A Distributed Architecture for the Visualization of Geometric Models

Sourav Dalal* Frank Dévai* Md Mizanur Rahman*

The geometric aspects of the visualisation of large data sets, in particular, digital models of real or planned solid objects in a heterogeneous distributed environment are investigated. One of the fundamental problems of rendering is visibility determination; the process of deciding what parts of the surface of the model can be seen from a possibly moving point. Any algorithm for determining the visibility of a set of polygons in three-dimensional space with a total of N edges takes $\Theta(N^2)$ time in the worst case. This growth rate is a serious difficulty. To overcome it hardware accelerators are used in low-end systems, and parallel algorithms are proposed for high-performance graphics systems. For example, the hidden-line problem can be solved in $\Theta(\log N)$ time either on N^2 EREW or on $N^2/\log N$ CREW PRAM processors, and the hidden-surface problem in the same $\Theta(\log N)$ time on N^2 CREW PRAM processors. The $\Theta(\log N)$ result cannot be further improved even if arbitrarily many processors were available [2, 3]. Algorithms with a smaller number of processors, called *coarse-grain algorithms*, can also be derived from these results, e.g., a hidden-surface algorithm that takes $O(\frac{N^2 \log N}{p} + \log N)$ time in the worst case by using p CREW PRAM processors [3].

Unfortunately, parallel architectures either suffer from the problem of latency or do not scale well, hence are expensive. Using distributed shared memory (DSM) running on a network of workstations [1] seems to be a better alternative. There are some drawbacks, however, e.g., overhead: if a processor needs to access a variable available only on a remote machine, the whole page must be transferred. Experts like Tanenbaum and van Steen [5] are sceptical: *"After almost 15 years of research on distributed shared memory, DSM researchers are still struggling to combine efficiency and programmability. To attain high performance on large-scale multicomputers, programmers resort to message passing despite its higher complexity compared to programming (virtual) shared memory systems. It seems therefore justified to conclude that DSM for high-performance parallel programming cannot fulfil its initial expectations."*

The objective of this research is the design and implementation of a light-weight, inexpensive message-passing architecture for the visualisation of large geometric models. This system also uses networks of workstations, typically available in a university environment, but unlike DSM, it is a more efficient approach using scanline algorithms. A scanline is a row of pixels of the image. The pixels of a scanline can be obtained by determining the visibility of the objects in the plane perpendicular to the screen and containing the scanline. The intersection of a polygon-mesh model (i.e., a set of polygons) and a plane is a set of line segments in the plane.

Our system is based on a *client-server architecture*, where a number of servers support typically one client. Each server is responsible for the calculation of a couple of scanlines, and the delivery of the results back to the client which displays the image.

The operation of the system is outlined as follows. The user, sitting in front of a graphics workstation running the client software, selects a geometric model or virtual environment they want to visualize. Once the geometric model has been selected, the client software distributes the corresponding data set on a high-speed network by using *scalable reliable multicast.* All workstations having the server software installed pick up the model, and store it on their local disk. On user interaction, e.g., zoom, translation or rotation of the model, the client broadcasts a 4x4 homogeneous transformation matrix. All the available servers receiving this matrix perform the required transformation on their stored model.

Each server is associated with a unique name, which is an integer in the range $[0, m-1]$, where m is the total number of the available servers at any given point of time. In reply to a client request, each server calculates a number of scanlines. Scanlines are numbered from 0 to $h-1$, where h is the height of the image, i.e., the total number of scanlines on the screen of the client's workstation. Server k calculates scanlines $i = k + jm$, for $j = 0, 1, 2, \ldots$, such that $i < h$, and sends them back to the client.

*London South Bank University, London, 103 Borough Road, London, SE1 0AA, UK, email: fl.devai@lsbu.ac.uk

The client then assembles the scanlines, and displays the next frame. It also calculates a new transformation matrix from user input, gathers server statistics, rename servers if necessary and informs each server involved. Then it goes back to the beginning of the loop broadcasting a new transformation matrix and requesting scanlines.

In the unlikely case of the output of a single server is lost, the scanlines from the previous frame are used. Often there is no difference between the same scanlines of subsequent frames (e.g., when a designer is contemplating a part of the model) and the difference is hardly noticeable on moving images. (Note that the scanlines are delivered by the servers in an interleaved pattern: a scanline from server k is followed by one from server $k+1$, and so on.)

If, however, the client notices from server statistics that the output of a server is consistently lost in the last few frames, the client renames the servers: If, say, server k is not responding, server $m-1$ is renamed as k, and m is reduced by 1. If a server re-appears, it is given the name m, and m is incremented by 1.

From the above it follows that the system is *fault tolerant* in the sense that it tolerates both lost messages and server failures. The latter is particularly important, because in this way the workstations do not have to be dedicated to the system. The servers run on workstations as background processes; when a workstation gets idle, the server joins the system, and it leaves the system when preempted by an interactive process. Thus, the hardware cost of the system is negligible, as organizations like universities or design offices already have extensive workstation networks.

Another significant advantage of the proposed architecture is that the servers only need to compute a planar visibility problem with a complexity of $\Theta(n \log n)$ rather than the 3D problem of complexity $\Theta(N^2)$, where $n \leq N$ is the number of line segments in the plane of the scanline. The planar visibility problem is well understood, and this research gave us the opportunity to implement 10 scanline algorithms, and analyse their asymptotic resource requirements.

Asymptotic analysis, however, cannot take into consideration constant factors, which can be different in different environments. Therefore a *portable testbed* was developed for the comparative evaluation of the actual performance of the algorithms on the particular hardware-software platform they are used. In this way in a heterogeneous distributed system the fastest version of the server can be installed in any given machine environment.

Considering the number of possible algorithms together with their variants, the number of time measurements required for conclusive results is substantial, hence using real 3D models would be too expensive and time consuming. Since the input to a scan-line algorithm is only a planar set of line segments, a more efficient test-data generation method based on random line segments was developed.

As some scanline algorithms exploit the fact that the input obtained from real solid models results in a set of line segments that are non-intersecting (except at their endpoints) a test-data generation method was required to produce a planar set of non-intersecting random line segments. Our method takes a total of $4n$ 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 in the plane.

The Java language was used for both the development of a prototype distributed system and the testbed. Since the running time of scanline algorithms is very short, accurate time-measurement techniques are required. These were implemented by reading the time-stamp counters of the processors using Java native methods.

Though the planar visibility problem is well researched, we encountered some interesting problems which, we believe, are unsolved. For example, one of our algorithms, called the *priority-queue method*, gave better than expected experimental results. This algorithm uses a heap to maintain an order on line segments. It is known that n elements can be inserted in a heap in $O(n)$ expected time, but deleting the minimum element takes $\Theta(\log n)$ time on average [4]. Our algorithm, however, deletes *arbitrary elements*, which are near to leaf nodes most of the time, thus repairing the heap costs little. So far, however, we could not turn our arguments into a formal proof.

References

[1] Amza, C. *et al.* TreadMarks: Shared memory computing on networks of workstations. *IEEE Computer*, 29(2):18–28, February 1996.

[2] Dévai, F. On the computational requirements of virtual reality systems. *State of the Art Reports*, EUROGRAPHICS'97, Budapest, 1997, 59–92.

[3] Dévai, F. Parallel algorithms for visibility computations. Proc. EUROGRAPHICS *UK Chapter 17th Annual Conference*, Cambridge, UK, 1999.

[4] Schaffer, R. and Sedgewick, R. The analysis of heapsort. *J. Algorithms*, 15(1):76–100, 1993.

[5] Tanenbaum, A. S. and van Steen, M. *Distributed Systems—Principles and Paradigms.* Prentice Hall, NJ, 2002 (p 32)