DESIGN AND ANALYSIS OF A PARALLEL RAY TRACING COMPUTER

John G. Cleary, Brian Wyvill, Reddy Vatti, Graham M. Birtwistle,

Department of Computer Science, The University of Calgary
2500 University Drive NW, Calgary, Canada T2N 1N4

ABSTRACT

An MIMD parallel computing system is described. Each processor is connected to four or six neighbouring processors to form a square or cubic array. There is no global bus of any kind connected to all the processors, this has the advantage that the system can more easily be scaled up in size. It is proposed that this system be used for doing parallel ray tracing of three dimensional scenes to yield projected view. The software required to do this is described. The performance of the system is analyzed. While the time taken will depend strongly on the scene and the presence of features such as specular reflection it is possible to derive expresssions showing how this varies with the number of processors and the number of rays to be generated. These analytic results are supported by simulations of ray tracing on a number of scenes with polygonal surfaces. Some comments are made on the problems involved in construction of such a system and a square array of processors is constrasted with a cubic array.

KEYWORDS: graphics, ray tracing, parallel processing, multiple micro-processors, animation.

Summary

A well known technique for creating two dimensional representations of three dimensional scenes is ray tracing. This technique projects one or more rays from the "eye" of the observer outward through each pixel. Each ray is followed until it intercepts a surface in the three dimensional scene. The intensity of the illumination at this surface is then added to the pixel corresponding to the ray. If the surface has specular reflection or is transparent then further reflected or refracted rays may be spawned.

Ray tracing has a number of advantages over other hidden surface methods. If more than one ray is used per pixel it automatically anti-aliases the image. It provides very exact shading for curved surfaces. Reflective and transparent surfaces can be easily handled. Thus this method is excellent for producing high quality realistic images.

Its main disadvantage is the very large amount of computing that is necessary. In this paper we propose a two or three dimensional array of micro-processors to do ray tracing calculations. The array is characterized by the following properties. It is a multiple instruction multiple data (MIMD) system with each processor running asynchronously. Each processor is connected to at most six nearest neighbours, that is there is no global bus for communication. The only global signals are a reset line and possible some diagnostic probes. Communication between neighbours uses shared memories and a simple queueing system for message passing.

There are three main components to the software for such a computer. The ray-tracing algorithm itself, the set-up of a picture description and operating overhead for inter-process communication and other tasks such as loading executable code and monitoring and diagnosis. The first two of these are briefly described below.

During the ray-tracing operation the three dimensional scene is divided into rectangular volumes, which are assigned to one processor each. Each processor stores information on those parts of surfaces which pass through its own volume. Each ray is represented by one packet of about 25 bytes specifying its direction and other information. When a processor receives a ray packet it checks to see if it will intercept any of the surfaces within its own cube. In the simplest (and fastest) case there will be no intersection. Then the ray packet is handed to the processor owning the next cube of space which the ray will pass through. (This will of course be one of the six neighbouring processors). If the ray does intersect a surface then a new ray is sent in the direction of any specular reflection and of any refraction (if the surface is transparent). Finally the intensity of the diffuse reflection back along the original ray is computed. This information is encoded in a "return" packet which is passed from processor to processor back to where the ray started. A ray which reaches the edges of the three dimensional scene is checked to see if it is directed toward a light source. If it is, a return packet is sent, if it is not the ray is forgotten.

It is of course necessary to initially set-up the description of the picture in the processors. This will be done by sending the descriptions of the individual surfaces serially from a host computer to one or more of the individual processors (the 8 corner processors might be good candidates). The descriptions of the surfaces will then be passed from processor to processor. Those processors which intersect (part of) the surface will retain a description of that (part of) the surface and hand the description onto their neighbours. For animated pictures it should be possible to set-up frames other than the first one by performing local increments to the positions of surfaces. A similar techniques is envisaged for initially loading code into the processors.

When analyzing the performance of the ray tracing it can be shown that the total time to complete a ray trace will contain three components: A constant overhead to account for creation and termination of individual rays; a term which grows linearly with the length of the edges of the array that is with the square or cube root of the number of processors); and finally a term which is inversely proportional to the square of the lengths of the edges and proportional to the number of rays to be generated. As a results of this there is a minimum possible processing time for a particular scene. Any increase in the number of processors beyond this will cause the system to slow down.

While the systems performance will depend strongly on the scene being processed it is still possible to make order of magnitude estimates of price and performance of a processor array. If we assume that a processor of about the power of an Intel 8086 could be placed on a board with memory and interface logic for say $200 then a 1000 processor system would cost $20,000. Conservative assumptions about execution speeds of the algorithms show that it could construct a 1000 by 1000 pixel picture (one ray per pixel) in 5 secs. and 4000 by 4000 in 80 secs. A 125,000 processor system (50 by 50 by 50) would cost $50,000,000. It could process a 1000 by 1000 pixels in 0.15 secs. and 4000 by 4000 in 2 secs. It is notable that 0.15 secs. is sufficiently small to make real time production feasible. Certainly it would be good enough to preview in degraded form pictures to be produced later in greater detail.

Ray tracing is a perfect candidate for the application of parallel micro-processor systems: the calculations for the individual rays are independent of one another; and the actual physical space in which the calculations take place can be mapped directly onto a configuration of processors with only local interconnections. We are currently proceeding with the development of software to simulate the system as part of the distributed software prototyping project (JADE) at the University of Calgary. Simultaneously we are designing and building one or two prototype processors to test ray-tracing code and inter-processor communication. The next step will be to build a system with about 40 processors which should be sufficient to test almost all aspects of the software.