An O(log N) Parallel Time Exact Hidden-Line Algorithm

F. Dévai

Computer and Automation Institute
Hungarian Academy of Sciences
POB 63, Kende 13-17, Budapest, Hungary, H-1502

Parallel algorithms are given for the exact solution of the hidden-line problem. Most of the parallel algorithms proposed for visibility problems in the literature give approximate solutions, and thus cannot yield an upper bound on the complexity of the particular problem. The first algorithm proposed here is worth mentioning not only for its simplicity, but also from a practical point of view: a speed up of a factor P is achieved by using P processors, $1 \leq P \leq N$, where N is the number of edges used to describe a polygonal scene. Additionally, the problem of aliasing inherent with approximation methods is avoided.

The significance of the second algorithm, which is based on the first one, is mainly on the theoretical level: it is used to establish the parallel complexity of the hidden-line problem. The sequential complexity of this problem has recently been proved to be $\Theta(N^2)$, and now we can prove that in the parallel case the problem is in the complexity class NC, i.e., it can be solved in time polynomial in $\log N$ by using a number of processors polynomial in N, assuming any reasonable model of parallel computation. More particularly, an $O(\log N)$ parallel time solution is given which cannot be further improved even if arbitrarily many processors of a concurrent read, exclusive write parallel RAM model are available.

1. Introduction

Hidden-line algorithms are used in computer graphics for the elimination of invisible parts of edges in line drawings. The input to a hidden-line algorithm can be a set of polygons, and a viewer's position in the three dimensional space. The output of the algorithm is a planar set of line segments corresponding to the parts of edges actually visible from the viewer's position.

The methods of visibility computations seem to divide naturally into two classes: *exact* and *approximation* algorithms. Approximation algorithms, rather than returning exact picture portions corresponding to visible object parts, approximate the image by using an integral number of picture elements, called *pixels*. This classification roughly corresponds to the one given by Sutherland, Sproull and Schumacker [SUTH74] where *image space* algorithms are approximation, and *object space* algorithms are exact algorithms.

Using approximate methods, such as scan-line algorithms, the visibility problem can be divided conveniently among as many processors as are available [KAP79], [HUMC85]. Recently Dèvai [DEV85] has described an architecture based on a scan-line algorithm that determines the visibility of a set of polygons with altogether N edges in time proportional to $(\frac{R}{P})N \log N$ in the worst case, where R is the number of scan lines in the picture, and P is the number of processing elements at the final stage of a parallel processor pipeline architecture.

Using exact algorithms, however, not only the picture is theoretically correct, but also such problems as aliasing [NEWM79], [FOLE82] can be avoided. Moreover, the execution time can be less than the time required by an approximation method used with a high-resolution display device. The exact hidden-surface algorithm given by Kuijk, ten Hagen and Akman has also the desirable features of being incremental and amenable to parallelization [AKMA87], [KUIJ87].

While the complexity of the exact hidden-line [DEV86a] and the exact hidden-surface problems [McKE86] have recently been proved to be $\Theta(N^2)$, most of the algorithms published in the computer graphics literature, e.g., [APP67], [LOUT70], [GAL69], [WEIL77], [FRAN80] take $\Theta(N^3)$ time†) in the worst case [DEV81]. Using the latter algorithms, we cannot achieve better than $O(N^2)$ execution time, even with as many as N processors.

Recently Nurmi [NURM85] has given an $O((N+k)\log N)$ algorithm for the hidden-line, and Dèvai [DEV86b], [DEV86c] for the hidden-surface problem, where k, $k = O(N^2)$, is the number of intersections in the worst case. These algorithms, however, are not easy to decompose for execution on concurrent hardware, and neither are the algorithms given in [DEV86a] and [McKE86]. Instead, we will parallelize the $O(N^2 \log N)$ worst-case time hidden-line algorithm described by Schmitt [SCHM81] and Dèvai [DEV81] independently, and referred to as Schmitt's algorithm throughout this paper.

†) We say that $f(n) = O(g(n))$ if there exist positive constants c and m such that for all $n > m$, $f(n) \leq c\, g(n)$. Similarly, $f(n) = \Omega(g(n))$ if there exist positive constants c and m such that for all $n > m$, $f(n) \geq c\, g(n)$. Finally, $f(n) = \Theta(g(n))$ if there exist positive constants c, d and m such that for all $n > m$, $c\, g(n) \leq f(n) \leq d\, g(n)$.

2. The Hidden-Line Problem

Given a set of pairwise disjoint planar polygonal faces, called a scene, in the three-dimensional space. The faces can be simple polygons possibly with holes, and are projected to a *projection plane*, where each face corresponds to one polygon. We assume that the coordinate system of the three-dimensional space has been choosen so that the projection plane correspond to the plane $z = 0$, and the faces have only positive z-coordinates.

A point (u, v, w) in the three-dimensional space is said to be *visible* if w is greater than the z-coordinate of any point of the scene along the line going through the point (u, v) in the projection plane, parallely with the z-axis.

The above definition of visibility assumes that the viewer's position is on the positive z-axis at infinity. This assumption is usual in computer graphics, as usually a perspective transformation is performed on the scene before the visibility computations.

If no point of an interval of a line is visible, the interval is called a *hidden interval.* The determination of the visibility of the edges in a scene is referred to as as the *hidden-line problem* in computer graphics.

3. Models of Parallel Computation

The widely accepted models of parallel computation are the several variants of the model called *Parallel Random Access Machine* (PRAM). This model consists of random access machines (RAMs) [AHO75], called *processors*, and a *global memory*. All the processors have access to the global memory, and run synchronously. The global memory accesses are assumed to take unit time. The variants of the PRAM model handle concurrent reads and writes to the global memory cells differently. The major variants are the exclusive read, exclusive write (EREW), concurrent read, exclusive write (CREW) and concurrent read, concurrent write (CRCW) models.

In this paper, we use the most widely accepted variant, the CREW PRAM model. In this model, any number of processors can read a given global memory cell at once, but at most one processor is allowed to write into a given memory cell in one step. If more than one attempts to write, the computation is invalid.

The generally accepted definition for the *fast* solvability of a problem by parallel algorithms is if it can be solved in time polynomial in $\log N$ by using a number of processors polynomial in N, where N is called the problem size, and is expressed usually by the number of input data. This class of problems is commonly referred to as NC.

One reason why NC is broadly accepted as the class of problems amenable to parallelization is that this class remains the same whether it is defined in terms of any variant of the PRAM model, or in terms of any other reasonable models, e.g., uniform circuits [BORO77], [RUZZ81]. To convert one model to another, a slow down or speed up of a factor of $\log N$ emerges. The advantage of NC is that it allows us to ignore the factors of $\log N$ that separate the various models.

Our purpose is to decide whether there exist NC solutions for visibility problems, in particular for the hidden-line problem. In Section 2, we assumed that a perspective transformation of the scene is performed before the visibility computations. It is easy to see that a perspective transformation of a scene with N edges can be obtained in $O(\frac{N}{P})$ time by using a CREW PRAM with P processors, $1 \leqslant P \leqslant N$.

4. A Parallel Algorithm with a Small Number of Processors

As the sequential complexity of the hidden-line problem is $\Theta(N^2)$ [DEV86a], the number of processors required to obtain an $O(\log N)$ time solution is $\Omega(\frac{N^2}{\log N})$. A parallel algorithm with a small number of processors, however, may be interesting from a practical point of view. A linear number of processors are considered within reach of current technology [AGGA85], [MEAD80].

Let P be the number of processors, and let each processor be responsible for the visibility of a group of at most $\frac{N}{P}$ edges. For simplifying the presentation, assume that $P = N$. Then each processor executes Schmitt's algorithm on the edge assigned to it, as follows.

The Schmitt Algorithm

(1) [Hidden interval determination] Prepare the list of all (possibly non-disjoint) intervals hidden by other polygons along the edge.

(2) [Interval union] Determine the union of the hidden intervals obtained in step 1. Taking the complement of the resulting intervals with respect to the edge, the visible segments of the edge are obtained.

The intervals of an edge hidden by a given polygon can be determined as follows. Using *line/polygon classification*, a standard geometric algorithm [TIL81], the segments of the projection of the edge inside the projection of the polygon can be obtained in $O(m \log m)$ time for an m-sided polygon. By comparing z-coordinates, we can decide for each line segment whether it is hidden by the appropriate polygon.

Now, let m_i be the number of edges of the i-th polygon, $1 \leqslant i \leqslant M$, then there exists a positive constant c such that step (1) can be executed in time

$$T_1(N) \leqslant c \sum_{i=1}^{M} (m_i \log m_i)$$

Since $\log m_i \leqslant \log N$ for any m_i, and $\sum_{i=1}^{M} m_i = N$, we can write

$$T_1(N) \leqslant c \sum_{i=1}^{M} (m_i \log m_i) \leqslant c \log N \sum_{i=1}^{M} m_i = c\, N \log N$$

In step (1) we cannot obtain more than $\frac{N}{2}$ intervals. The union of these intervals can also be determined in $O(N \log N)$ time [FRED78]. Since each processor completes its work in $O(N \log N)$ time for an edge, we obtain the following result.

Theorem 1. *Given P processors, $1 \leqslant P \leqslant N$, the exact hidden-line problem for a set of pairwise disjoint polygons with a total of N edges can be determined in $O((\frac{N}{P})N \log N)$ time.*

The practical significance of the above algorithm lays in the fact that the reduction of its worst-case time is proportional to the number of processors.

5. An *NC* Solution

Now, let we assign N processors to each edge. Then the algorithm given in Section 4 for determining the visibility of a single edge can be extended as follows.

(1) [Hidden interval determination]

Let E be the given edge, and let s be the straight line containing edge E. We can assume that s coincides with the x axis of the coordinate system.

(1.1) Find the intersection points of the projection of line s with the projections of edges different from E in the projection plane. Using $N-1$ processors, these can be obtained in $O(1)$ parallel time. Assume, for simplifying the presentation, that the intersection points are pairwise disjoint.

(1.2) For each polygon P, sort the intersection points of the projections of the edges of P along the projection of the line s. By using the Ajtai-Komlós-Szemerédi (AKS) parallel sorting algorithm [AJTA83], this can be accomplished in $O(\log n_{\max})$ time, where $n_{\max}$ is the number of edges of the polygon with the maximum number of edges.

(1.3) From the sorted lists obtained in Step (1.2) the hidden intervals of line s can be obtained in $O(1)$ parallel time by taking adjacent pairs of the intersection points of each polygon P, and comparing their z-coordinates with the appropriate z-values of s.

(2) [Interval union]

(2.1) Define the left endpoint of E as a right endpoint of a hidden interval, and the right endpoint of E as a left endpoint of another hidden interval. Then prepare the sorted arrays **L** and **R** of the left and right endpoints of the hidden intervals, respectively. Let k be the number of hidden intervals, $\mathbf{R}[0] = -\infty$, and $\mathbf{R}[k+1] = \infty$. This step takes $O(\log N)$ time by the AKS method.

(2.2) For each left endpoint i, $1 \leqslant i < k$, insert $\mathbf{L}[i]$ in $\mathbf{R}$ by using binary search in $O(\log N)$ serial time, such that

$$\mathbf{R}[j-1] < \mathbf{L}[i] < \mathbf{R}[j].$$

Then the visible segments of edge E can be obtained by executing on each processor i, $1 \leqslant i < k$, the following program:

```
if $i = j$ then
        output ($\mathbf{R}[j-1]$, $\mathbf{L}[i]$) as a visible segment
else
        halt processor $i$;
```

Using at most $\frac{N}{2}$ processors, Step (2.2) can also be executed in $O(\log N)$ parallel time.

Now, we are ready to prove the following result.

Theorem 2. *The exact hidden-line problem for a set of pairwise disjoint polygons with a total of N edges can be solved in* $O(\log N)$ *parallel time, and* $O(N^2)$ *space by using* N^2 *processors.* $\Theta(\log N)$ *time is the best possible under the CREW PRAM model of parallel computation with arbitrarily many processors.*

Sketch of Proof: The above algorithm can be executed in $O(\log N)$ time for a single edge by using N processors. Using N^2 processors, the algorithm can be executed for N edges within the same time.

It follows from the definition of visibility that finding the maximum of N integers is $O(1)$ time reducible to the hidden line problem by using N processors. Cook and Dwork [COOK82] has given $\Omega(\log N)$ lower bound for finding the maximum of N integers allowing infinitely many processors of a CREW PRAM model. From here the theorem follows.

6. Reduction of the Constant Factor

On the theoretical level, we can conclude that the hidden-line problem is easy in the sense that it is solvable in polylogarithmic parallel time by using polynomially many processors. The algorithm given in Section 5, however, has an impractically large constant factor due to the usage of the AKS parallel sorting algorithm.

Recently Cole [COLE86] has given $O(\log N)$ time sort with a small constant factor on a CREW PRAM of N processors. A more complex version of the algorithm for the EREW RAM is also given which also uses N processors and $O(\log N)$ time, where "the constant in the running time is still moderate, though not as small" [COLE86]. It should be noted, however, that the hidden-line algorithm proposed here exploits the power of the CREW model, e.g., in step (1.1).

7. Concluding Remarks

While the algorithm is optimal in a stronger sense, the question that whether $\frac{N^2}{\log N}$ processors are enough to maintain $O(\log N)$ time remains open. (Although the number of visible segments is $\Theta(N^2)$ in the worst case, it might still be possible that each processor reports $\Theta(\log N)$ visible segments.)

Another question of interest is that whether the $O(\log N)$ time can be beaten in a stronger model with write conflict resolution, where $\Omega(\log\log N)$ is a tight bound on finding the maximum of N integers [FICH85].

8. References

[AGGA85] Aggarval, A., Chazelle, B., Guibas, L., O'Dunlaing, C., Yap, C. "Parallel computational geometry." *Proc. 26th Annual Symp. on Foundations of Computer Science*, Portland, Oregon, (Oct., 1985), 468-477.

[AHO75] Aho, A. V., Hopcroft, J. E., Ullman, J. D. "The Design and Analysis of Computer Algorithms". Addison-Wesley, Reading, Mass., 1975.

[AJTA83] Ajtai, M., Komlós, J., Szemerédi, E. "An $O(n\log(n))$ sorting network." *Proc. 15th ACM Symp. on Theory of Computing* (1983), 1-9. Also in: *Combinatorica* **3**,1 (1983) 1-19.

[AKMA87] Akman, V., ten Hagen, P., Kuijk, A. A. M. "A vector-like architecture for raster graphics". (*these proceedings*)

[ANDE85] Anderson, R. "The Complexity of Parallel Algorithms." *Report No. STAN-CS-86-1092.* Department of Computer Science, Stanford University, Stanford, CA, 1985.

[APP67] Appel, A. "The notion of quantitative invisibility and the machine rendering of solids". *Proc. ACM National Conference*, 1967, 387-393.

[BORO77] Borodin, A., "On relating time and space to size and depth". *SIAM J. Computing* **6** (1977) 733-744.

[COOK82] Cook, S., Dwork, C. "Bounds on the time for parallel RAMs to compute simple functions." *Proc. 14th ACM Symp. on Theory of Computing*, San Francisco, Californa, (May, 1982), 231-233.

[DEV81] Dévai, F. "Complexity of Visibility Computations". *Dissertation for the degree of Candidate of Sciences.* Budapest, Hungary, 1981 (In Hungarian).

[DEV85] Dévai, F. "A digital signal processor architecture for real-time image synthesis." *Proc. IEEE Int. Symp. on New Directions in Computing*, Aug. 12-14, 1985, Trondheim, Norway, 371-376.

[DEV86a] Dévai, F. "Quadratic bounds for hidden-line elimination". *Proc. Second Annual ACM Symposium on Computational Geometry*, Yorktown Heights, New York, USA, June 2-4, 1986, 269-275.

[DEV86b] Dévai, F. "Solid modelling in IGOS". *Working Paper GD/15*, Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, Hungary, Oct., 1986, 22 pp.

[DEV86c] Dévai, F. "An intersection-sensitive hidden-surface algorithm." *Proc. Eurographics'87*, Amsterdam, the Netherlands, Aug. 24-28, 1987, 495-502.

[FICH85] Fich, F. E., Meyer auf der Heide, F., Ragde, P., Wigderson, A. " One, two, three ... infinity: Lower bounds for parallel computation." *Proc. 17th ACM Symp. on Theory of Computing*, Providence, Rhode Island, (May, 1985), 48-58.

[FOLE82] Foley, J. D., van Dam, A. "Fundamentals of Interactive Computer Graphics". Addison-Wesley, Reading, Mass., 1982.

[FRAN80] Franklin, W. R. "A linear time exact hidden surface algorithm," *Computer Graphics* **14**, 3 (1980), 117-123.

[FRED78] Fredman, M. L., Weide, B. "On the complexity of computing the measure of $U\ [a_i,\ b_i]$. *Comm. ACM* **21**, 7 (July, 1978), 540-544.

[GAL69] Galimberti, R. Montanari, U. "An algorithm for hidden-line elimination." *Comm. ACM* **12**, 4 (Apr., 1969), 206-211.

[HUMC85] Hu, M. C., Foley, J. D. "Parallel processing approaches to hidden-surface removal in image space." *Comput. & Graphics* **9**, 3 (1985), 303-317.

[KAP79] Kaplan, M., Greenberg, D. P. "Parallel processing techniques for hidden-surface removal." *Computer Graphics* **13**, 2 (Aug.1979), 300-307.

[KUIJ87] Kuijk, A. A. M., ten Hagen, P. J. W., Akman, V. "An exact incremental hidden-surface removal algorithm." (*these proceedings*)

[LOUT70] Loutrel, P. P. "A solution to the hidden-line problem for computer drawn polyhedra." *IEEE Trans. Comp.* **C-19**, 3 (Mar. 1970), 205-213.

[McKE86] McKenna, M. "'Worst-case optimal hidden-surface removal". *Report JHU/EECS-86/05*, The Johns Hopkins University, Baltimore, Maryland, 8 pp. (To appear, ACM Transactions on Graphics)

[MEAD80] Mead, C., Conway, L. "Introduction to VLSI Systems". Addison-Wesley, Reading, Mass., 1980.

[NEWM79] Newman, W. M., Sproull, R. F. "Principles of Interactive Computer Graphics". (Second Edition) McGraw-Hill, 1979.

[NURM85] Nurmi, O. "A fast line-sweep algorithm for hidden line elimination." *BIT* **25**, 3 (1985), 466-472.

[RUZZ81] Ruzzo, W. L., "On uniform circuit complexity." *Journal of Computer and System Sciences* **22** (1981) 365-383.

[SCHM81] Schmitt, A. "Time and space bounds for hidden line and hidden surface algorithms." *Proc. Eurographics'81*, Darmstadt, FRG, (Sep., 1981) 43-56.

[SUTH74] Sutherland, I. E., Sproull, R. F., Schumaker, R. A. "A characterization of ten hidden-surface algorithms." *Computing Surveys* **6**, 1 (March, 1974) 1-55.

[TIL81] Tilove, R. B. "Line/polygon classification: A study of the complexity of geometric computation." *IEEE CG&A* (Apr., 1981) 75-88.

[WEIL77] Weiler, K., Atherton, P. "Hidden surface removal using polygon area sorting". *Computer Graphics* **11**, 2 (Summer, 1977) 214-222.