Scan-Line Methods for Parallel Rendering

Chapter · January 1996

DOI: 10.1007/978-1-4471-1011-8_7

1 author:

Frank L. Devai
London South Bank University
43 PUBLICATIONS **160** CITATIONS

SEE PROFILE

Scan-Line Methods for Parallel Rendering

Frank Dévai
School of Computing and Mathematics, University of Ulster
Londonderry, UK

Abstract

The scan-line approach to parallel rendering requires a smaller number of processors than the processor-per-pixel or the processor-per-object approaches, and also offers linear scalability. A theoretical background for the time requirements of the computational problems involved in the rendering process is given. The concept of deferred rasterization, ie, rasterization of only the visible parts of the polygons is introduced. Then four new scan-line algorithms, including two probabilistic methods, are proposed and compared with five existing ones. A computational complexity analysis is provided in terms of time and space requirements for each algorithm. The scan-line approach is demonstrated to be feasible both for special- and general-purpose parallel architectures.

1 Introduction

The computational requirements of applications like flight and vehicle simulation, virtual reality, computer-aided design and scientific visualisation exceed the performance of a single processor. An approach often proposed in the literature is to assign a processing element to each picture element (pixel) of a raster display, to implement either the z-buffer [12] or the ray-tracing method [16].

The massively parallel Pixel-Planes 4 architecture [12] is a single-instruction, multiple-data-stream (SIMD) array of 512x512 processors implementing the z-buffer algorithm. Pixel-Planes 4 suffers from poor processor utilisation as the processors assigned to pixels not falling into the area of the input polygon are idle. Pixel-Planes 5 [14] attempts to overcome the above difficulty by using multiple 128x128 processor regions that are dynamically reassigned to rasterize the entire screen. An additional requirement is a global routing network to transfer primitives from geometry processors to the appropriate rasterizer. The global routing network imposes a limit on the scalability of the architecture [19].

In the *image-composition* approach taken by the PixelFlow project [19] the global routing network is no longer used: full-screen images of subsets of primitives are calculated, then these images are composed to form the final image. The composition is based on visibility, and implemented by an image-composition network. This approach is expected to offer *linear scalability*, ie, for double the number of processors one can double the speed. There are also some disadvantages, however: a full frame buffer is required for each renderer, and the image-composition network requires a high bandwidth as it must transfer every pixel during every frame.

M. Chen et al. (eds.), *High Performance Computing for Computer Graphics and Visualisation*

An alternative approach is to assign each one of p processors to R/p rows of a raster-scan image of R rows, such that each processor computes a full-screen image of R/p rows. The scan-line approach does not necessarily need special-purpose hardware, but it can be implemented on general-purpose, distributed-memory parallel computers (DMPCs). DMPCs are scalable multiple-instruction, multiple- data-stream (MIMD) computers, and many variants of them are commercially available, such as hypercubes or transputer-based systems.

A potential drawback of the scan-line approach is that the number of processors is limited by the number of scan lines, furthermore the limit of scalability is up to a few hundred processors, which may not be sufficient for the real-time rendering of high-quality scenes. Therefore the reminder of the paper is concerned with the reduction of the computational requirements of rendering.

In Section 2 the idea of *deferred rasterization*, ie, the rasterization of only the visible parts of the input polygons is introduced. Then the main computational problems of rendering are identified as transformations, clipping and visibility computations. The visibility problem is a bottleneck; its worst-case time requirement is proportional to N^2, where N is the total number of edges in the model. In theory the best possible *exact* parallel solution takes time proportional to $\log N$ by using $N^2/\log N$ processors [7, 11].

Scan-line algorithms offer an inherently more efficient *approximate* solution to the visibility problem. In Section 3 four new scan-line algorithms are proposed and compared with five existing ones. A computational complexity analysis is provided in terms of time and space requirements for each algorithm.

In Section 4 implementation by dedicated or commercially available hardware, and experimental performance evaluation of scan-line algorithms are discussed. Section 5 concludes that the computational requirements of rendering can be drastically reduced by deferred rasterization and shading, and by using efficient visibility algorithms, which makes the scan-line approach feasible for both general and special-purpose parallel architectures.

2 Theoretical Background

Rendering three-dimensional objects polygon-mesh models are widely used. The computational problems to be solved are transformations, clipping, visibility computations, rasterization, shading and antialiasing. Realistic rendering can also be achieved by performing only the above-mentioned functions in real time, assuming that interreflection calculations are prepared in advance for a static scene. Shading can be applied only to polygons that are visible—the concept is recently called *deferred shading* in the literature [19]. If one assumes that rasterization of all polygons anyway has to be performed, visibility can be determined by only one extra comparison per pixel by using the z-buffer algorithm. The z-buffer algorithm is easy to implement in hardware, and the speed of that system is difficult to surpass if there are few overlapping surfaces in the scene.

However, eg, in virtual-reality applications, the depth of the scene increases; there are multiple overlapping surfaces. In this case speed can be increased by rasterizing only the part of the polygons that are visible—the concept perhaps could be called *deferred rasterization*. Then the time requirement of both shading and rasterization depends only on the total number of pixels, but in-

dependent of the size of the model.

Antialiasing can be performed by supersampling [13], object-precision antialiasing [4] or by the A-buffer algorithm [3]. The latter two methods are based on visibility computations, and well suited to scan-line algorithms, therefore the three remaining functions to be provided are transformations, clipping and visibility computations.

The computational complexity notation is gradually accepted in computer graphics (eg, [13, p. 717]). If f and g are functions of nonnegative variables $n, m, \ldots$, we say 'f is $O(g)$' if there are positive constants c_1 and c_2 such that

$$f(n, m, \ldots) \leq c_1 g(n, m, \ldots) + c_2$$

for all $n, m, \ldots$. We say 'f is $\Omega(g)$' if g is $O(f)$, and 'f is $\Theta(g)$' if f is both $O(g)$ and $\Omega(g)$.

Then any transformation can be done in $O(N)$ time, where N is the total number of edges in the model, clipping can be performed in $O(N \log N)$ time, however visibility computations need time at least proportional to N^2, ie, $\Omega(N^2)$ time in the worst case [6, 21]. Practical exact (object-space) algorithms take even more than quadratic time, such as $O(N^2 \log N)$ or $O(N^3)$ in the worst case. It has been demonstrated that $O(N^2)$ time is indeed sufficient for the hidden-line problem [6]. McKenna proved the same result for hidden surface removal [18].

For parallel solutions to the visibility problem the parallel random-access machine (PRAM) models of computation have been considered, in particular, the concurrent-read, exclusive-write (CREW) and the exclusive-read, exclusive-write (EREW) models. The parallel complexity of the exact hidden-line problem has been demonstrated to be $\Theta(\log N)$ both under the CREW [7] and under the EREW models [11]. The significance of the above results is mainly on the theoretical level; the best possible running time one can achieve on real machines is $O(\log^d N)$ time, where d is a small positive constant depending on the particular architecture. It follows from the sequential complexity of the visibility problem that $N^2/\log^d N$ processors are necessary to achieve $O(\log^d N)$ time. To reduce the processor number approximation methods such the z-buffer or scan-line algorithms can be used.

3 Scan-Line Algorithms

With scan-line methods a narrow horizontal rectangle of the exact image is approximated by the visible parts of line segments obtained as the intersection of the input polygons with a plane perpendicular to the viewing plane. Scan-line methods consist of two phases. In the first phase all polygons are intersected with the scan-line planes perpendicular to the viewing plane, then in the second phase the visibility of the line segments obtained in the first phase are determined.

If N is the total number of edges in the scene, the first phase produces $O(N)$ non-intersecting line segments for each scan line, and takes $O(N)$ time per scan line, assuming that each polygon has at most a constant number of edges, eg, if the scene is modelled by triangles. Even if the scene is modelled by a set of arbitrarily simple polygons possibly with holes, $O(N)$ time is sufficient, assuming that an $O(N \log N)$ time preprocessing is allowed [9].

Determining the visibility of a set of N non-intersecting line segments in the plane takes $\Omega(N \log N)$ time in the worst case [5]. The methods offered by textbooks [13, 20] are *Watkins' method* and the scan-line variant of the *z-buffer* algorithm. Early scan-line algorithms [2, 22] attempt to exploit coherence [13] which is no longer possible in a parallel environment. Watkins' algorithm takes $O(N^2)$ [17] and the z-buffer $\Theta(NR)$ time in the worst case, R being the number of pixels in the scan line.

Two other scan-line algorithms are the *priority-queue* [5] and the *merge* [1] methods. Both take $O(N \log N)$ time in the worst case, ie, these methods are best possible in terms of worst-case time, but use sorting, and merging of visible sets of line segments, therefore less appropriate for hardware implementation. In the remainder of this section we offer four simple but efficient scan-line algorithms. First two hierarchical methods based on subdivision techniques, then two simple probabilistic algorithms are proposed.

3.1 Subdivision Techniques

A scan-line method can be developed as the two-dimensional variant of Warnock's algorithm for the visibility of a set of polygons in three-dimensional space [13, 20]. Let S be a set of line segments, W a window and b a background line segment. Initially W represents the scan line consisting of R pixels.

The set S of line segments is within the trapezoid formed by the scan-line, the background segment b and two lines parallel to the z-axis at the left and right endpoints of the scan line. Though the window is defined as a set of contiguous pixels, sometimes it is convenient to refer to that trapezoid as the window.

If the x-coordinate of the left endpoint of a line segment t is less than or equal to the x-coordinate of the left-hand side of the window, and at the same time the x-coordinate of the right endpoint of t is greater than or equal to the x-coordinate of the right-hand side of the window, we say that t totally overlaps the window. The *scan-line Warnock method* is formally stated as Figure 1.

With reference to the notation of Figure 1, it should be noted that S_1 will always be empty when window W is equal to a pixel, as all segments will overlap W, and will be put on list B. Therefore the termination of the algorithm can be assured by testing if S_1 is empty.

The recursive subdivision of the scan line can be represented by a binary tree of R leaf nodes. The binary tree has $R - 1$ internal nodes, then the total number of subdivisions is $2R - 1$. There are $O(N)$ operations in any subdivision, therefore an upper bound on the worst-case time of the scan-line Warnock method is $O(NR)$. The same $O(NR)$ upper bound can be given on its space requirement, based on similar arguments. The actual time and space requirements are better, since not all line segments are processed at every subdivision.

The scan-line Warnock method does not subdivide line segments, only the scan line, therefore each line segment should be tested against another sub-window, even if the line segment is already outside that sub-window. The extra tests are avoided by the second method we propose, called the *subdivision method*, which is similar to the scan-line Warnock method, except the line segments are always subdivided at the window boundaries. The subdivision method is presented as Figure 2.

```
Warnock(S, W, b)
    initialise B and S1 as empty lists of line segments;
    for each segment t in S do
        if t is in window W then
            if t totally overlaps W then
                relocate t from S to list B
            else
                relocate t from S to list S1
            endif
        endif
    endfor;
    if B is not empty then
        find a on list B as the segment nearest to the observer;
        for each segment t in S do
            if t is behind a then remove t from S endif
        endfor;
        substitute a for b
    endif;
    discard set S and list B;
    if S1 is empty then
        display b in W;
        return
    else
        subdivide W into two sub-windows W1 and W2 of equal size;
        subdivide b accordingly into b1 and b2;
        Warnock(S1, W1, b1);
        Warnock(S1, W2, b2);
        return
    endif
end
```

Figure 1: The two-dimensional variant of Warnock's method

With the subdivision method each line segment is tested in, and divided into at most $2 \log_2 R$ sub-windows, and there exist scenes where each segment is divided into $\Theta(\log R)$ parts, therefore the worst-case time requirement is $\Theta(N \log R)$. For similar considerations the worst-case space requirement is also $\Theta(N \log R)$. The slightly more sophisticated *z-tree method* [8] reduces the space requirement to $\Theta(R)$ while retaining the $\Theta(N \log R)$ worst-case time bound.

3.2 Probabilistic Algorithms

Finally we consider two probabilistic algorithms that may exhibit different running times if applied twice for the same input. The advantage of probabilistic algorithms is simplicity and good expected running time. The first method, called *random*, is stated as follows. Choose a line segment t at random. Discard all line segments and all parts of line segments hidden by t. Let S_1, S_2 and

```
subdiv(S, W, b)
    initialise B as an empty list of line segments;
    for each segment t in S do
        if t totally overlaps W then
            relocate t from S to list B
    endfor;
    if B is not empty then
        find a on list B as the segment nearest to the observer;
        for each segment t in S do
            if t is behind a then remove t from S endif
        endfor;
        substitute a for b, and discard list B
    endif;
    if S is empty then
        display b in W;
        return
    else
        subdivide W into two sub-windows W1 and W2 of equal size;
        subdivide b accordingly into b1 and b2;
        subdivide S into S1 and S2 such that S1 in W1 and S2 in W2;
        subdiv(S1, W1, b1);
        subdiv(S2, W2, b2);
        return
    endif
end
```

Figure 2: The subdivision method

```
trapezoid(S, W, b)
    if S is empty then display b in W; return
    else
        for min(k, |S|) line segments chosen at random from S do
            find segment a with the maximum-area trapezoid formed
            by a, b and the vertical lines through the endpoints of a;
            remove a from S
        endfor;
        subdivide W into sub-windows W1, W2 and W3, and b into b1, b2
        and b3 at the endpoints of a such that W1 and b1 are to the left of a,
        W2 and b2 are to right of a, and a and b3 are in W3;
        for each segment t in S do
            if t is to the left of a then put t into S1
            else if t is to the right of a then put t into S2 endif
            else
                if t intersects the trapezoid of W1 and/or W2 then
                    subdivide t and put its appropriate
                    part(s) (if any) into S1 or S2;
                    rename the remainder as t
                endif;
                if t is in front of a then put t into S3 endif
            endif
        endfor;
        trapezoid(S1, W1, b1);
        trapezoid(S2, W2, b2);
        trapezoid(S3, W3, a)
    endif
end
```

Figure 3: The trapezoid method

S_3 be the set of line segments left and right to t and in front of t respectively. Use the same procedure recursively for S_1, S_2 and S_3.

The *trapezoid method* examines k randomly selected line segments, where k is a constant, and chooses the segment with the largest area behind it to subdivide S. Let S_1, S_2 and S_3 be as above, and continue as with the random method. The trapezoid method is stated formally as Figure 3. The best value of k can be determined experimentally.

Both the random and the trapezoid method could be classified as a Las-Vegas algorithm. This type of algorithms never give a wrong result, but their running time is not always guaranteed, though usually very good on the average. For establishing an upper bound on the worst-case running time of the random and the trapezoid methods, consider a set S of N line segments with non-overlapping images. If always the leftmost or the rightmost line segment is selected for partitioning S into subsets, the running time is $\Omega(N^2)$. On the other hand, both methods compare at most N line segments in at most $2N-1$ vertical strips, which takes $O(N^2)$ time, therefore the worst-case time requirement is

method	time	space
Watkins	$O(N^2)$	$O(N)$
z-buffer	$\Theta(NR)$	$\Theta(N+R)$
priority-queue	$\Theta(N \log N)$	$\Theta(N)$
merge	$\Theta(N \log N)$	$\Theta(N)$
Warnock	$O(NR)$	$O(NR)$
recursive subdivision	$\Theta(N \log R)$	$\Theta(N \log R)$
z-tree	$\Theta(N \log R)$	$\Theta(R)$
random	$\Theta(N^2)$	$\Theta(N^2)$
trapezoid	$\Theta(N^2)$	$\Theta(N^2)$

Table 1: Worst-case time and space requirements of scan-line algorithms

$\Theta(N^2)$.

There can be at least $N/2$ line segments broken into at least $N/2$ parts in the worst case, but there are at most N parts in any of the $2N-1$ vertical strips. This gives a $\Theta(N^2)$ bound on the space requirement in the worst case.

Table 1 summarises the scan-line methods investigated, together with the time and space requirements of the particular method, where N is the number of line segments and R is the number of pixels in the scan line.

4 Implementation and Performance Evaluation

Let $T(N,R)$ be the running time of a scan-line algorithm, let the number of scan lines be the same as the number of pixels in a scan line, ie, R, and let $p, 1 < p < R$, be the number of processors. Then visibility computations take $RT(N,R)/p$ time, where N is the total number of edges of the model.

Assuming $R = 1024$, and considering $T(N,R)$ functions provided by Table 1, the z-tree and the subdivision methods can determine visibility two orders of magnitude faster than the z-buffer algorithm, assuming the constant factors are the same. The subdivision method does not use sorting or advanced data structures, therefore can easily be implemented in hardware, which results in small constant factors.

Molnar, Eyles and Poulton [19] estimate that rasterization (and visibility) of a scene of 100,000 polygons updated at 30 Hz requires 750 million integer operations per second. On the other hand, eg, the iPSC/860 hypercube with 64 processors, each with 33 MIPS performance, can update a 1024-line screen such that each processor computes 16 (interleaved) scan lines. The total performance is more than 2000 million integer operations, which matches the estimate by Molnar et al, even if the reduced computational requirements due to deferred rasterization and more efficient visibility algorithms is not taken into consideration. This arrangement is also in the range of linear scalability, ie, double (or halve) the number of processors the load remains balanced. Therefore the scan-line approach is feasible for both special- and general-purpose parallel architectures.

We have provided the asymptotic worst-case time and space requirements of nine scan-line algorithms. Asymptotic analysis, however, cannot take into consideration the constant factors of the algorithms. The constant factors can also be different in different environments. Therefore a portable testbed has been developed that makes possible the comparative evaluation of the actual performance of the algorithms on the particular hardware platform used.

As some algorithms exploit the fact that the input produced by a solid modeller results in a set of non-intersecting line segments, a test-data generation method is required to produce a set of non-intersecting random line segments in the plane. It would be a naive approach to divide an R by R square into M rows and M columns, where M is the square root of R, and then choose the two endpoints of a line segment uniformly, independently, at random from each square of the subdivision. This results in a very sparse scene, ie, relatively short line segments with a lot of empty space among them.

The method proposed in a companion paper [10] produces scenes of higher density, though the minimum enclosing rectilinear rectangles of the line segments will still be pairwise disjoint. The technique is based on a channel-assignment algorithm [15], and takes $4N$ random numbers and $O(N \log N)$ time in the worst case to generate $4N$ coordinates for a set of N non-intersecting random line segments.

5 Conclusions

Scan-line methods for parallel rendering require a smaller number of processors than the processor-per-pixel or the processor-per-object approach. The scan-line approach also offers linear scalability. The main contributions of this research are as follows. The rasterization of only the visible parts of the polygons is recommended. Then four new simple but efficient scan-line algorithms are proposed, and compared with five existing ones. Hierarchic techniques, particularly the subdivision method, are appropriate for hardware implementation, and all nine methods can be implemented on commercially available MIMD parallel machines. The computational requirements of rendering is drastically reduced by shading and rasterizing only the visible parts of polygons, and by using efficient visibility algorithms, which makes the scan-line approach feasible either on general- or special-purpose hardware.

References

[1] Atallah, M. J., Cole R., Goodrich M. T. Cascading divide-and-conquer — a technique for designing parallel algorithms. *SIAM Journal on Computing* **18**,3 (1989) 499–532.

[2] Bouknight, W. J. A procedure for generation of three-dimensional halftoned computer graphics presentations. *Comm. ACM* **13**,9 (Sep. 1970) 527–536.

[3] Carpenter, L. The A-buffer, an antialiased hidden surface method. *Computer Graphics* **18**,3 (Proc. Siggraph'84, July 1984) 103–108.

[4] Catmull, E. A hidden-surface algorithm with anti-aliasing. *Computer Graphics* **12**,3 (Proc. Siggraph'78, Aug. 1978) 6–11.

[5] Dévai, F. Complexity of two-dimensional visibility computations. Proc. *3rd European Conference on CAD/CAM and Computer Graphics*, Paris, France, Feb. 1984, MICAD'84 Vol. **3**, 827–841.

[6] 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.

[7] Dévai, F. An O(log N) parallel time exact hidden-line algorithm. In: Kuijk, A. A. M., Strasser, W. (Eds) *Advances in Graphics Hardware II*, Springer-Verlag, Berlin, Germany, 1988, 65–73.

[8] Dévai, F. Approximation algorithms for high-resolution display. Proc. *PIXIM'88, 1st International Conference on Computer Graphics in Paris*, Péroche, B. (Ed) France, Oct. 24–28, 1988, 121–130.

[9] Dévai, F. On the complexity of some geometric intersection problems. *Journal of Computing and Information* **1**,1 (May 1995) 333–352.

[10] Dévai, F. Test-data generation for the performance evaluation of parallel hidden-surface techniques. (Submitted for publication).

[11] Dévai, F. Optimal parallel algorithms for interval union and hidden-line elimination. (In preparation).

[12] Eyles, J., Austin, J., Fuchs, H., Greer, T., Poulton, J. Pixel-Planes 4: A Summary. In: Kuijk, A. A. M., Strasser, W. (Eds) *Advances in Graphics Hardware II*, Springer-Verlag, Berlin, Germany, 1988, 183–207.

[13] Foley, J. D., van Dam, A., Feiner, S. K., Hughes, J. F. *Computer Graphics: Principles and Practice.* (Second Edition) Addison-Wesley, Reading, Mass., 1990. 1174 pp.

[14] Fuchs, H. *et al.* Pixel-Planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. *Computer Graphics* **23**,3 (Proc. Siggraph'89) 79–88.

[15] Gupta, U. I., Lee, D. T., Leung, J. Y.-T. An optimal solution for the channel-assignment problem. *IEEE Trans. Comput.* **C-28**,11 (1979) 807–810.

[16] Kedem, G., Ellis, J. L. The raycasting machine. Proc. *1984 Int. Conf. on Computer Design*, Oct. 1984, 533–538.

[17] Kremer-Patard, G. Evaluation d'algorithmes de calcul de la visibilité d'un ensemble de segments du plan. *Revue de CFAO et d'Infographie* **3**,3 (1988) 39–57.

[18] McKenna, M. Worst-case optimal hidden-surface removal. *ACM Transactions on Graphics* **6**,1 (Jan. 1987) 19–28.

[19] Molnar, S., Eyles, J., and Poulton, J. PixelFlow: High-speed rendering using image composition. *Computer Graphics* **26**,2 (Proc. Siggraph'92, July 1992) 231–240.

[20] Newman, W. M., Sproull, R. F. *Principles of Interactive Computer Graphics.* (Second Edition) McGraw-Hill Kogakusha Ltd, Tokyo, Japan, 1979, 541 pp.

[21] Schmitt, A. Time and space bounds for hidden line and hidden surface algorithms. Proc. *Eurographics'81*, Darmstadt, Germany, (Sep. 1981) 43–56.

[22] Wylie, C., Romney, G. W., Evans, D. C., Erdahl, A. C. Halftone perspective drawings by computer. Proc. *Fall Joint Computer Conference 1967*, Thompson Books, Washington DC, 1967, 49–58.