Adaptive Grid for Polyhedral Visibility in Object Space: an Implementation

W. R. FRANKLIN* AND V. AKMAN†‡

* *Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, N.Y. 12180, USA*

† *Department of Computer Science, University of Utrecht, Budapestlaan 6, P.O. Box 80.012, 3508 TA Utrecht, The Netherlands*

This paper presents an implementation of Franklin's object space hidden surface algorithm for polyhedral scenes.[4] It is known that if the faces are independently and identically distributed this algorithm performs in time linear in the number of faces, and in particular is not affected by the depth complexity. The algorithm overlays a grid on the scene with fineness depending on the statistics of the edges and the faces. It then preprocesses the edges and the faces in a grid data structure so that distant edges and faces will not be compared.

The implementation of the algorithm on a Prime 750 using Ratfor shows that it is indeed very fast for random and structured scenes alike.

Received May 1986, revised October 1986

1. INTRODUCTION

Hidden surface removal is now a well-researched and reasonably well-developed area of computer graphics. There are many algorithms, most of which work in the *image space* (the realm of graphics output devices) as opposed to the *object space* (the realm of precise computations). Object-space algorithms for visibility are important because in nearly all computer-aided design applications the exactness of the computation is a prerequisite for integrity. For a rather dated but otherwise perfect survey of hidden surface research see Ref. 13. Important algorithms include (but are not limited to) those by Newell,[11] Weiler and Atherton,[15] Whitted,[16] Franklin,[4] Sechrest and Greenberg[12] and Carpenter.[2] Newell's solution is a very elegant technique known as the *painter's algorithm*. Weiler and Atherton presented what may essentially be summarized as an inverse painter's algorithm. Whitted gave an extension of *ray tracing*, another elegant technique originally invented by Arthur Appel. Sechrest and Greenberg's algorithm is based on image coherence. Finally, Carpenter described an extension of the celebrated *z-buffer* technique and called it the *A-buffer*.

In this paper we give an implementation of Franklin's object-space hidden-surface algorithm for polyhedral scenes.[4] Franklin's algorithm is based on the idea of an *adaptive* (*variable*) *grid*. For a detailed study of adaptive grids the reader is referred to Refs 3 and 6; applications of adaptive grids similar to the one reported in this paper are given in Refs 8 and 4. The adaptive-grid data structure is a uniform grid overlaid on the scene. The fineness of the grid is some experimentally determined function of the statistics of the objects, e.g. average edge length, average face area, number of edges, etc. The adaptive grid is especially suitable to tell efficiently which few pairs of a large number of short edges intersect. In this case the average execution time is linear in the expected number of intersections plus the number of edges, thus optimal within a multiplicative constant for a wide range of scenes.

‡ To whom correspondence should be addressed at: Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands.

The input to our program (which is written in Ratfor[10] and runs on a Prime 750) is a set of polyhedra. The output of the program is a set of visible polygons with associated shading information.

2. DATA STRUCTURES

The algorithm uses several data structures in addition to the input data structures consisting of FACES, the set of faces and EDGES, the set of edges of the objects in the scene. In the implementation, the inputs to the algorithm are FACES and VERTICES, the set of vertices. EDGES are obtained from this information. A simple *boundary representation* is assumed for an object. In particular, each face is given as an ordered list of its vertices around it.

This algorithm works on an object consisting of a set of disjoint faces. A complication to cover the situation where faces may intersect could be added, but this will be omitted here.

As an important aspect, it should be noted that all the data structures are of the abstract data type *set*. Thus, in an implementation environment which can support the set data type efficiently, all the operations would work on sets. This is a feature which not only simplifies the algorithmic description but also shows its key characteristics that make it a good candidate for implementation in silicon. In particular, the algorithm lends itself to being executed in parallel. It spends most of its time either (i) performing the same operation on each element of a set, or (ii) sorting the elements of a set. There are various ways to make the former task parallel, the most obvious being an N-way parallelism for an N-element set. As for the latter, there is now a considerable amount of work on parallel sorting algorithms.[14]

We now summarize the other data structures used by the algorithm.

(i) A regular G by G grid of square cells superimposed on the projected scene. Each cell C of the grid has the following three data structures which are initially empty. (1) The name of the closest *blocking face* BF(C), if any, of this cell. BF(C) covers C completely and thus blocks everything behind. (2) The set of faces FF(C) which

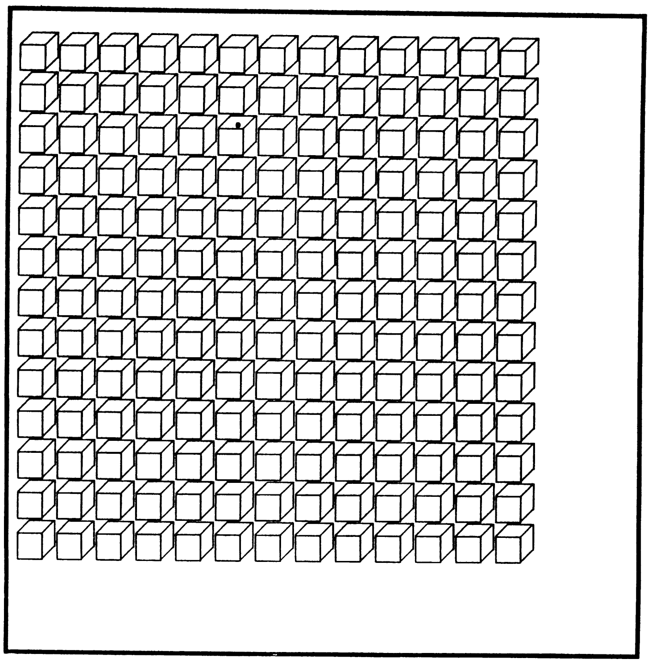

Figure 1. A 13 × 13 array of cubes with hidden lines removed.

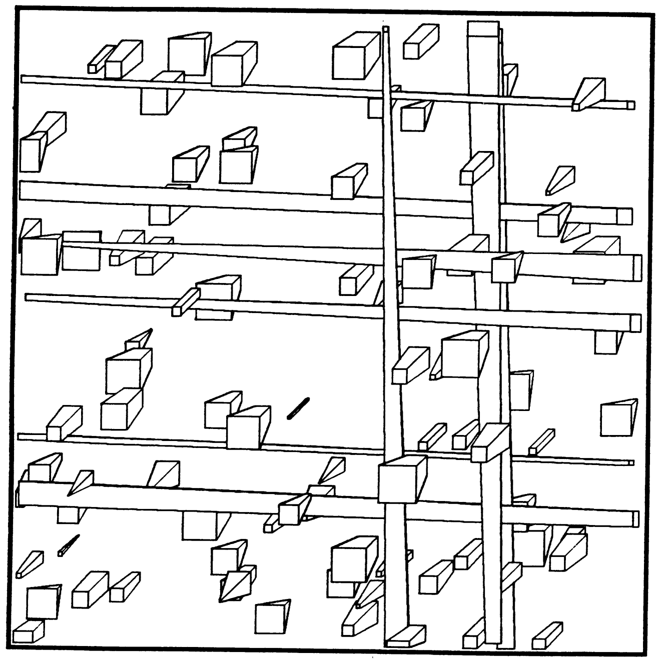

Figure 2. A random arrangement of blocks with hidden lines removed.

intersect C and are in front of BF(C). (3) The set of edges FE(C) which intersect C and are in front of BF(C). It is noted that, without loss of generality, the *viewpoint* is at the infinity in the positive z-direction and orthographic projections of the objects are taken in the xy-plane.

(ii) The set XSECT of pairs of intersecting edges (along with their intersection points). XSECT contains all the visible and some of the hidden intersections.

(iii) The set VISS of visible edge segments. The elements of VISS make up the edges of VISP given below.

(iv) The set VISP of visible polygons. Each element of VISP is a connected visible part of a face.

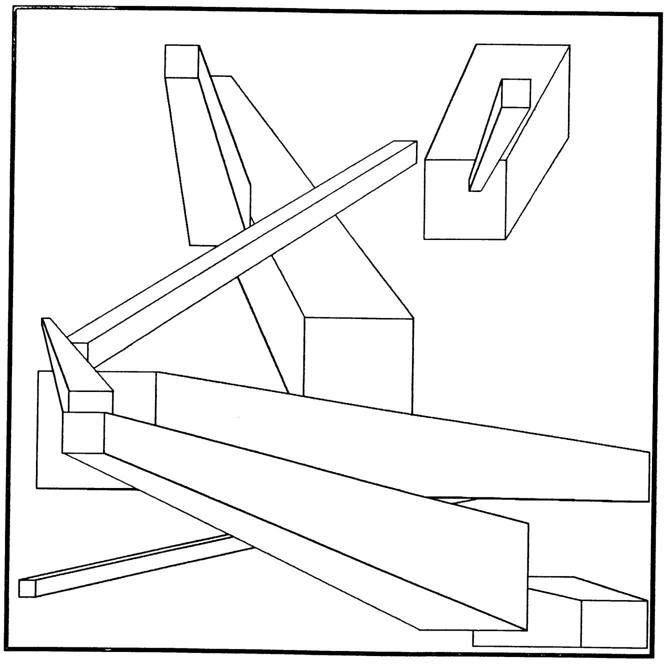

Figure 3. Another random arrangement of blocks with hidden lines removed.

3. THE ALGORITHM (HIGH-LEVEL DESCRIPTION)

The algorithm first transforms the scene to fit inside a unit square at the origin using known techniques. Then it decides on the grid size G. The choice of G to obtain optimal execution time is heuristically made and is based on the statistics of the edges and the faces. (Fine-tuning the algorithm by choosing a suitable G is discussed later.) After these preparations, the algorithm proceeds as follows.

Algorithm A

A1. Overlay a G by G grid on to the scene and initialise the data structures for each grid cell. Initialise XSECT, VISS and VISP.

A2. For each projected face F in FACES do the following. Determine the grid cells CELLS that F partly or wholly covers. For a cell C in CELLS see if C has a blocking face BF(C) which is always in front of F throughout C. If so, this C is no more considered with F. Otherwise, F itself may be a blocking face of C. In that case, BF(C) is updated as F. If none of the previous cases holds, then F is added to FF(C). This is repeated for each C in CELLS.

A3. For each grid cell C do the following. Compare BF(C) to all Fs in FF(C). Delete from FF(C) all Fs that are behind BF(C) throughout cell C. The reason that there may be faces to delete is that BF(C) may have been changed as the faces were processed in step A2.

A4. For each projected edge E in EDGES do the following. Determine the grid cells CELLS to which E partly or wholly belongs. For a cell C in CELLS see if E is behind BF(C). If not, then E is added to FE(C). This is repeated for each C in CELLS.

A5. For each grid cell C do the following. For each pair of projected E1 and E2 in FE(C) see if E1 and E2 intersect. If so, and if the intersection point is inside C, add E1 to the list of edges intersecting E2, and E2 to the

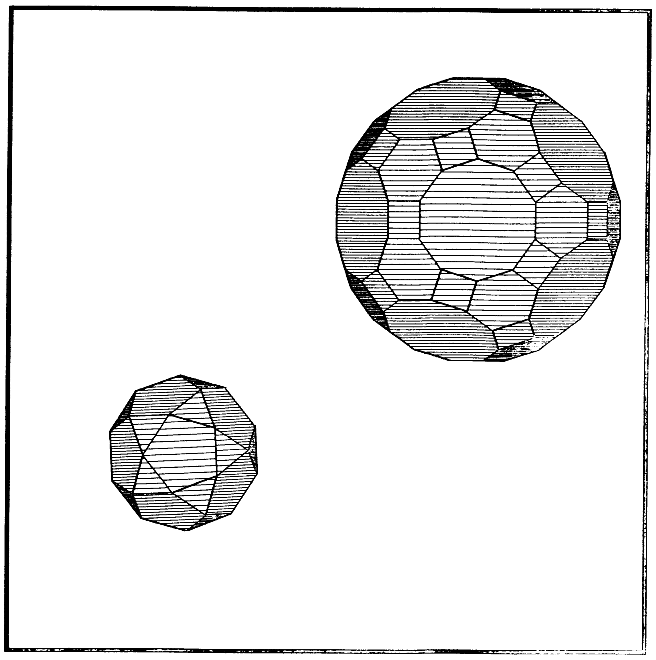
Figure 4. Two semi-regular polyhedra with visible surfaces cross-hatched.

list of edges intersecting E1. Due to the way XSECT was defined earlier, we are in fact adding pairs to XSECT, i.e. (E1, E2) and (E2, E1) with an implied intersection point. Note however that once the members of XSECT are all found, it is trivial to obtain the list of edges intersecting each edge via sorting.

A6. For each projected edge E in EDGES do the following. Sort the intersection points of E along it and use them to split E into segments SEGS. (An edge with no intersections is just one segment.) Then for each segment S in SEGS, if the midpoint of S is visible add S to VISS. (The visibility of the midpoint is determined by comparing it against BF(C) and the faces in FF(C), where C is the cell including the midpoint.)

A7. Determine the regions VISP of the straight-edge planar graph formed by the elements of VISS. This is done by the procedure given in Ref. 9. The idea is to form a data structure using the edges and to navigate through it to obtain the regions one after another, with special care exercised to handle regions within regions correctly.

A8. For each polygon P in VISP do the following. Let I be a point inside P and let C be the cell which includes I. Compare I against FF(C). Let F be the closest face in FF(C) whose projection contains I. Then P corresponds to a visible part of F and should be given an appropriate shading value. If I is not on any element of FF(C), it corresponds to the *background* and is given the default shading value.

End

4. THE ALGORITHM (DETAILS)

Proving the above algorithm correct is easy since it is a refinement of the following näive algorithm which is of quadratic complexity in the number of edges.

Intersect all the projected edges pair by pair, to cut each edge into segments. (Each segment is either completely visible or else completely hidden.) Compare each segment against all faces to

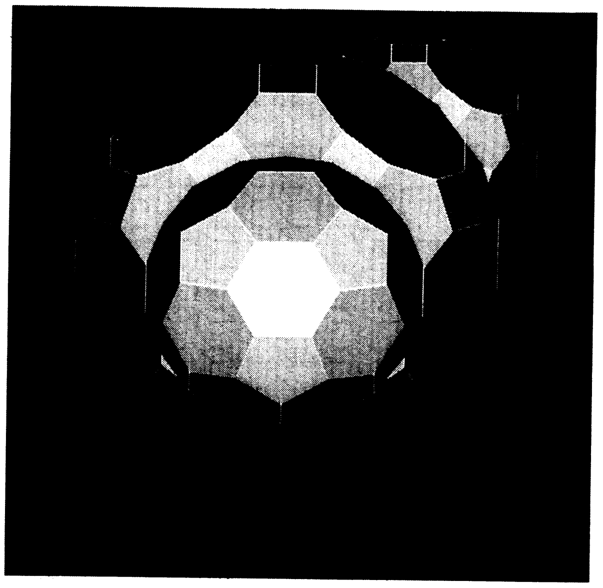
Figure 5. Three semi-regular polyhedra with visible surfaces painted.

determine its visibility. Construct the visible regions from the visible segments and compare each region against all faces to see which it corresponds to and shade it accordingly.

There are a number of rather low-level details that are omitted from the description of the algorithm in the previous section. These are crucial for an implementation and are mentioned now.

1. As for the optimal grid size, the following is found to be a suitable value (although C here is an unspecified constant less than 1 and is heuristically determined)

$$G = \text{floor}\,(C \min(N^{\frac{1}{2}}, L^{-1}))$$

Here N is the number of edges and L is the average projected edge length. An experimentally observed fact about the grid size is that changing it within a factor of two makes little difference. It is also apparent that C will be hard to predict in general since it depends on the relative speeds of various parts of the program implementing the algorithm. It is emphasized that L is the average length as a fraction of the image size. The algorithm has been described as an object space algorithm, so the reader should not interpret L as a distance in the original world coordinates.

2. At step A2 of the algorithm, determining the cells is not done by comparing each of the cells with F. A good approximation is the cells belonging to the bounding box of F. A better determination can be made by considering F's edges. It is emphasized that if extra cells are included the algorithm will still perform correctly, albeit more slowly.

3. At step A4, the visibility of E versus BF(C) is seen by clipping E at the borders of C to E′ and testing whether the endpoints of E′ are both behind BF(C).

4. At step A5, the point-in-cell test is necessary. This catches duplications caused by the fact that the same pair of edges may pass through several cells. In fact, the total number of duplicate intersections gives a good idea about the correctness of a particular grid size. If only the visible edges are wanted and not the visible surfaces, one must compute which one of E1 and E2 is in front of the other in three dimensions at the point of intersection. Assuming

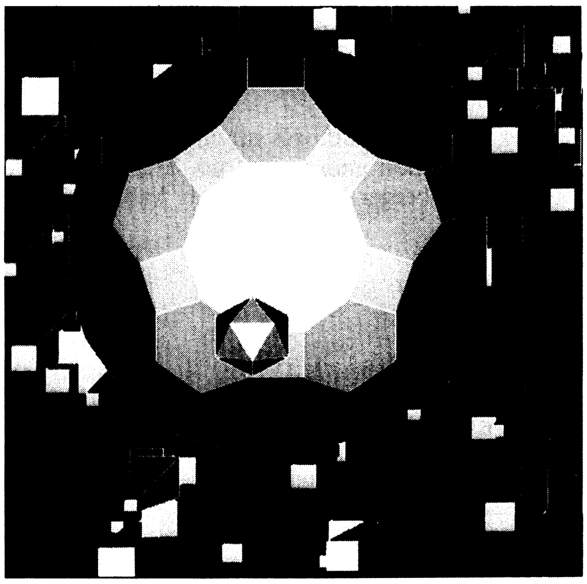

Figure 6. A random arrangement of blocks with visible surfaces painted.

that El is in front, the intersection point is only added to E2's list.

5. At step A6, a face F of FF(C) hides the midpoint in question if it satisfies two conditions: (i) the point is behind F's plane in three dimensions, and (ii) the projected point is inside the projected F.

6. At step A8, the interior point I must be taken very close to the boundary of F in order to avoid incorrect pictures. Notice that an arbitrary interior point of F will not do since there may be separate components enclosing each other.

5. THE IMPLEMENTATION

The implementation of an algorithm is a good way of testing its viability and effectiveness. This is an especially important issue in graphics and computational geometry, since conceptually easy tasks such as applying the Boolean operators to polyhedra or constructing the Voronoi diagram of a set of points turn out to be difficult when it comes to handling all the special cases.

Our implementation of the algorithm outlined above was done in Ratfor, a structured variant of Fortran.[10] The program, which is about 3500 lines of moderately commented code, runs on a Prime 750. After reading the input polyhedra the program works basically in one of three modes, as follows.

Computation. In this mode, no graphics is done and no results are written to a file; they are left in core. This mode exists to see the effectiveness of the program in terms of execution speed.

Graphics. The results are drawn on a vector terminal (e.g. Imlac) as they are being computed. There is another facility which takes a picture file generated by the next mode of operation and displays it on a raster screen (e.g. DeAnza).

Output. The results of the visibility computation (i.e. visible polygons along with intensities) are written to a file for device-independent and off-line graphics.

The program gathers extensive statistics about the objects, the usage of the data structures, and its execution. The polyhedra are given as a set of vertex coordinates and a set of faces which consist of the vertex names making up their boundary. The edges are obtained by the program. In general no extra information is input for an object (such as the face plane equations.) If a particular information is needed, it is computed and either stored (when necessary) or consumed. The program has been tested in several modes for a particular scene to determine the bottlenecks. To see the effectiveness of the grid method, the first mode of operation mentioned above has been chosen. It has especially been observed that the preprocessing step of the algorithm (i.e. entering all the edges and the faces to the grid data structure) is extremely fast, e.g. a few per cent of the overall execution time. Other relevant statistics that the program collects include average/maximum/minimum edge lengths, average/maximum/minimum face areas, number of cells with a blocking face, number of edges belonging only to a grid cell/grid column/grid row, number of edge intersections in the projection, number of duplicate edge intersections (which are due to the use of a grid), average number of edges/faces per cell, number of segments, and number of visible segments. By careful study of these statistics it is possible to determine a close-to-optimal grid size for a wide range of scenes. However, it remains as a valuable heuristic that the execution time of the algorithm is not affected by changing the grid size within a factor of two.

A number of pictures illustrating the type of scenes that the program handles are shown in Figures 1–6. The first four pictures are taken from an Imlac vector terminal via hardcopy. Figs 5 and 6 are from a DeAnza raster display via hardcopy. (These pictures were made at the Image Processing Laboratory of RPI.) All these scenes took only a few seconds to compute, the largest requiring about ten seconds.

6. FINAL REMARKS

We have successfully tested the adaptive-grid data structure on a key problem in graphics, namely, polyhedral visibility computation; similar previous tests are given in Refs 5 and 8. It becomes clear that the data structure has a very good average case behaviour. It is noted that a common method of line intersection in computational geometry is some sort of *plane sweep* as reported by Bentley and Ottmann.[1] The problem with Bentley and Ottmann's method is that it is difficult to program and hides large constants inside big-oh bounds. Nevertheless, the plane sweep has an excellent worst case time while the worst case time for the adaptive grid is bad, although it works perfectly on a wide variety of scenes. Franklin also remarks that the adaptive grid can be used to solve some problems efficiently that the sweep method cannot handle fast, e.g. interference detection between two parts such as a robot arm and a workpiece.[7]

Still the best environment that the adaptive grid approach most efficiently handles is that of a scene in which few pairs of a large number of short edges intersect. Accordingly, the kind of object configuration that gives the worst case is a large number of long edges which cover a lot of cells and intersect wildly. Thus the worst-case timing may be as bad as quadratic.

Although it can be varied within a factor of two without changing the performance of the algorithm appreciably, the grid size G is a potential area of investigation. It would be worthwhile to collect statistics on a scene before processing to determine the optimal G. The reader is referred to Ref. 5 for a thorough discussion of this subject. If the scene is not homogeneous, a *hierarchical* (as opposed to regular) grid may be faster. This obviously would make some steps of the algorithm more complex. Besides, an adversary can find sequences of scenes that still execute slowly.

Acknowledgements

The work of the first author is supported by the National Science Foundation under grant no. ECS-8351942, and by the Schlumberger-Doll Research Labs, Ridgefield, CT.

REFERENCES

1. J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers* **28** (9), 643–647 (1979).
2. L. Carpenter, The A-buffer, an antialiased hidden surface method. *ACM Computer Graphics* **18** (3), 103–108 (1984).
3. W. R. Franklin, *Combinatorics of Hidden Surface Algorithms*, Harvard University, Center for Research in Computing Technology, Cambridge, Mass. Report TR-12-78 (1978).
4. W. R. Franklin, A linear time exact hidden surface algorithm. *ACM Computer Graphics* **14** (3), 117–123 (1980).
5. W. R. Franklin, An exact hidden sphere algorithm that operates in linear time. *Computer Graphics and Image Processing* **15** (4), 364–379 (1981).
6. W. R. Franklin, Adaptive grids for geometric operations. *Proceedings of the Sixth International Symposium on Automated Cartography* **2**, Ottawa, 230–239 (1983).
7. W. R. Franklin, Validation of efficient dynamic computer aided design algorithms on large databases, ECSE Dept., Rensselaer Polytechnic Institute, Troy, New York (1984). Manuscript.
8. W. R. Franklin and V. Akman, *A Simple and Efficient Haloed line Algorithm for Hidden Line Elimination*, University of Utrecht, Department of Computer Science, Report RUU-CS-85-28. Utrecht, The Netherlands (1985).
9. W. R. Franklin and V. Akman, Reconstructing visible regions from visible segments, to appear in BIT.
10. B. W. Kernighan and P. J. Plauger, *Software Tools*. Addison-Wesley, Reading, Mass. (1976).
11. M. Newell, R. Newell and T. Sancha, A new approach to the shaded picture problem. *Proceedings of the ACM National Conference*, p. 443 (1972).
12. S. Sechrest and D. P. Greenberg, A visible polygon reconstruction algorithm, *ACM Transactions on Graphics* **1** (1), 25–42 (1982).
13. I. E. Sutherland, R. F. Sproull and R. Schumacker, A characterization of ten hidden surface algorithms. *ACM Computing Surveys* **6** (1), 1–55 (1974).
14. L. G. Valiant, Parallel computation. In *Foundations of Computer Science IV* (*Distributed Systems*: *Part 1, Algorithms and Complexity*), edited J. W. de Bakker and J. van Leeuwen, pp. 35–48. Mathematical Centre Tracts, Amsterdam (1983).
15. K. Weiler and P. Atherton, Hidden surface removal using polygon area sorting. *ACM Computer Graphics* **11** (2), 214–222 (1977).
16. T. Whitted, 'An improved illumination model for shaded display. *Communications of the ACM* **23** (6), 343–349 (1980).