[FVFH90] FOLEY, J.D., - VAN DAM, A., - FEINER, S., - HUGHES, J.: Computer Graphics:Principles and Practice, Addison-Wesley Publishing Company, Inc., 1990.

[Koli93] KOLINGEROVÁ, I.: Využití duálního prostoru pro metodu sledování paprsku, Zimní škola PG, Plzeň, 1993.

[KuWy??] KUNII, T. L., - WYVILL, G.: CSG and Ray Tracing Using Functional Primitives, pp. 137-152, Computer Generated Images: The State of the Art.

[Mart92] MARTINKA, J.: Ray Tracing, ŠVK 1992, MFF UK Bratislava.

[Mart93] MARTINKA, J.: Ray Tracing (implementácia urýchľovacieho algoritmu), ŠVK 1993, MFF UK Bratislava.

[MSHG92] McNEIL, M.D.J., - SHAH, B.C., - HÉBERT, M.-P., - LISTER, P.F., - GRIMSDALE, R.L.: Performance of Space Subdivision Techniques in Ray Tracing, Computer Graphics forum, Volume 11 (1992), number 4, pp. 213-220.

[Ruži91] RUŽICKÝ, E.: Úvod do počítačovej grafiky, skriptum, MFF UK Bratislava, 1991.

[SoŽa89] SOCHOR, J., - ŽÁRA, J.: Světlo a stín v počítačové grafice, Proceedings of Moderní programování I. diel, 1989

[Sung91] SUNG, Kelvin: A DDA Octree Traversal Algorithm for RayTracing, Eurographics '91, pp. 73-85.

[TaLu88] TANG, Zesheng, LU, Shengkai: A New Algorithm for Converting Boundary Representation to Octree, Eurographics '88, pp. 105-116.

[Zeit92] ZEITLBERGER, René: The Octree Data Structure, Seminar aus Informatik, Technische Universität Wien, 1992.

PARALLELISATION OF THE RAY–TRACING ALGORITHM

Jiří Žára *(zara@cs.felk.cvut.cz)*
Aleš Holeček *(zholecek@sun.felk.cvut.cz)*
Jan Přikryl *(zprikryl@sun.felk.cvut.cz)*

CTU, Fac. of Electrical Eng.
Dept. of Computer Science
Karlovo nám. 13,
121 35 Praha 2.

Abstract

The two typical methods for distribution of ray–tracing rendering algorithm are presented in this article. The implementation of a distributed ray-tracer on a network of UNIX workstations is described in details. The first results are discussed from the point of view of memory load, time of computation and cost of communication among processes.

Keywords

computer graphics, rendering, parallel algorithm, ray-tracing, PVM.

1 Ray–tracing algorithm

The methods of computer generated, realistic looking pictures of three dimensional scenes are characterized by their extremely high claims on computing equipment and the fact that they are incredibly time consuming. Typical algorithms (ray tracing, radiosity method) are so complex and complicated, that their direct transformation into computer hardware is not effective yet. The performance capacity of a single CPU and the amount of memory available on today's personal computers and workstations are still low for large, reality describing scenes when using the algorithms mentioned above.

Although some rendering methods were already implemented into hardware of graphics workstations (z–buffer, Gouraud shading), the ray–tracing technique is still too complex method and its hardware support is a task for computing equipments in the future.

We can compare typical features of now-a-days rendering methods done by hardware with ray-tracing in the following table:

z–buffer & Gouraud shading	ray–tracing
integer arithmetics in raster	floating point arithmetics in 3D space
sequential processing of polygons	computations in whole 3D scene
constant number of rendered faces	recursive generation of rays

The disadvantages of the ray–tracing algorithm are clear, but the quality of images rendered by ray–tracing is so high, that this algorithm is used in many applications. Its improvement and increasing of its efficiency are one of current topics in computer graphics.

The ray–tracing algorithm is based on tracing of rays, that are shot from viewpoint through the screen window to the scene. When an intersection point between a ray and the nearest solid or face is found, a new generation of rays is created — one reflected ray, one refracted ray (if solid is transparent or semitransparent) and several "shadow" rays (one for each light source). Recursive creation of rays is finished either after several generations or in case of achievement of

certain conditions (reaching the lower limit of the intensity that is decreased in every generation). Resulting images then contain shaded solids with shadows and reflected objects.

The main problem of the discussed algorithm is the huge number of tests for solid—ray intersections. In a typical scene, several thousands of intersections must be computed, but only the nearest intersection point is used for later color evaluation. The time needed for finding the nearest intersection of the ray is about 60–90% of whole rendering time, depending on the amount and quality of textures and other special features.

The effort for decreasing the time of a ray–tracing corresponds very often with a decreasing of number of intersection tests. One way to do this is to use more independent processors for rendering one picture.

2 Ways of distributing the ray–tracing method

There are two different approaches to distribute the rendering of a ray–traced scene:

- screen subdivision
- space subdivision

Using the first approach, each processor receives information about the whole scene and a part of the screen that has to be filled up with pixels. Each processor in the system is absolutely autonomous, that means there is no interprocessor communication in the system. It decreases the time of computation but not the amount of memory needed for the description of a scene.

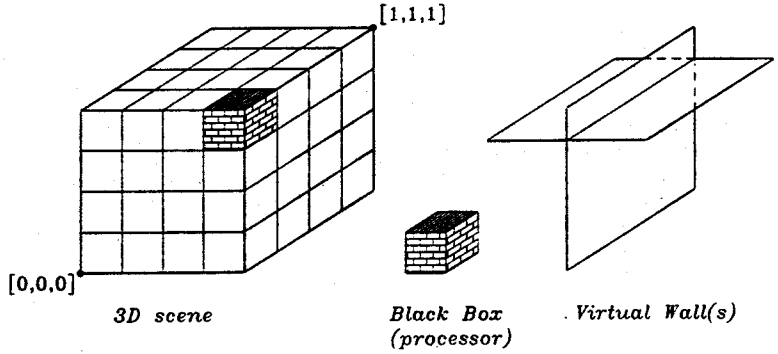

Figure 1: Space subdivision using "Virtual Walls" concept

The aspect of insufficient memory is one of the main limitations for rendering reality, well describing scenes. The solution for that is the second type of distribution. It is suitable for computers with a lower capacity of memory and with the ability of fast interprocessor communication (transputers). Instead of dividing the screen we divide the object space [2]. Every processor is then responsible for the computation of all rays going through its part of space. Messages going through the system carry information about the rays. That is why the interprocessor communication plays such an important role in the evaluation of the scene.

We decided to implement a distributed ray–tracing algorithm that uses the space subdivision approach. The space subdivision technique called "Virtual Walls" [1] was designed and improved at the Paderborn Center for Parallel Computing. The 3D model space is subdivided by walls that are parallel with coordinate axes, as shown in Fig.1. The division can be uniform or adaptive with respect of displacement of objects in a scene. We'll get a set of volume boxes (called "Black Boxes" in our implementation) which are surrounded by walls as a result of this subdivision.

Every box performs one ray–tracing algorithm, but only with its local scene that is subset of the complete original scene. This is an important decrease of time — the amount of intersection computations is much lower comparing with the whole scene. The higher efficiency of geometrical computations costs greater communication needs. Once a ray generated in a box leaves its boundaries, it is sent to another neighboring box. In a common situation one ray goes through many boxes and several communication packages have to be sent to the parallel computing environment for this ray.

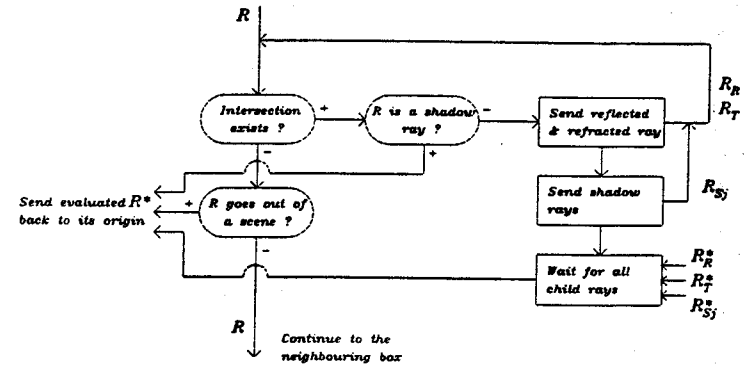

Figure 2: Flow of rays R in a "Black Box"

Fig.2 shows scheme of data flow in a "Black Box". If a ray intersects any object inside the box, a new generation of rays (called child rays) is created and sent from the point of intersection to the space. The original ray must be stored in a local queue and it waits for evaluation of all its child rays. Local queue for waiting rays can become long, especially in the case of a large number of boxes, a high recursion level and a big number of light sources that are reason for sending shadow rays.

Every ray holds information about its geometrical characteristics and address of its original "Black Box". After complete evaluation of a ray, a small data package with color information is sent to the "Black Box", in which the origin of this ray was previously computed.

3 Implementation under PVM

For the implementation we used two different kinds of platforms:

1. heterogeneous network of processors using standard network methods of communication (workstations using TCP/IP),

2. special homogeneous processor network (transputer grid).

In this article we present the first platform, because the second one is still under development. The software package PVM (Parallel Virtual Machine) was used for implementation of ray-tracing in a network of Sun workstations. PVM is a simple but powerful tool, that joins many UNIX based workstations into one parallel computational resource. A programmer works with the standard programming language C and uses PVM as a library for:

- initiation of single processes
- termination of processes
- communication among processes (sending and receiving messages)
- synchronization of processes

Every process can run either on a certain computer identified by its network ID or on certain computer architecture (SUN, SGI, Cray, CM2, ...) or on an arbitrary computer in whole network. This choice depends on a programmer. In our implementation we map the processes into computers specified by ID. This approach enables better work loads of computers in a network, especially in the case of a small number of workstations compared with the number of processes. More powerful computers can perform several processes, slower machines perform only few.

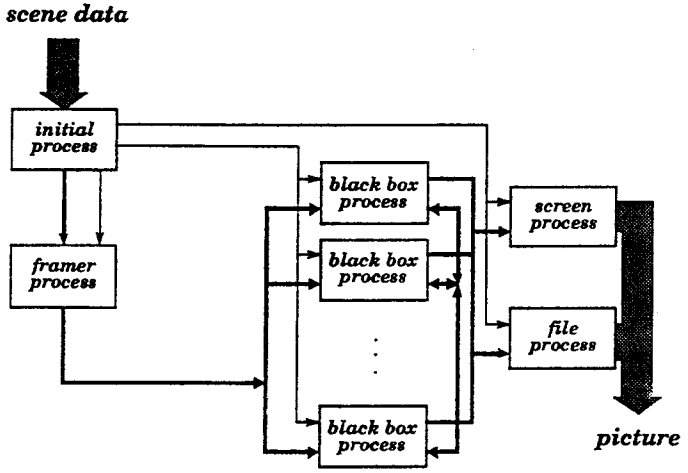

Figure 3: PVM processes in distributed ray-tracer

The structure of computing processes and data flow are shown on Fig.3. Each computer in a network can perform several processes named *Black box*. These processes are the core of the system.

They contain the ray-tracing algorithm and the communication layer, which makes each individual process able to evaluate and send rays for evaluation to surrounding *Black boxes*. It also provides the communication with other processes of the system then *Black boxes*. One of them is *Initial process*. Its responsibility is to start up the application, to produce routing tables and to distribute the global scene among the *Black boxes*. The two other are *Output processes - File* and *Screen*. Their task is to visualize and store computed image. There is also a *Framer process*, which allows "walking through" the scene by changing the position of the observer and starting the computation from the beginning. This process has not been implemented yet.

4 Results and conclusions

We checked the strength of our implementation on a grid of $4 \times 4 \times 4$ processes and network of 8 SUN-SPARC computers. The distributed ray tracing algorithm was tested on scenes containing a large number of solids. Our typical testing scene was built from 5.000 spheres, the depth of recursion was 5 and the resolution of the image was 600×600 pixels.

The time for computing such scenes was approximately one hour and forty five minutes. This result is considered good, although it is observed that some improvements in the implementation have to be made.

Getting exact answers for some questions touching mainly the communication in the system seems to be difficult if this application is performed on a public network. There are other time consuming processes running on the computers, which make the measuring of the performance capacity almost impossible.

Some observation was made that helped us to upgrade the core of implemented application. One of the problems that appeared were too long local queues with rays waiting for the return of their evaluated children. The resolution was using priority in selection which ray will be evaluated next. This decreased the entropy of the computational system. Using hash tables decreased the time needed to access the rays waiting in those queues.

Other speed up is by involving new high efficiency ray-tracing procedure. This algorithm is based again on volume subdivision of the object space into octree. Upgrading is also expected after implementing some of the load balancing methods which will lead to exploiting nearly the full performance capacity of the system. The strategy of load balancing is based on a movement of the "Virtual Walls" and redistribution of a scene.

Also the reduction of the amount of transferred data by collecting rays into bigger packages before sending and using UDP protocol will bring a major decrease of time needed for communication and evaluation of the scene.

Literature

1 Menzel, K., Ohlemeyer, M.: **Walking-Through Animation in 3D-Scenes on Massively Parallel Systems**, Visual Computer, International Journal of Computer Graphics, Springer International, 1992.

2 Pitot, P.: **The Voxar Project**, IEEE Computer Graphics & Applications, January 1993, pp.27-33.