The Object Complexity Model for Hidden-Surface Elimination

Edward F. Grove[*] *T. M. Murali*[†] *Jeffrey Scott Vitter*[‡]

Department of Computer Science
Duke University
Durham, NC 27708
Email:{efg,tmax,jsv}@cs.duke.edu

Abstract

We define a new complexity measure, called *object complexity*, for hidden-surface elimination algorithms. This model is more appropriate than the standard scene complexity measure used in computational geometry for predicting the performance of these algorithms on current graphics rendering systems.

We also present an algorithm to determine the set of visible windows in 3-D scenes consisting of n isothetic windows. It takes time $O(n \log n)$, which is optimal. The algorithm solves in the object complexity model the same problem that Bern [Ber] addressed for the standard scene complexity model.

1 The Object Complexity Model

Hidden-surface removal is a central problem in computer graphics. Given a collection of objects in three-dimensional space, we want to render the objects as they would be seen from an observer at a specified viewpoint. Determining which objects are obscured is a major part of the rendering process.

Various hidden surface algorithms have been proposed in the computational geometry literature. Worst case optimal algorithms are presented in [Dev, McK]. Algorithms sensitive to the number of intersections of the objects on the viewing plane are presented in [Goo, Nur, Sch]. More recently, algorithms have been *output-sensitive*; that is, the running time of these algorithms depends on the input size and some feature of the output, typically the *scene complexity*, which is the number of visible line segments in the final rendered scene. Such algorithms are found in [Ber, GAO, KOS, PrV, PVYa, PVYb, ReS]. More output-sensitive algorithms which do not assume that a back-to-front ordering of the objects in the scene with respect to the viewpoint exists are described in [Bera, Berb, BHO, BeOa, BeOb]. The fastest algorithm for hidden-surface elimination takes time $O(n^{2/3+\epsilon}k^{2/3} + n^{1+\epsilon})$, where n is the size of the input and k is the scene complexity [AgM].

The computer graphics community, which is the source of the problem, has studied hidden-surface removal extensively. Sutherland, Sproull and Schumacker [SSS] survey early hidden surface removal algorithms used in graphics. Recent approaches [Air, Tel] have studied *walk-through systems*. The aim here is to visually simulate the experience of walking inside a building using an architectural model of the building. The simulation achieves realism when at least ten scenes are generated and displayed per second.

[*]Support was provided in part by Army Research Office grant DAAH04–93–G–0076.

[†]This author is affiliated with Brown University. Support was provided in part by NSF research grant CCR–9007851 and by Army Research Office grant DAAL03–91–G–0035.

[‡]Support was provided in part by NSF research grant CCR–9007851 and by Army Research Office grant DAAH04–93–G–0076.

The model of scene complexity used in computational geometry does not match the constraints of the hardware and software typically used for hidden-surface elimination and rendering. We propose a more realistic model of complexity, called *object complexity*, in which the running time is measured in terms of the input size and the number of objects visible in the scene.

Object complexity is motivated by use of z-buffer rendering hardware [Cat, FDF], which is found in high-performance graphics machines like the Silicon Graphics RealityEngine [Ake]. The z-buffer sequentially processes the objects input to it, and updates the pixels of the display corresponding to each object, based on distance information. Assuming that the input objects are triangles, the cost of z-buffer processing depends on the number of triangles processed by the z-buffer except in the atypical case where the triangles are extremely large, when the processing cost is dominated by the number of pixels covered by the triangles.

The z-buffer can be implemented very fast in hardware. For example, the Silicon Graphics RealityEngine is capable of rendering a million triangles per second. Fast as the z-buffer is, datasets are becoming so huge that even the fastest z-buffers cannot render them in real time. Some aircraft models consist of tens of millions of triangles, and submarine models may have a billion triangles. This problem is compounded for interactive real-time applications like walk-through systems [Air, Tel]. In such applications, new scenes need to be generated at least 10 times a second. Processing all the input through the z-buffer at these rates is currently not possible. If the visible scene is to be displayed in real time, it is imperative that the z-buffer should process only a small superset of the visible triangles. This strongly motivates the development of provably fast algorithms for determining a small superset of the visible triangles (a.k.a. "culling") so that the requirements on z-buffers are eased.

Object complexity is always less than n, the number of objects in the input and hence can be much less than the scene complexity (which can be $\Omega(n^2)$). This happens, for example, when the viewpoint is at $z = +\infty$ and the scene contains $n/2$ thin rectangles parallel to the x-axis lying directly above $n/2$ thin rectangles parallel to the y-axis. Algorithms whose running time depends on scene complexity can be used trivially to determine visible objects by outputting all the objects that contain segments in the view. However, in the worst case, this might entail spending $\Omega(n^2)$ time to output $O(n)$ distinct objects.

The Object Complexity Model When doing hidden-surface elimination, we have two goals:

- Minimize internal computation time.
- Find a small superset of the visible triangles.

Since these two measures are incomparable, we wish to minimize both in our algorithms.

2 Hidden-Surface Elimination in Static Scenes

Our model of object complexity is relevant not only for dynamic scenes as mentioned above but also for static scenes like the one we address in this paper. Most machines do not have z-buffers and must resort to software z-buffers, or else they have hardware z-buffers that perform at a fraction of the speed of a state-of-the-art z-buffer like the RealityEngine. In such cases, the speed of the rendering process is considerably heightened by a fast and efficient software algorithm which culls all but a small superset of the visible triangles and feeds only these to the z-buffer. Even in machines with state-of-the-art z-buffers, faster CPUs can put the bottleneck of rendering back on the z-buffer.

The problem we study is finding the exact set of rectangles visible from the point $z = +\infty$ in a set of n rectangles with sides parallel to the $x-$ and $y-$ axes. We solve this problem in optimal $\Theta(n \log n)$ time. Bern [Ber] addresses this problem for the standard scene complexity model. Our algorithm is novel because we cannot afford to maintain information about all the visible segments explicitly (like he does). We maintain this information implicitly by using the segment tree in a clever manner. A full version of this extended abstract appears in [GMV]. All proofs are given in detail there.

3 Window Visibility Problem

Our input is n rectangles, each with sides parallel to the $x-$ and $y-$ axes. The object is to report the rectangles visible from the point $z = +\infty$. Each rectangle R is specified by five numbers, $R.x_1, R.x_2, R.y_1, R.y_2$, and $R.z$ such that $R = [x_1, x_2] \times [y_1, y_2] \times [z, z]$, where $x_1 < x_2$ and $y_1 < y_2$. If two edges belonging to different rectangles project to the same line segment on the xy-plane but have different z coordinates, the edge with higher z coordinate is considered to obscure the edge with lower z coordinate. This problem arises in windowing systems where windows are drawn on the screen according to a priority assigned to each window.

Theorem 1 *In the algebraic decision tree model, any algorithm that determines which of n rectangles with sides parallel to the x- and y- axes are visible from $z = +\infty$ requires $\Omega(n \log n)$ tests.*

4 Algorithm

We sweep a plane perpendicular to the x-axis from $x = -\infty$ to $x = +\infty$. Event points of the sweep are the coordinates of the vertical edges (the edges parallel to the y-axis) of the rectangles. If more than one vertical edge share the same x coordinate, they are processed in decreasing order of z coordinate with right edges processed before left edges[1]. The intersections of the rectangles with the sweep plane are stored in a segment tree $\mathcal{T}$. We define the segment tree below. In what follows, the left and right children of node v in the segment tree are u and w, respectively.

Let Y be the set of y-coordinates of the endpoints of the n segments in S. The elements of $Y \cup \{-\infty, \infty\}$ partition the y-axis into at most $2n+1$ intervals of the form $[y_i, y_{i+1}), 1 \leq i \leq 2n+1$, where $y_i, 1 \leq i \leq 2n+2$, is the ith smallest element in $Y \cup \{-\infty, \infty\}$. The segment tree $\mathcal{T}$ is a height balanced binary tree constructed on the elements of $Y \cup \{-\infty, \infty\}$. Each node v of $\mathcal{T}$ is associated with an interval called its *basic segment* and denoted by b_v. If v is the ith leaf of $\mathcal{T}$ (counting from left to right), then b_v is $[y_i, y_{i+1})$. If v is an internal node of $\mathcal{T}$, then $b_v = b_u \cup b_w$. See [Meh, PrS] for more details on segment trees.

A *cross-section* is the one-dimensional intersection of a rectangle with the sweep plane. Each such cross-section is stored as $O(\log n)$ basic segments in $\mathcal{T}$
Lspace [PrS]. If an event point corresponds to the left edge of a rectangle R, the corresponding cross-section is inserted in $\mathcal{T}$ using procedure LEFT-INSERT and each cross-section it is divided into is checked for visibility. If an event point corresponds to the right edge of a rectangle R, the corresponding cross-section is deleted from $\mathcal{T}$ using procedure RIGHT-DELETE and cross-sections which become visible as a result of this deletion are reported. Each basic segment a rectangle is stored as is reported at most once. Hence a rectangle may be reported as visible $O(\log n)$ times.

The following fields are stored at each node v in $\mathcal{T}$.

1. b_v: the basic segment associated with v.
2. v_{mid}: the midpoint of b_v.
3. $\mathcal{H}_v$: a heap storing the cross-sections stored at v sorted in decreasing order of z. Each element of $\mathcal{H}_v$ has a flag *unrep* which is true iff that element has not been reported as visible so far. top($\mathcal{H}_v$) returns the cross-section of maximum z coordinate stored in the heap.
4. l_v: the height of the lowest visible cross-section stored in the subtree rooted at v. If there is no such lowest visible segment then l_v is $-\infty$. Visibility is with respect to the cross-sections stored in the subtree rooted at v. This field can be calculated as follows.
$l_v = \max\{\min\{l_u, l_w\},\ \mathrm{top}(\mathcal{H}_v).z\}$
5. h_v: the height of the highest visible unreported cross-section (which is the top of the heap of some node) in the subtree rooted at v. Visibility is with respect to the cross-sections stored in the subtree rooted at v. If there is no such unreported basic segment h_v is $-\infty$. The h_v field can be calculated as follows.

[1]For a rectangle $[x_1, x_2] \times [y_1, y_2] \times [z, z]$, the *left edge* is the edge with x-coordinate x_1 and the *right edge* is the edge with x-coordinate x_2.

```
    h_v = max{h_u, h_w};
    if (top(H_v).unrep = false) and (top(H_v).z > h_v) then
      h_v = -∞;
    else
      h_v = max{h_v, top(H_v).z};
```

LEFT-INSERT(R, S, *root*), where S is the background rectangle with $S.z = -\infty$ and *root* is the root of $\mathcal{T}$, inserts the cross-section of a rectangle R into $\mathcal{T}$ by dividing it into $O(\log n)$ cross-sections. At each node where the cross-section of R is stored, it is checked for visibility.

```
procedure LEFT-INSERT(R: rectangle, S: rectangle, v: segment tree node)
    if [R.y1, R.y2] ⊆ b_v then
      insert R into H_v and set the unrep field of R in H_v to true;
      if (R.z ≥ l_v) and (R.z ≥ S.z) then
        report R as visible and set the unrep field of R in H_v to false;
    else
      if (S.z < top(H_v).z) then S = top(H_v);
      if (R.y1 ≤ v_mid) then LEFT-INSERT(R, S, u);
      if (v_mid ≤ R.y2) then LEFT-INSERT(R, S, w);
    update l_v and h_v;
```

RIGHT-DELETE(R, S, *root*), where S and *root* are as defined in LEFT-INSERT, deletes the cross-section of rectangle R from $\mathcal{T}$. RIGHT-REPORT is called at each node where a visible cross-section of R is deleted.

```
procedure RIGHT-DELETE(R: rectangle, S: rectangle, v: segment tree node)
    if [R.y1, R.y2] ⊆ b_v then
      delete R from H_v;
      update h_v, l_v;
      if ((S.unrep = true) and (S.z ≥ l_v))
        report S as visible;
        set the unrep field of S in T to false;
      else
        if (R.z ≥ l_v) and (R.z ≥ S.z) then
          RIGHT-REPORT(S, v);
    else
      if (S.z < top(H_v).z) then S = top(H_v);
      if (R.y1 ≤ v_mid) then RIGHT-DELETE(R, S, u);
      if (v_mid ≤ R.y2) then RIGHT-DELETE(R, S, w);
    update l_vand h_v;
```

RIGHT-REPORT(S, v) is called by RIGHT-DELETE at a segment tree node v where a visible cross-section of a just-deleted rectangle R was stored. RIGHT-REPORT reports all previously unreported cross-sections that become visible as a result of the deletion of R.

```
procedure RIGHT-REPORT(S: rectangle, v: segment tree node)
    if (h_v ≤ S.z) return;
    if (S.z ≤ top(H_v).z) then
      if ( top(H_v).unrep = true) and ( top(H_v).z ≥ l_v)
        report top(H_v);
        top(H_v).unrep = false;
      S = top(H_v);
    RIGHT-REPORT(S, u);
    RIGHT-REPORT(S, w);
    update h_v;
```

5 Correctness and Analysis

The proof of correctness is given in detail in [GMV]. The analysis depends on the following key lemma which is also proven in [GMV]. We say that a node is *marked* if a rectangle is reported as visible when the node is visited by RIGHT-REPORT.

Lemma 1 *Let $\mathcal{U}$ be the subtree of $\mathcal{T}$ explored by a single call to* RIGHT-REPORT. *If two leaves of $\mathcal{U}$ are siblings and unmarked, then their parent is marked.*

Theorem 2 *The rectangles visible from $z = +\infty$ in a set of n rectangles with sides parallel to the $x-$ and $y-$axes can be reported in $O(n \log^2 n)$ time. The space used is $O(n \log n)$.*

6 An Improved Algorithm

In this section, we improve the running time of the algorithm to $O(n \log n)$. When a rectangle is reported for the first time by the above algorithm, in $O(\log n)$ time all cross-sections corresponding to it can be marked as reported. Since these cross-sections are the leaves of a subtree of $\mathcal{T}$ of size $O(\log n)$, the h_v values in the tree can be updated to reflect the changes to $\mathcal{T}$ in $O(\log n)$ time. This reduces the $O(k \log^2 n)$ component of the running time (which is hidden by $O(n \log^2 n)$ in Theorem 2) to $O(k \log n)$.

To reduce the time taken by the rest of the algorithm from $O(n \log^2 n)$ to $O(n \log n)$ we use Bern's [Ber] trick. He noted that anytime a node, v, of $\mathcal{T}$ is visited, it is enough to know just the value of $\mathsf{top}(\mathcal{H}_v)$ rather than what is stored in the entire heap. At each node v of the segment tree, the modified algorithm stores a list of values of $\mathsf{top}(\mathcal{H}_v)$. Each entry in the list has a range of x values for which it is valid. The modified algorithm simulates the old algorithm except that no insertions and deletions are made into the heaps and whenever the value at the top of a heap is needed, the correct value is taken from the corresponding list.

Once the skeleton of $\mathcal{T}$ and the event schedule have been determined, for all nodes v in $\mathcal{T}$, we calculate a sorted list of $R.z$ values for all rectangles R ever stored at v. We can do this in $O(n \log n)$ time. We also keep a sorted list of $R.x_1$ and $R.x_2$ values for each node corresponding to the insertions and deletions made at that node.

For a single node v, we can represent the sorted list of $R.z$ values by ranks between 1 and m, where m is the total number of rectangles stored at v. Computing the list of $\mathsf{top}(\mathcal{H}_v)$ values now is an off-line "extract-maximum" problem. Bern shows how a sequence of $O(m)$ insert, delete and find-max operations on integers between 1 and m can be processed in $O(m)$ time.

The total length of all $\mathsf{top}(\mathcal{H}_v)$ lists is $O(n \log n)$ since each rectangle is stored at $O(\log n)$ nodes. The computation of each list requires time linear in the length of the list. Combining with Theorem 1, we have the following theorem.

Theorem 3 *The rectangles visible from $z = +\infty$ in a set of n rectangles with sides parallel to the $x-$ and $y-$axes can be reported in optimal $O(n \log n)$ time. The space used is $O(n \log n)$.*

7 Conclusions

We have developed a new model of complexity for measuring the performance of hidden-surface elimination algorithms. This model, called the object complexity model, is motivated by the characteristics of graphics rendering hardware like the z-buffer. Our model is appropriate for both dynamic and static hidden-surface elimination. We believe that this model measures the performance of a hidden-surface elimination algorithm much more realistically than the standard computational geometry model of output complexity.

We have also presented a simple, easy-to-implement algorithm under this new model to report the set of rectangles visible from the point $z = +\infty$. All these rectangles are parallel to the xy-plane and have sides parallel to the $x-$ and $y-$axes. This algorithm runs in optimal $O(n \log n)$ time and takes $O(n \log n)$ space.

References

[AgM] P. K. Agarwal and J. Matoušek, "Ray Shooting and Parametric Search," *SIAM J. Comput.* 22 (1993), 794–806.

[Air] J. M. Airey, "Increasing Update Rates in the Building Walkthrough System with Automatic Model-space Subdivision and Potentially Visible Set Calculations ," UNC, Chapelhill, Ph. D. thesis, 1990.

[Ake] K. Akeley, "RealityEngine Graphics," *Proceedings of SIGGRAPH 93* (1993), 109–116.

[Bera] M. de Berg, "Dynamic Output-Sensitive Hidden Surface Removal for *c*-oriented Polyhedra," *Computational Geometry: Theory and Applications* 2 (1992), 119–140.

[Berb] M. de Berg, "Generalized Hidden Surface Removal," *Proceedings of the 9th ACM Symposium on Computational Geometry* (1993), 1–10.

[BHO] M. de Berg, D. Halperin, M. H. Overmars, and J. Snoeyink, "Efficient Ray Shooting and Hidden Surface Removal," *Proceedings of the 7th ACM Symposium on Computational Geometry* (1991), 21–30.

[BeOa] M. de Berg and M. H. Overmars, "Hidden Surface Removal for Axis-Parallel Polyhedra," *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science* (1990), 252–261.

[BeOb] M. de Berg and M. H. Overmars, "Hidden Surface Removal for *c*-oriented Polyhedra," *Computational Geometry: Theory and Applications* 1 (1992), 247–268.

[Ber] M. Bern, "Hidden Surface Removal for Rectangles," *J. Computer and System Sciences* 40 (1990), 49–69.

[Cat] E. Catmull, "A Subdivision Algorithm for Computer Display of Curved Surfaces," Computer Science Department, University of Utah, Salt Lake City, UT, Ph. D. Thesis, Report UTEC-CSc-74-133, December 1974.

[Dev] F. Devai, "Quadratic Bounds for Hidden Line Elimination," *Proceedings of the 2nd ACM Symposium on Computational Geometric* (1986), 269–275.

[DoL] D. Dobkin and R. Lipton, "On the Complexity of Computations under Varying Sets of Primitives," *Journal of Computer and System Sciences* 18 (1979), 86–91.

[FDF] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice*, Addison Wesley, Reading, MA, 1990.

[Goo] M. T. Goodrich, "A Polygonal Approach to Hidden-Line Elimination," *Proc. 25th Allerton Conf. on Comm., Control and Comp.* (1987), 849–858.

[GAO] M. T. Goodrich, M. J. Atallah, and M. H. Overmars, "An Input-Size/Output-Size Trade-Off in the Time Complexity of Rectilinear Hidden Surface Removal," *Proc. 17th Int. Coll. on Automata, Languages and Programming* (1990), 689–702.

[GMV] E. F. Grove, T. M. Murali, and J. S. Vitter, "The Object Complexity Model of Hidden-Surface Elimination," Dept. of Computer Science, Duke University, Technical Report CS-1995-09, April 1995.

[KOS] M. J. Katz, M. H. Overmars, and M. Sharir, "Efficient Hidden Surface Removal for Objects with Small Union Size," *Computational Geometry: Theory and Applications* 2 (1992), 223–234.

[McK] M. McKenna, "Worst-Case Optimal Hidden-Surface Removal," *ACM Trans. on Graphics* 6 (1987), 19–28.

[Meh] K. Mehlhorn, *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, Springer-Verlag, New York–Heidelberg–Berlin, 1984.

[Nur] O. Nurmi, "A Fast Line Sweep Algorithm for Hidden Line Elimination," *BIT* 25 (1985), 466–472.

[PrS] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, New York–Heidelberg–Berlin, 1985.

[PrV] F. P. Preparata and J. S. Vitter, "A Simplified Technique for Hidden-line Elimination in Terrains," *Proceedings of the 1992 Symposium on Theoretical Aspects of Computer Science (LNCS 577)* (February 1992).

[PVYa] F. P. Preparata, J. S. Vitter, and M. Yvinec, "Computation of the Axial View of a set of Isothetic Parallelepipeds," *ACM Trans. Graphics* 3 (1990), 278–300.

[PVYb] F. P. Preparata, J. S. Vitter, and M. Yvinec, "Output-sensitive Generation of the Perspective View of Isothetic Parallelepipeds ," Dept. of Computer Science, Brown University, Tech. Rep. 89-50, December 1989.

[ReS] J. H. Reif and S. Sen, "An Efficient Output-sensitive Hidden-Surface Removal Algorithm and its Parallelization," *Proceedings of the 4th Annual ACM Symp. on Computational Geometry* (June 1988), 193–200.

[Sch] A. Schmitt, "Time and Space Bounds for Hidden Line and Hidden Surface Computation," *Proceedings of Eurographics '81* (1981), 43–56.

[SSS] I. E. Sutherland, R. F. Sproull, and R. A. Schumacker, "A Characterization of Ten Hidden-Surface Algorithms," *Comput. Surveys* 6 (1974), 1–55.

[Tel] S. Teller, "Visibility Computations in Densely Occluded Polyhedral Environments," CS Dept., UC, Berkeley, Ph. D. thesis, 1992.