RAY TRACING ON THE MPP

John E. Dorband
Image Analysis Facility/635
NASA/Goddard Space Flight Center
Greenbelt, MD 20771

ABSTRACT

Generating graphics to faithfully represent information can be a computationally intensive task. We will present a way of using the MPP to generate images by ray tracing. This technique uses sort computation, a method of performing generalized routing interspersed with computation on a single-instruction-multiple-data (SIMD) computer.

Keywords: Ray Tracing, Graphic Generation, Sort Computation, SIMD, MPP.

INTRODUCTION

New ways of representing data must be developed so that researchers can most effectively focus their time and effort in exploring the ever increasing volume and complexity of data that they acquire and generate.

Since computers are not yet able to discover knowledge for which no description has been given, we must rely on the cognitive abilities of the researcher to recognize undiscovered characteristics of the data. Tools are needed that extend the senses of the researcher deeper into the data he is studying. Just as a telescope allows an astronomer to visually search deep into the universe, so a computer allows a researcher to visually study data in regions that could not be explored through mere imagination.

Computer-assisted perception is one of the most valuable aspects of man-machine interaction, but the amount of information that needs to be transferred between man and machine for human perception to be effectively extended is huge. The visual interface between man and machine has the greatest potential for satisfying this requirement, but a fast and efficient means of converting logical information into visual information needs to be developed. Faster and less expensive computer hardware, along with more efficient algorithms, are needed to place this capability within the reach of most researchers.

The most versatile and realistic means of generating images of objects in three-dimensional space is ray tracing. Ray tracing generates images by simulating the movement of light rays in the three-dimensional space containing the objects to be displayed. Ray tracing is, however, computationally intensive making it both slow and expensive when the three-dimensional space holds many complex objects and especially when animated sequences composed of many images must be generated.

An experimental computer architecture termed single-instruction-multiple-data (SIMD) is being investigated at NASA's Goddard Space Flight Center. SIMD architectures show promise of delivering enormous computational power at less cost than other existing architectures. The Goddard prototype, the Massively Parallel Processor (MPP), has a computational element consisting of a 128 by 128 array of small computers.

It had been recognized that ray tracing, which requires the use of irregular data arrangements, could not be performed easily on the SIMD architecture. This has changed with the recent development of a technique called sort computation (Ref. 1), which uses the regular SIMD computer architecture to process irregular data arrangements more efficiently than was previously thought possible. With sort computation, the MPP can be used to start developing ray tracing as a method of rendering images of objects placed in three-dimensional space.

Sort computation is standard sorting with a twist provided by an enhancement to the usual comparison step in the sort. The enhancement performs computation during the sort. The kinds of computation that can be performed include aggregation and distribution type operations on sets of data records whose key values are the same. The routing mechanism of the sort guarantees that the right data will be brought together at the right time

to accomplish the computation without the programmer's a priori knowledge of where the data was before the computation or where it will be after the computation. This makes it easy to process irregularly arranged data.

Our initial step toward implementing ray tracing on the MPP was the development of a utility that allows a researcher to view a topographical map in three-dimensional space from any perspective. The map is a 512 by 512 grid of elevation points. These points are initially plotted in three-dimensional space. Then the intersection of the viewing screen with light rays from the view point to each point in the map is calculated. It is then known where all points will be seen on the viewing screen. A sort computation operation, sort minimum, is then used to determine which points are not hidden by other points. Sort minimum finds the closest point on the elevation map to the view point for each position in the viewing screen. Finally, the brightness values of those points that are not hidden are copied to their proper position in the viewing screen, using sort distribution.

This utility, run on the MPP, takes 3.5 seconds to generate an image of 262,144 elevation points. It was implemented on the MPP by a Code 635 summer student. Jennifer Trainor (see Figure 1 , Color Plate IV), and is being used to view elevation maps generated from Shuttle Imaging Radar-B (SIR-B) images by Dr. James Strong (Code 636).

SORT COMPUTATION

To truly understand how ray tracing is accomplished on the MPP one must understand sort computation. Computation in general requires both the ability to manipulate elements of data based on the values of other elements of data and the ability to route these elements of data to places where they can affect each other. Sort computation uses sorting as a routing mechanism to support interspersed routing and data manipulation. Sort aggregation and sort distribution are the two basic forms of sort computation. Sort computation is performed on sets of records, grouped according to a key contained in each record. Groups of records contain only records that have been determined by some function to be equal.

Sort aggregation generates an accumulative result for each group of records and places this result in one of the records. Usually it is placed in the first record or the record with the smallest key value. The accumulation operation can be any operation that is both associative and commutative. That is any operation for which the result is not dependent on the order of the operand or the order in which the operations will be performed. Addition, multiplication, and, or, and exclusive-or are examples of valid operations.

Sort distribution copies the value of a field of a specific record in a group into that field in every record of that group. The record that contains the value to be distributed contains a flag that is set to true, indicating that this record contains the value that is to be copied into all other records of that group. Note that there may be more than one record in a group for which the flag is set, as long as they contain the same value in the field that is to be distributed.

Sort computation is really quite simple. First, we must view a sorting algorithm as having two parts, the comparison of records and the routing of records. The comparison of records results in a determination of which of two records being compared is larger. The routing of records takes the result of the comparison and determines where each of the records are to go for their next comparison. Thus, the sort can be viewed as a routing routine and a comparison routine, where the routing routine calls the comparison routine when necessary. All sorts consist of these two parts. Sort computation can use the routing part of any sort algorithm. The comparison routine is replaced depending on what type of computation is to be performed.

The routing routine only determines the order in which the record will end after the sort is through, not how they will be modified. The comparison routine contains all the code that makes any sort computation different from another in terms of how the contents of the records are changed. The comparison routine has two functions. One function is to determine if the two records being compared are in the same group, generally whether or not their keys are equal, and whether a record from one group will come before or after a record from another group. The other function is to modify the records if they both belong to the same group.

SORT AGGREGATION

Pseudo code will be used to describe the following algorithms . The expression "A[5].(B,C,D)" will define an array of 5 records, where each record has 3 fields, B, C, and D.

For the sake of simplicity we will just show that sort summing works. Note that the addition operation can be replaced by any other valid aggregate operation. The command "SORT(SUM,A)" performs the sort sum over the array A defined by "A[n].(K,V)".

```
boolean function SUM(A1,A2)
        given A1.(K,V)
        given A2.(K,V)
        if  A1.K = A2.K then
                A1.V = A1.V + A2.V
                A2.V = 0
                return(true)
                end if
        if A1.K < A2.K then
                return(true)
                end if
        if A1.K > A2.K then
                return(false)
                end if
end function
```

Figure 1. Sum routine

SUM (Figure 1) is the comparison routine which will, when used in conjunction with SORT, sum all the values in field V of the records for which the K fields are equal. SUM returns a value true if the records A1 and A2 are in the correct order and false if they are not. SUM will put the sum of all the V fields of records of the same group in the first (or smallest) record in the group.

The proof that this will work as described goes as follows. Even though the keys of the records we are comparing may be equal, SUM can affect their ordering by returning to the routing routine the response that either the records are in the correct order (true) or not (false). This in effect gives order within a group. SUM always designates the record that contains the result of the sum as the smaller of the two records and the larger of the two records contains a value of zero. This means that the sum of the value fields of the records of a group will be contained in the record that was designated smaller than all other records of that group. Let us assume, however, that not all values of records in a specific group were summed into the same record. This means that at least two records contain only part of the result for that group. Each one of these records would have been designated less than all records of that group to which it was compared. Yet, the records which contained partial results must not have been compared to any one of the others or the partial results would have been summed into one of the two records. Thus, each one of the records would have been designated the smallest record of the group. Since there is only one smallest record of a group, there can only be one record that contains the result for any group.

A routine corresponding to SUM can be writen for any operation that is both associative and commutative, as described above.

```
boolean function COPY(A1,A2)
        given A1.(K,F,V)
        given A2.(K,F,V)
        if  A1.K = A2.K then
                if A2.F then
                        A1.V =  A2.V
                        A1.V = true
                        return(true)
                        end if
                if A1.F then
                        A2.V =  A1.V
                        A2.V = true
                        return(true)
                        end if
        else
                return(true)
                end if
        if A1.K < A2.K then
                return(true)
                end if
        if A1.K > A2.K then
                return(false)
                end if
end function
```

Figure 2. Copy routine

SORT DISTRIBUTION

Sort distribution is slightly more complex than sort aggregation. The idea in sort distribution is to copy the value of a record in a group of records, which has been flagged as having a valid value for that group, to all records that do not already have that value. The command to perform this would be "SORT(COPY,A)", where SORT is the routing routine, COPY is the comparison routine, and A is the array of records. This array of n records is of the form "A(n).(K,F,V)", where K is the key, F is the valid value flag, and V is the value field. COPY used in conjunction with SORT distributes the correct value for each group to all members of

the group (see Figure 2). Like SUM, COPY returns a value true if the records A1 and A2 are in the correct order and false if they are not. COPY puts the same value in all records of the same group or no value at all if no record of the group had its valid value flag set prior to performing the distribution.

The proof that distribution works is similar to that of aggregation. Note that when two records are determined to be in the same group and one of the records contains a valid value, it is copied to the other record and its valid value flag is set. This in effect cause the record with a valid value to be considered both larger and smaller than a record that does not have a valid value. Thus, at the completion of the sort computation, at least the largest and smallest record of each group that had a record with a valid value, will contain a valid value. Assume that a record without a valid value remained so after the sort was complete. If it was either the largest or the smallest record of the group, then there must not have been a record in the group that had a valid value. If it was not the smallest or the largest value of the group, either there was no record in the group with a valid value or it was not compared to a record in the group with a valid value. If there is a record without a valid value and one with a valid value in the same group, there is such a pair that is logically next to each other that has never been compared. If such a pair exists, there is no way of knowing which one is larger, since they have never been compared. Thus, the sort must not have been complete. Therefore, a record can only be left without a valid value after the sort is complete if there were no records in its group with a valid value.

RAY TRACING BY PARALLEL RECURSIVE SUB-DIVIDING OF SPACE

Originally, ray-tracing algorithms were based on an approach that compared each object with each ray to determine whether the ray intersected the object. More recently, the viewing space has been divided into varying size regions depending on how many objects are in the region (Ref. 2-6). If the region has too many objects in it, it is divided into smaller regions. The rays are traced through the regions until they intersect an object. This sub-dividing of space is done prior to the actual ray-tracing. The collection (a hierarchy) of regions is stored as a structure to be searched during the actual ray tracing.

We recursively subdivide space during ray tracing; therefore we do not have to either store or search the hierarchy of subdivided regions. We also do not actually compare rays to objects to find their intersection, but determine the intersection of subdivided regions (cells) and rays and the intersections of cells and objects. When divided each cell is split into eight subcells of equal size. If a cell does not both intersect a ray and an object it is deleted. Thus, all further processing associated with that cell is terminated. As each remaining cell is subdivided into smaller cells, the only cells that remain are those that intersect at least one ray and one object. The cells are subdivided until the cross-section of the subcell is smaller than some specified fraction of the area that can be viewed through a pixel of the viewing screen.

The position of the intersection of the object and the ray can either be taken as the center of the cell that they both intersect or a more accurate determination of the intersection can be calculated from the analytic description of the object and the ray. Once the intersection of the ray and the object has been determined, reflected or refracted rays may be calculated if further generations of rays are to be simulated, or a brightness value for the ray can be calculated. All ray/object intersections of a generation of rays are calculated before any intersection of the next generation.

This is a highly parallel algorithm since there is no interaction between processing of neighboring cells. The only global processing required is the determination and deletion of all cells that do not intersect both a ray and an object.

The algorithm initially starts with a set of rays and a set of objects. This set of rays usually start at a view point and each ray intersects a simulated viewing screen at different pixel positions.

For the sake of this discussion, we will assume that all computation can be contained within the processor array. In reality data must be swapped to and from the staging memory.

We start with an initial cell that contains all the objects. Each processor holds the description of a ray and an object that have no relation to each other. First it is determined which rays intersect the cell. These rays are then sorted to a region of the array of processors. The remaining rays are considered to be deleted. The same thing is done with the objects. This makes more efficient use of the processors when the ray descriptions are duplicated and moved to processors not being used. At this point there will probably be both rays and

objects left, so there will be no reason to delete the only cell.

Each ray and object will be duplicated eight times, moved to different processors, and paired up with one of the eight subcells of the initial cell. For each ray/cell pair, it will be determined if the ray and the cell intersect. If they do not, then the ray/cell pair will deleted. The same will be done with object/cell pairs.

Deletion of cells that do not intersect both a ray and an object can be handled by the application of sort-computation. This is done by creating two types of records, ray/cell and object/cell, that will be sorted together using a sort distribution operation. Each record will consist of a key field and two flag fields, the ray and the object flag. The key field contains the ID of the cell that intersects the ray or the object. The ray flag is true if the record is a ray/cell record and the object flag is true if the record is an object/cell record. During the sort distribution operation all the ray flags will be anded together for all records with the same key values. This also happens with the object flags. Therefore any cell that has a record that does not have both the ray flag and the object flag set will be deleted. This cycle of dividing cells and deleting ray/cell and object/cell pairs is repeated until the desired intersection accuracy has been obtained. At this point, object descriptions are distributed to where ray descriptions are, based on the common cells that they intersect. The result is a list of all intersections of rays and objects.

The actual implementation on the MPP uses a 16,384 element-wide stack stored in the staging memory, which is manipulated within the array memory of the MPP. The feasibility of the algorithm was demonstrated by James Hurst on a VAX-11/780 and is being implemented on the MPP.

Additional work needs to done to eliminate unnecessary computation, but this is the first step in the parallelizing of ray tracing on a SIMD architecture.

REFERENCES

1. Dorband, John E., "Sort Computation and Conservative Image Registration," Ph.D. thesis, Pennsylvania State Univ., December 1985.

2. Glassner, Andrew S., "Space Subdivision for Fast Ray Tracing," *IEEE CG&A*, Vol. 4, No. 10, Oct. 1984, pp. 15-22.

3. Kajiya, James T., "SIGGRAPH 83 Tutorial on Ray Tracing," *Proc. SIGGRAPH83*, Course 10 Notes, 1983.

4. Kaplan, Michael R., "The Uses of Spatial Coherence in Ray Tracing," *ACM SIGGRAPH '85 Course Notes 11*, July 22-26 1985.

5. Kay, Timothy L. and James T. Kajiya, "Ray Tracing Complex Scenes," *ACM SIGGRAPH '86 Proc.*, Vol. 20, No. 4, Aug. 1986, pp. 269-276.

6. Rubin, Steven M. and Turner Whitted, "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," *Computer Graphics* 14(3), July 1980, pp. 110-116.