4

A Massively Parallel Approach for the Design of a Raytracing Oriented Architecture *

T. Muntean, and Ph. Waille

Introduction

Solving time critical problems requires a computing power of an order of magnitude greater than todays available conventional computers. The use of massively parallel architectures appears to be an attractive and effective way towards the required performances. The ray tracing technique is known as the best synthesis method for the construction of realistic images but also as the most time consuming. Computation time of several hours per image on a conventional mainframe is usual. Fortunately, this technique exhibits a huge amount of potential parallelism and therefore massively parallel architectures fit well and straightforwardly. This paper presents an efficient implementation of the ray tracing algorithm on a dedicated network of transputers. The INMOS's transputers are a family of monochip processors specially designed for parallel, asynchronous architectures without shared memory. The main features of the floating point transputer T800 are:

- a scalar processor providing 10Mips and a floating point unit at 1.5 Mflops
- 4 Kbytes of fast static memory
- 4 bidirectional serial links communicating in parallel at 1.7 Mbyte/s each
- up to 4 Gbytes of external memory
- a hardwired process scheduler and communication controller
- a RISC style instruction set with support of the OCCAM† programming model.

* This work has been partially funded under a CCETT (a division of the french telecoms) contract.

† OCCAM is a trade mark of INMOS Group of Compagnies.

The first part of the paper presents an overview of the ray-tracing technique, the second part deals with the implementation of some of our novel solutions. They are actually related to data base distribution and load balancing.

Principle of Ray Tracing

The ray casting based rendering algorithm consists in a (reverse) reconstruction of the path traversed by the light from its sources to the observer. Through each pixel a so called primary ray starting from the observer is cast.

Each time an object is intersected by a ray, the direct illumination of the object at the intersection point is computed and a new ray is sent back in the direction of each light source. The colour found for the intersection point is the colour that will be attached to the pixel. When the object intersected by a ray is reflexive or transparent, a new ray is recursively sent in the appropriate direction and the corresponding contribution is added to the illumination of the object. This technique is usually referred to as ray tracing. For an introduction to ray-tracing see [1-5].

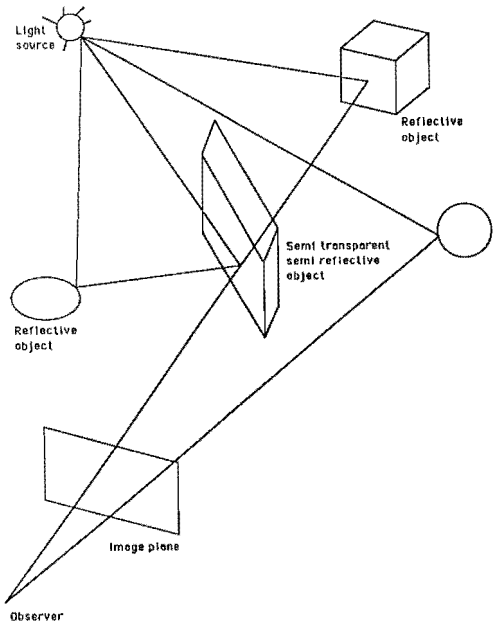

Figure 1: Ray tracing technique reconstructs the light path from the sources to the observer.

Working in a non-projective manner, the ray-tracing method exhibits three important characteristics:

- the method deals with any object specification for which an intersection algorithm is known, allowing an exact (mathematical, procedural) modelling, not restricted to polygons
- it is an elegant and natural method which can take into account complex light effects such as reflection and refraction
- the computational complexity is high; a simple scene described by constructive solid geometry may contain an order of 10^2 primitives and a common resolution will be $10^3 \times 10^3$ pixels (i.e. at least 10^6 rays when just the minimum one ray per pixel is considered).

Ray tracing requires a test of intersection between each ray and each primitive (object). It results an order of 10^8 intersection tests, and therefore a great amount of computation.

Computer-synthesized images often exhibit annoying defects such as jagged edges and distortions of very small objects due to the discrete pixel sampling. Removing these defects is known as antialiasing. Antialiasing usually involves oversampling (computing "sub-pixels" at a higher resolution level), each pixel being a composite value of the sub-pixel samples. Stochastic sampling has been investigated in order to reduce the number of samples to compute.

Sequential Optimizations

For the ray-trace algorithm, most of the computing power is needed for intersection tests. Several optimizations have been proposed in order to reduce their number or their complexity, among which the following:

Bounding volumes

Each (complex) object is surrounded by a simple bounding volume. No intersection test is performed for the rays which do not intersect the bounding volume. Bounding volumes are particularly efficient when dealing with complex objects (e.g. bicubic patches), or scenes combining a lot of primitives ([10-12]).

Spatial coherency and subdivision

Obviously some rays cannot possibly intersect some objects due to their relative spatial positions: in other words, some spatial coherency may be exploited in order to reduce the number of useless tests. For example, Figure 2 shows that the ray can only intersect the objects a, b, c and d. The test with the other objects is a waste of time.

By dividing the scene into three-dimensional regions, and restricting traversal of the tree describing the scene only to the objects situated inside regions visited by the ray, a great amount of intersection computation will be avoided. As the intersection tests are performed in order of increasing distance from the starting point of a ray, their number tends to relate only to visible objects instead of all the objects of a scene.

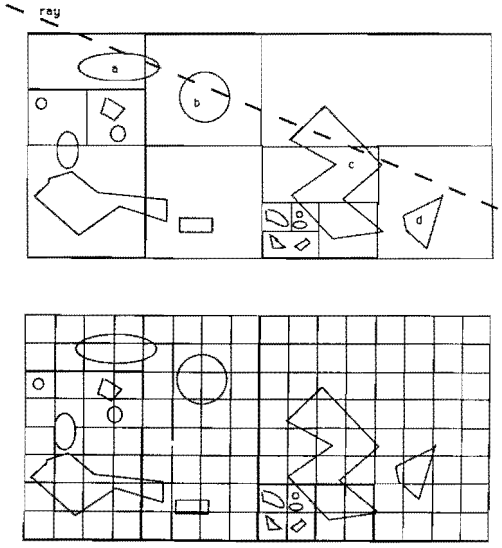

Figure 2: Adaptive and total subdivision.

The time needed to preprocess the scene must be only a small percentage of the total ray-tracing time. Two alternatives are possible:

- perform a regular division, which is not time consuming
- parallelize the division itself.

The best ray-tracing algorithms known today exploit some kind of multidimensional subdivision ([14-22]).

Cacheing intermediate results for complex objects

Intersection tests with some kinds of objects may involve some sophisticated, time consuming algorithms. As an example, bicubic patches and revolution surfaces generally involve a polynomial iterative root finder. In order to avoid convergency problems, a first surface subdivision is performed before the iterative root finder is started. Intersecting fractals involve recursive generation of polygons. Each step also involves (expensive) computation of a bounding volume.

As the same preprocessing is performed for two neighbouring rays, a cache memory may be used to save partial results between rays. Sometimes, this kind of cache technique may drastically decrease the amount computing time ([13] gives an example on fractals with 98% of saving).

Different Levels of Parallelism

The ray tracing algorithm exhibits different kinds of parallelism:

Pixel level

As the values of pixels are mathematically independent, the obvious level of parallelism is at pixel level. When dealing with antialiasing, the same applies to the samples which are composited to produce the pixels values. The pixels may be computed in parallel, but they share the different values of the samples. The amount of parallelism involved is in the order of 10^6.

Rays

As several rays (for reflection, refraction, and light sources) are sent from the intersection point where a ray hits an object, there is a second level of parallelism at ray level. The degree of parallelism is two, three or possibly more in case of multiple light sources.

Geometric transformation

Before the intersection test itself, a reference transformation is applied to the ray coordinates. This linear transformation involves a matrix by vector multiply which uses classical parallel algorithms. The degree of parallelism at this low-level is limited to the size of the matrix, i.e. the space dimension.

Objects and primitives

Let us now consider the scene instead of the rays. As the objects (or the primitives they contain) are independent, the intersection test between a given ray and all objects may be done in parallel. We expect a typical number of primitives per image of an order of magnitude of 10^3 to 10^5 when powerful dedicated machines will be available. Such a number of primitives allow more realistic images, but implies an unacceptable rendition time on today computers.

Spatial region

The same principal of independence may be applied in case of subdivision into spatial regions. The parallelism is then in the order of 10^2 to 10^4, depending on the scene complexity.

Duplicating or Distributing the Data Base

As there is no shared memory in the target architecture (i.e. a transputer network), the designer is faced with an important decision: install a copy of the data base on each processor or split it up and distribute the parts among all the processors.

Duplication and processors farm

In case of data base duplication, each processor owns the entire data base describing the scene. The *processors farm* approach may then be used: a master processor feeds a pool of slave processors and collects the results.

The main benefit of this approach is its inherent simplicity. Any sequential ray-tracing algorithm can be used without modification. Indeed, only the top loop to initialise the pixel computation has to be modified. This leads also to the best computation/communication ratio. Neither communication bandwidth nor load balancing do not need particular care.

The first drawback of this approach is the great amount of memory needed. The size of the data base describing the three-dimensional scenes increases as pictures tend to be more and more realistic. On the other hand, the total amount of memory used for a given picture grows linearly with the number of processors.

The second drawback concerns the optimizations based on cache memory. As the intermediate results for two neighbouring rays may be computed on two differents processors, the locality property is lost, and the cache hit ratio falls down.

Distributing the data base

Each processor is given a subset of the data base and performs the intersection tests for all its objects and all the rays. This does not have the drawbacks of the duplication strategy mentioned above. However, at the cost of an increased complexity. Special attention has to be paid to three importants aspects: how to split and allocate the data base on the different processors, the communication bandwidth required and the load balancing.

Choosing an Algorithm

A good algorithm should exhibit the following properties:

- *low computational complexity*. Spatial or multidimensional subdivision tends to lead to a cost growing in $\log(n)$ instead of n with the number of objects.
- *locality*. The computation of an intersection test should require only a region bounded subset of the data base.
- *low cost preprocessing*.
- *no restriction on the scene*.

The algorithms proposed in the literature, for instance [14] and [19], do not satisfy the locality constraint; non regular subdivision as proposed in [22] involves heavy and difficult to parallelize preprocessing; the solution proposed in [17] tends to restrain the kind of scene to modelize.

The algorithm used in our approach is an optimization of the algorithms proposed in [15] and [18].

Load Characterization

The load associated with a given region depends on the number of rays which intersect the region and the average computing time per ray in this region:

$$L = N_{ray} \times T_{mean}$$

N_{ray} depends mainly of the external environment of the region (light sources, reflective surfaces, shadowing $\cdots$) and of the volume of the region. T_{mean} reflects the region's content (number and complexity of the objects).

Unfortunately, only the volume of the region and the size of its restricted data base may be evaluated without involving the ray tracing algorithm itself. As the subdivision is adaptive, we may roughly consider an uniform load and perform a dynamic load balancing during the rendering step.

Communication and Reconfiguration

The serial nature of the transputer links allows dynamic reconfiguration of the transputer networks and cascadable programmable switches of up to 72 links have been developed for the Supernode architecture.

A possible and elegant method for ray communication between processors (in function of region allocation strategy) consists to partially reconfigure the network of transputers for each communication. A direct connection is temporarily established between the source and destination processors and as soon as the ray is transmitted the connection may be removed.

A rough estimation shows an average computation time per ray of about ten milliseconds. As each reconfiguration step involves an arbitration between the different requests for connecting a given transputer link, this method leads to a bottleneck in the command of the switch. This becomes worse when the number of processors in the network grows.

Allocation Strategy

Choosing a total subdivision, all regions would be of the same size and may be easily mapped onto a two- or three-dimensional grid of processors. Communications would be limited to neighbouring regions and processors. When choosing the adaptive subdivision, an allocation strategy has to be defined in order to keep neighbouring regions as much as possible on neighbouring processors.

A simple method considers each region (of variable size) as the union of sub-regions of the smallest size of the subdivision and then one can proceed as with a total subdivision. In such case, it seems highly desirable that the number of sub-regions be a multiple of the number of processors. While regions of larger size will be replicated on several processors, the principle of the data base distribution is not violated. Indeed, these regions may only contain a few number of objects.

Two options are allowed: map neighbouring regions on neighbouring processors or on the same processor. The second method obviously results in less communication bandwidth, but the first one has been chosen for it allows a better static load balancing. Indeed, global load disparities in the scene will be distributed on more processors. As a result, processors at opposite extremities of the grid are allocated neighbouring regions, and have to be directly connected together. Due to the number of links available, a two-dimensional network of transputers can be used. The example below shows a possible allocation scheme for a two dimensional division algorithm.

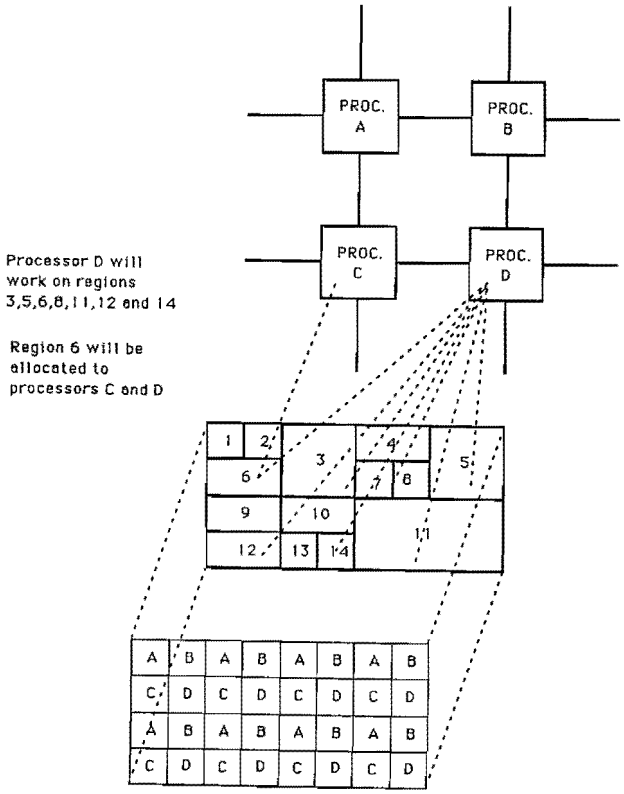

Figure 3: Allocation of regions to processors.

Link Performances and Routing

Once subdivision is done, the mapping of spatial regions onto processors will take place. Each processor is given a data base restricted to the regions it handles. Each ray will be sent from its origin and will travel from region to region (and so from processor to processor), intersects something or leaves the scene.

As neighbouring regions may be placed on non neighbouring processors, each working transputer has to perform a routing algorithm as well. The simulation of the communications on the T800 has shown a significant loss of communication performance when the same transputer performs the computation as well as the routing tasks. On the other hand, some links have to be freed on each working transputer in order to communicate with the environment (local data base acquisition and results extraction). Therefore extra processors for routing have been added to each working transputer.

Dynamic Load Balancing

The allocation strategy used tends to reduce the static load disparities (i.e. before the actual computation). Also some kind of dynamic load balancing has to be performed during the ray tracing itself. Real examples show that the restricted data base associated with each region will usually not exceed 10 Kbytes. This traffic could be transferred on a transputer link in only a few milliseconds.

The load balancing unit (LBU) of our architecture is functionally described hereafter.

Each working transputer is connected through a special control bus to a master controller and to a programmable switch, also controlled by the master. Each working transputer is periodically tested on its actual load. When disparity appears between a pair of workers, the master establishes a direct connection between them through the switch. The overloaded transputer sends a copy of its database to the underloaded worker and the former submits part of its job to the later. The connection is removed when either one of the two transputers returns to an acceptable load.

Displaying the Results

A dedicated display transputer drives a second switch to which two of its four links are connected. The free link of each worker transputer in a basic module is connected to this switch as well. As the ray tracing progresses, the display transputer interrogates the worker transputers in turn for pixel values already available. We expect a single link to work at about 100 Kbytes per second, switching overhead included. A whole image of 1000×1000 pixels of three colours bytes each will be transferred in less than 15 seconds, far less than the expected computing time of a few minutes.

Conclusion

Good performances may be obtained by running the ray tracing algorithm on massively parallel architectures, and a novel transputer based architecture has been proposed. This architecture can be efficiently emulated on a Supernode of 64, working transputers (four basic Supernode modules). A future paper will present the experimentals results and performances.

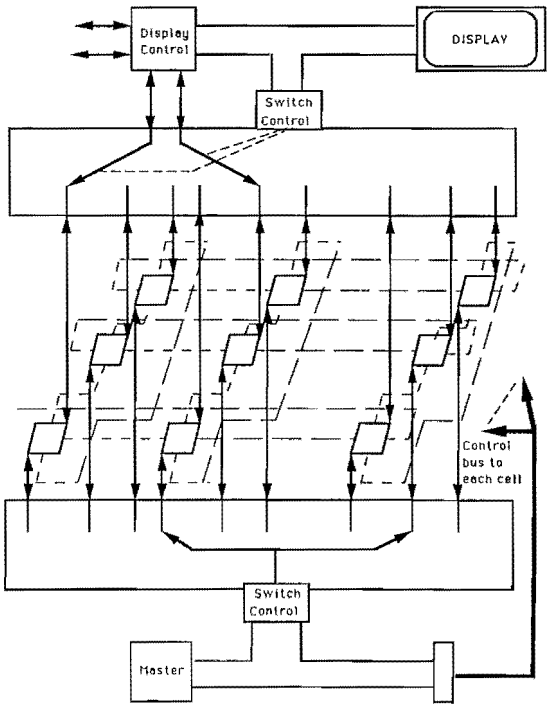

Figure 4: Overall Transputer Based Architecture.

Bibliography

[1] A. Amanatides, "Realism in Computer Graphics: a Survey", *IEEE Computer Graphics & Applications*, January 1987

[2] S.D. Rott, "Ray casting for modelling solids" *Computer graphics and image processing*, vol. 18, 1982

[3] J.T. Kajiya, "New techniques for ray tracing procedurally defined objects", *Computer Graphics*, vol. 17, no. 3, July 1983

[4] Ch. Bouville, R. Brusq, J.L. Dubois, and I. Marchal, "Generating high quality pictures by ray-tracing", *Computer Graphics Forum*, vol. 4, 1985, North-Holland

[5] R.A. Hall, and D.P. Greenberg, "A testbed for realistic image synthesis" *IEEE Computer Graphics & Applications*, November 1983

[6] R.L. Cook, T. Porter, and L. Carpenter, "Distributed Ray Tracing", *Computer Graphics*, vol. 18, no. 3, July 1984

[7] D.P. Mitchell, "Generating antialiased images at low sampling densities", *Computer Graphics*, vol. 21, no. 4, July 1987

[8] J. Amanatides, "Ray tracing with cones", *Computer Graphics*, vol. 18, no. 3, July 1984

[9] P.S. Heckert, and P. Hanrahan, "Beam tracing polygonal objects", *Computer Graphics*, vol. 18, no. 3, July 1984

[10] H. Weghorst, G. Hooper, and D.P. Greenberg, "Improved computational methods for ray tracing", *ACM Transactions on graphics*, vol. 3, no. 1, January 1984

[11] J. Goldsmith, and J. Salmon, "Automatic creation of objects hierarchies for ray tracing", *IEEE Computer Graphics & Applications*, May 1987

[12] S. Rubin, and T. Whitted, "A 3-Dimensional representation for fast rendering of complex scenes", *ACM Computer Graphics*, vol. 14, no. 3, July 1980

[13] Ch. Bouville, "Bounding ellipsoids for ray-fractal intersection", *ACM SIGGRAPH 85*

[14] T. L. Key, and J. T. Kajiya, "Ray tracing complex scenes", *ACM SIGGRAPH 86*

[15] M. R. Kaplan, "Space tracing, a constant time ray-tracer", *SIGGRAPH 85*

[16] A. Fujimoto, T. Tanaka, and K. Iwata, "ARTS: accelerated ray-tracing system", *IEEE Computer Graphics & Applications*, October 1984

[17] G. Wyvill, T. L. Kunii, and Y. Shirai, "Space subdivision for ray-tracing in CSG", *IEEE Computer Graphics & Applications*, April 1986

[18] A.S. Glassner, "Space subdivision for fast ray tracing", *IEEE Computer Graphics & Applications*, October 1984

[19] J. Arvo, and D. Kirk, "Fast ray tracing by ray classification", *Computer Graphics*, vol. 21, no. 4, July 1987

[20] S. Coquillart, "An improvement of the ray tracing algorithm", *EUROGRAPHICS' 85*

[21] M. Dippe, and J. Swensen, "An adaptive subdivision algorithm and parallel architecture for realistic image Synthesis", *Computer Graphics*, vol. 18, no. 3, July 1984

[22] T. Priol, B. Arnaldi, and K. Bouatouch, "A new space subdivision method for ray tracing CSG modelled scenes", INRIA Research Report No 613 (Rennes France)

[23] K. Bouatouch, M. O. Madani, T. Priol, and B. Arnaldi, "A new algorithm of space tracing using a CSG model", INRIA Research Report No 612 (Rennes France) 1987

[24] M. O. Madani, "Etude d'architecture de calculateur parallèle adaptée à la synthèse d'image 3D par lancer de rayon", *DI Thesis* ENST (Brest, France) 1986