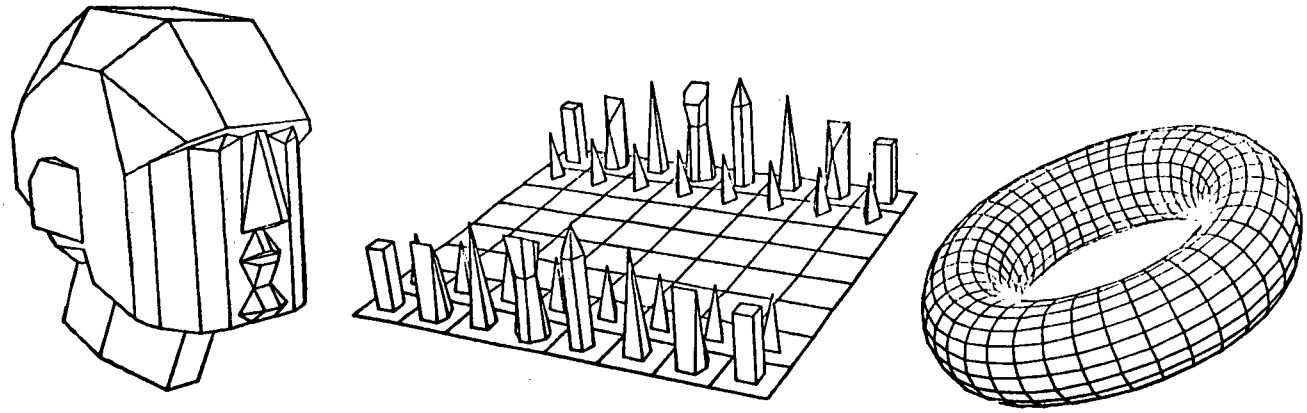

Figure 4(c). A visible surface rendering.

handled in parallel. Ensuring that this is done correctly, and detecting cases in which it cannot be handled in parallel, requires further investigation.

We remark in passing that one of the difficulties which Appel, Rohlf and Stein encountered was the problem of deciding just how much of a line should be haloed. Our implementation suffers in this respect (it is particularly noticeable in the third example of Figure 4). One possible solution is to use a two-pass algorithm which treats "horizontal" and "vertical" lines separately, a technique already employed in our implementation of Watkin's algorithm [A].

6. Related Problems

The haloing algorithm is only one instance of a problem which can be solved using a scan line approach. Beretta and Nievergelt have observed the same phenomenon which prompted us to look at scan line algorithms:

> Recent progress in computational geometry has made it possible to isolate the algorithmic core of scan conversion algorithms, to assess their generality, and to describe fairly clearly the kinds of geometric and topological questions they can answer [BN].

Their notion of a plane-sweep algorithm which sorts line segments according to x and then processes cross-sections in y is what we have been calling a scan line algorithm. By way of example, we point out two other problems for which scan line (plane-sweep) algorithms provide efficient solutions.

The algorithm presented by Brassel and Fegeas [BF] for cross-hatching can be improved by viewing it as a version of hidden surface processing. In applications such as cartography a number of polygons frequently need to be cross-hatched. Typical algorithms have substantial overhead computing the intersections of cross-hatching lines with the polygons. Reformulating the algorithm as a top-to-bottom scan allows most of the calculations to be done incrementally, taking full advantage of coherence properties in the cross-hatched regions.

Bentley, Ottman, Six and Wood [BO, SW] have algorithms for determining the overlap of a set of rectangles in two dimensions. Their algorithms are yet another rediscovery of the scan line approach, although with some additional features to detect rectangles which are entirely included within others. Again, the basic idea is a scan line approach which detects crossings incrementally in a single pass over the data.

A number of other examples of scan line or plane-sweep algorithms appear in literature [DM, LP, SH].

7. Conclusions

Our original goal was to achieve a visible surface algorithm which truly ran in time proportional to the visible complexity of a scene. To date we have not reached this goal. We are analyzing additional coherence properties which may be of help in this regard. The haloing algorithm detects all intersections. Because all of the edges are being drawn, this is not unreasonable. We are still looking for a way to capitalize on scan line coherence to avoid this work when computing visible surfaces, since for that problem the invisible crossings do not appear in the output.

The complexity analysis in this paper is worst-case. Traditionally visible surface algorithms are measured by their average behaviour, but only in terms of measurements from sample data. More emphasis on the theoretical aspects of both the worst-case and the average case complexity is certainly in order.

The primary advantage of a scan line algorithm for the visible surface problem is its ability to per-

form anti-aliasing. Z-buffer and priority algorithms fall short in this respect. By way of contrast, the z-buffer and priority algorithms have taken advantage of parallelism to a much greater degree than have the scan line algorithms [KG, P]. One reason for this may be the synchronization required between the various spans as scan lines are processed. A distributed algorithm in which separate processors are assigned to visible spans will require more elaborate communication. This is again a topic for further research.

The similarity of scan line algorithms for visible surface processing and algorithms for other geometric problems found in computer graphics and VLSI layout suggest that scan line algorithms are themselves a rich area for investigation. Our effort and the related work of Beretta and Nievergelt are first steps in this direction.

Acknowledgements

The Watkins' and haloing algorithms were implemented using the Waterloo Pascal compiler on a Honeywell 66/60. They interface to the standard graphics package developed by students in the introductory graphics course at the University of Waterloo. This work was supported by the Natural Sciences and Engineering Research Council of Canada under grants A3022 and A4037 and by a Science 67 Scholarship.

References

[A]
Michael Archuleta, private communication.

[ARS]
Arthur Appel, F. James Rohlf and Arthur J. Stein, The haloed effect for hidden line elimination, Computer Graphics 13:2, pp. 151-157 (August 1979).

[BO]
Jon L. Bentley and Thomas A. Ottman, Algorithms for reporting and counting geometric intersections, IEEE Transactions on Computers C-28:9, pp 643-647 (September, 1979).

[BW]
Jon L. Bentley and Derick Wood, An optimal worst-case algorithm for reporting intersections of rectangles, Technical Report 79-CS-13, McMaster University, (1979).

[BN]
G. Beretta and J. Nievergelt, Scan conversion algorithms revisited, International Conference on Research and Trends in Document Preparation Systems, ETH, Lausanne, Switzerland, pp. 77-80 (February, 1981).

[BF]
Kurt E. Brassel and Robin Fegeas, An algorithm for shading of regions on vector devices, Computer Graphics 13:2, pp. 126-133 (August 1979).

[C]
James Clark, A VLSI geometry processor for graphics, Computer 13:7, pp. 59-68.

[Cr]
Franklin C. Crow, The aliasing problem in computer synthesized shaded images, Communications of the ACM 20:11, pp. 799-805 (November, 1977).

[DM]
David P. Dobkin and J. Ian Munro, Efficient uses of the past, Proceedings 21st Annual Symposium on Foundations of Computer Science, pp. 200-206 (October, 1980).

[HG]
Griffith Hamlin, Jr. and C. William Gear, Raster scan hidden surface algorithm techniques, Computer Graphics 11:2, pp. 206-213 (July, 1977).

[KG]
Michael Kaplan and Donald P. Greenberg, Parallel processing techniques for hidden surface removal, Computer Graphics 12:2, pp. 300-307 (August, 1979).

[LCWB]
Jeffrey M. Lane, Loren C. Carpenter, Turner Whitted and James Blinn, Scan line methods for displaying parametrically define surfaces, Communications of the ACM 23:1, pp. 23-34 (January, 1980).

[LP]
D.T. Lee and F.P. Preparata, Location of a point in a planar subdivision and its applications, SIAM Journal on Computing 6:3, pp. 594-606 (September, 1977).

[NS1]
William M. Newman and Robert F. Sproull, Principles of Interactive Computer Graphics, 1st edition, McGraw-Hill (1973).

[NS2]
William M. Newman and Robert F. Sproull, Principles of Interactive Computer Graphics, 2nd edition, McGraw-Hill (1979).

[P]
Frederic I. Parke, Simulation and expected performance analysis of multiple processor z-buffer systems, Computer Graphics 14:3, pp. 48-56 (July, 1980).

[SW]
H.-W. Six and Derick Wood, The rectangle intersection problem revisited, Technical Report 79-CS-24, McMaster University, (1979).

[SH]
Michael Ian Shamos and Daniel Hoey, Geometric intersection problems, Seventeenth Annual Symposium on Foundations of Computer Science, pp. 208-215 (October, 1976).

[SSS]
Ivan E. Sutherland, Robert F. Sproull and Robert A. Schumacker, A characterization of ten hidden-surface algorithms, Computing Surveys 6:1, pp. 1-55 (March 1974).

[W]
Gary S. Watkins, A Real-Time Visible Surface Algorithm, Computer Science Department, University of Utah UTECH-CSc-70-101 (1970).

[Y]
F. Frances Yao, On the priority approach to hidden-surface algorithms, Proceedings 21st Annual Symposium on Foundations of Computer Science, pp. 301-307 (October, 1980).