Experimenting with a parallel ray-tracing algorithm on a hypercube machine

Thierry Priol, Kadi Bouatouch

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 843

EXPERIMENTING WITH A PARALLEL RAY-TRACING ALGORITHM ON A HYPERCUBE MACHINE

Thierry PRIOL
Kadi BOUATOUCH

MAI 1988

RR-0843

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

EXPERIMENTING WITH A PARALLEL RAY-TRACING ALGORITHM ON A HYPERCUBE MACHINE

Thierry PRIOL
Kadi BOUATOUCH

Avril 1988
24 pages

Publication Interne n° 405

Abstract :

A parallel space tracing algorithm is presented. It subdivides the scene into regions. These latter are distributed among the processors of an iPSC hypercube machine designed by Intel company. Each processor subdivides its own region into cells to accelerate the ray tracing algorithm. Processors communicate by means of messages. The pyramidal shape of the regions allows the deletion of the primary ray messages. A method of performing a roughly uniform load distribution is proposed.

UN ALGORITHME PARALLELE DE LANCER DE RAYON SUR UN HYPERCUBE iPSC

Résumé :

Un algorithme de lancer de rayon sur un hypercube iPSC est présenté. La scène est subdivisée en régions qui sont associées à chaque processeur. Chacun d'eux subdivise la région qui lui est associée afin d'accélérer le calcul d'intersection entre les rayons et les objets appartenant à cette région. Pendant le calcul de l'image, les processeurs s'échangent des informations (rayons) par l'intermédiaire de messages. La forme des régions associées à chaque processeur permet de calculer localement les rayons primaires. Une méthode permettant une charge de travail équilibrée pour chaque processeur est également présentée.

Experimenting with a parallel ray-tracing algorithm on a hypercube machine *

Thierry Priol
Kadi Bouatouch
IRISA †
Campus de Beaulieu
35042 Rennes Cedex
France

Abstract

A parallel space tracing algorithm is presented. It subdivides the scene into regions. These latter are distributed among the processors of an iPSC hypercube machine designed by Intel company. Each processor subdivides its own region into cells to accelerate the ray tracing algorithm. Processors communicate by means of messages. The pyramidal shape of the regions allows the deletion of the primary ray messages. A method of performing a roughly uniform load distribution is proposed.

Key Words : image synthesis, parallel ray-tracing, hypercube.

1 Introduction

The latest research in computer graphics has provided several techniques of rendering high quality images including stochastic sampling [7,11,21] and sophisticated light models [6,8,18,19,27].

The rendering algorithm which may take into account all these techniques is ray tracing. It simulates the operation of a camera, following light rays in reverse order. It consists of shooting rays from an observer through a simulated screen plane towards the objects of a scene (primary rays). The program computes the intersection of each ray with each object, and determines which intersection is closest. Light sources' contributions to the pixel intensity are computed by shooting rays from the intersection point to each light source (light rays) and determining if the rays are occluded by some solid objects. If this is the case, the relevant point is shadowed. According to the photometric properties of the objects, new rays are shot from the closest intersection point, in order to take into account the contribution to the pixel intensity of the neighboring objects [8,17,28]. Indeed, if the object is transparent, then a new ray is shot in the refracted direction and if in addition it is reflective, then a new ray is shot in the reflected direction (secondary rays). Consequently, the number of rays is very important. This yields a lot of computations of intersections between rays and objects, and makes the ray-tracing technique extremely time-consuming.

Two approaches have been attempted in order to overcome this large number of intersections: algorithmical and architectural.

*This work has been supported by C^3 and by the CCETT (Centre Commun d'Etudes de Télédiffusion et Télécommunications) under contract 86ME46

†Institut de Recherche en Informatique et Systèmes Aléatoires.

1.1 Algorithmical approach

Two classes of algorithms have been proposed in the literature. The methods of the first class involve the creation of a tree of bounding volumes whose leaves are the extents of the objects and the nodes represent the bounding volumes of parts of the scene [24,25]. If a new ray fails to intersect the extent of an object, then it cannot intersect the object itself. This saves many computations.

As for the methods of the second class, they involve a 3D subdivision of the bounding volume of the scene which is generally a rectangular parallelepiped whose faces are perpendicular to the view coordinate axes [2,3,13,14,20,29]. Indeed, this bounding volume is subdivided into 3D regions containing a small number of objects. A ray which enters a region, intersects only those objects lying in this region. If no intersection is found or the intersected objects are all transparent (in the case of light rays), a computation of the next region traversed by the ray is performed.

These subdivision methods have proved their efficiency since they can reduce considerably the synthesis time, but at the expense of a more important memory requirement.

1.2 Architectural approach

All the parallel machines which have been completed or proposed can be classified in three groups. Those of the first group [4,23] use the fact that each pixel may be processed independently, thereby the pixels are divided evenly among the processors. There is no problem of interprocessor communication since the database is duplicated in each processor's memory. Even in the case of scenes of moderate complexity (several hundred objects) the performance is degraded due to the large amount of ray-objects calculations. One solution is to subdivide the space containing the scene and duplicate the data base associated with this subdivision, in each processor's memory. Unfortunately this would require a very large memory.

As for the machines of the second group [15], the objects of the scene are distributed among the processors. The method involves a tree of extents used for intersection computation. Only the top levels of this tree are replicated in every processor. The leaves of the subtree constituted by these top levels, enclose the parts of the database distributed among the processors. One or more parts of the database and a subset of the pixel array are controlled by each processor. The drawback of this technique relies on the fact that the criterion of distribution is not easy to determine automatically, and in addition, a lot of time is spent in the traversal of the tree of extents.

The machines of the last group are based on the subdivision of the space into 3D regions. Each of the processors is assigned one or more regions and each region contains a part of the database. This latter is then distributed among the processors. Neighbouring processors contain adjacent regions and communicate localy via messages. A very important source of inefficiency, inherent in these machines, is the load imbalance problem. Indeed, the processors near the light sources and those controlling the regions located in the middle of the scene, are more loaded than the processors near the edges. To overcome this problem, some attempts have been proposed in the literature [10,22]. They consist of a dynamic redistribution of the load among the processors, in order to make it uniform. We will see in the next section that these techniques raise many deficiencies.

In spite of these remarks, we think that the machines of the third group would be the most interesting if we could resolve all (or part of) the problems due to this kind of machine. These machines are presented in the next section and parallel space tracing is described in the third section. The last section is reserved to the conclusion.

2 Machines based on space subdivision

Firstly, in this section, we describe three machines based on the distribution of 3D regions among the processors and which have been proposed by Cleary [5], Dippe [10] and Nemoto [22]. Then we make remarks about some aspects of their implementation and raise some problems which have not been emphasized by their simulation and which influence considerably the efficiency of these machines.

2.1 Description

To our knowledge, three machines have been proposed in the literature but none has been completed.

- Cleary's machine

  This machine consists of an array of processors. The authors proved that a 2D array is better than a 3D one, since it reduces the number of messages exchanged by the processors. We approve of this since our experience on an iPSC hypercube has shown that even with a 2D array, the number of messages is so large that the associated queues are rapidly saturated. In this machine, each processor is connected to its four neighbors by means of dual port memories. It may happen that several processors contribute to the final intensity of a pixel. These contributions are passed from processor to processor back to the processor where the ray started. This is one of the drawbacks of this implementation since the overhead of messages is increased.

- Dippe's machine

  The architecture is a 3D array of processors, each one is assigned one or more regions. The shape of these regions are general cubes which are general hexahedra. A set of tetrahedra is constructed with groups of six tetrahedra forming a cube, which are then arranged to fill the space occupied by the scene. The boundaries of the regions are moved to assure a roughly uniform distribution of load. When a region's load is higher than those of its neighbors, some load is transferred to them. This is done by moving the corners of a region. The redistribution of load is performed by means of messages called redistribution messages. Each ray resulting from a ray-object intersection contains its contribution to the intensity of a pixel; thus reducing messages.

- Nemoto's machine

  The regions resulting from the space subdivision, are orthogonal parallelepipeds which consist of several unit cubes. A unit cube has a size equal to one, and edges parallel to each axis. The architecture is a 3D array of processors; each one has six connections to its six neighbours and is assigned one region. In order to avoid a load imbalance, the loads of two neighboring regions are compared by the associated processors. If the load of one region is lower than that of another region, and the lower load is under the given threshold value, the separating face is slid by one unit along the axis perpendicular to this face.

2.2 Remarks

Among the three machines described previously, those of Nemoto and Dippe seem the most realistic because they try to solve the problem of load imbalance. Nevertheless, many problems remain to be solved. This leads us to make remarks concerning the questions of space subdivision, dynamic load redistribution, distributed algorithms and overhead of interprocessor messages.

- Space subdivision

  In Dippe's machine, the movement of a ray (or objects in case of load redistribution) from a region to its neighbor involves a very expensive boundary intersection. This is due to the subdivision of the space into tetrahedra. As for Nemoto's machine, the regular space partition becomes irregular after a dynamic redistribution load. Indeed, a region may become adjacent to several other regions. This makes it expensive to route the messages on a 2D array.

- Load redistribution

  The load redistributions proposed are interesting but involve a lot of messages, whereas messages associated to primary, secondary and light rays are yet very numerous. This load redistribution is not so simple because the movement of a corner or a face affects all the adjacent regions. Which corner or face is selected first? What is the behaviour of the algorithm when all the processors are adjusting their load at the same time? What is the periodicity of the load redistribution? This latter may be a source of oscillations. How to avoid it?

- Distributed algorithms

  Due to the large number of messages, some processors may be in a situation of deadlock. How to avoid it ? This is a crucial problem which has never been evoked by the authors quoted previously. A second question arises: what is the algorithm of termination which does not affect the efficiency of the distributed machine. A termination algorithm will be given in the next section.

- Messages overhead

  It is clear that if we try successfully to reduce the number of messages, we can improve the efficiency of these machines and avoid the problems evoked previously. For example, light rays involve a lot of messages and make the processor near them, very busy. Finding a means for suppressing them would be very effective. We will propose an efficient one in the next section.

The following section presents some solutions for the evoked problems. We have chosen to implement these solutions on an existing parallel machine which is an iPSC hypercube designed by Intel Company, because simulation does not raise the problems inherent on a distributed machine.

3 Implementation of our algorithm an iPSC hypercube

Our algorithm is based on the distribution of the database among the processors of a multiprocessor machine. It is implemented on an iPSC hypercube which is a multiprocessor system consisting of

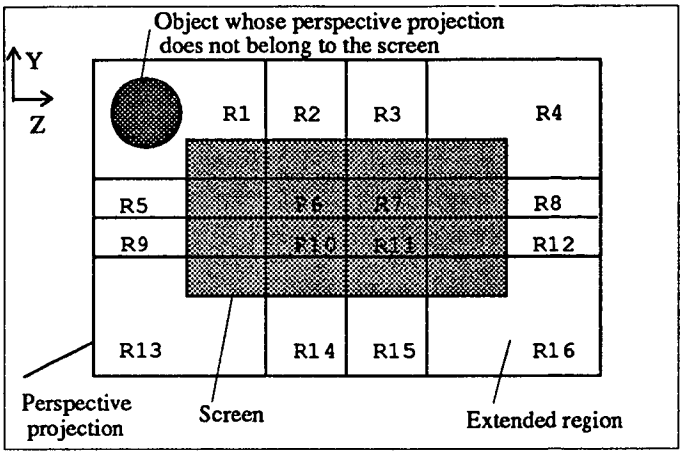

Figure 1: Subdivision in regions.

64 processors. Each processor is a 80286 microprocessor supplied with a 80287 floating-point coprocessor and 4.5 Mbytes of local memory. The topology of this machine is a hypercube. It allows a large choice of architectures (2D or 3D array, ring, etc...).

In a first approach, we have chosen a 2D array due to the reasons evoked previously. Each processor is connected to its four neighbors and is assigned one region containing a part of the database which is a CSG tree. These regions result from a space subdivision technique which is described below.

3.1 Space subdivision into regions

The method is illustrated by figures 1 and 2. The screen is subdivided into a 2D grid. All the elements of the grid (called pixel areas) contain roughly the same number of pixels. A 3D region controlled by a processor is a pyramid whose back face lies on the back face of the extent of the scene, and whose front face is a pixel area. The shape of the region is such that a primary ray which is shot in a region, does not leave it. This decreases the messages overhead since the transmission of primary ray messages is avoided.

The regions at the edges of the screen are made larger in order to take into account the secondary rays intersecting the objects whose perspective projection does not lie on the screen. Indeed, they are extended up to the edges of the perspective projection of the front face of the scene extent.

This subdivision does not sufficiently solve the problem of load imbalance since it is just based on a uniform distribution of pixels. It is just implemented to raise the problems evoked previously and to solve a part of them. A more efficient subdivision method performing a roughly uniform distribution load, has been implemented. It is discussed in section 3.7.1.

3.2 Overview of our algorithm

The algorithm consists mainly of four phases:

1. The host processor of the iPSC subdivides the space into 3D regions as described above. Then it assigns to each processor a region and a part of the data base which results from a *CSG tree pruning* technique [2,3,26].

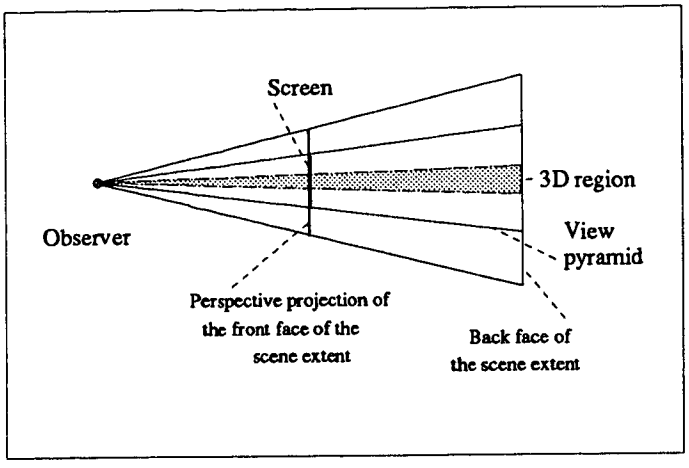

Figure 2: Projection of the 3D regions on a XZ plane.

2. Each processor subdivides in its turn its own 3D region into subregions called cells [2]. The result is a set of cells which are linked together by means of four pointers associated with the cell corners.

3. When all the processors have accomplished their subdivision, they notify (by messages) the host processor which sends them a message permitting them to start the synthesis phase.

4. Synthesis phase: each process shoots primary rays through its pixel area. These rays remain in the associated region and are not transmitted to the neighboring regions since the shape of a region is a pyramid. These primary rays yield light rays which may be transmitted to the neighboring regions by the sending of messages, which is the case in our present implementation. It will be seen later that these messages can be avoided. Primary rays generate also secondary rays whose associated messages are passed to the neighboring processors. Each processor is assigned four FIFO queues in order to save the messages which must be send to the neighboring processors. When a processor receives a ray, it computes its contribution to the intensity of the associated pixel and then transmits this pixel to the host processor to cumulate it to the contents of the frame buffer location associated with the relevant pixel. This is possible since the computation of the final intensity of a pixel is distributed among several processors.

These different phases are detailed below.

3.3 Subdivision of a region

Let us show now how each of the processor subdivides its own region. With each region is associated a subtree which is the restriction of the whole CSG tree, representing the scene, to the objects lying in this region. The subdivision technique is described in [2]. It consists in subdividing the screen into two halves along each axis x and y. This process is applied recursively on each half and terminates when a certain level of subdivision is reached. The result is a set of pixel areas. Each of them is related to a region whose extension along the depth axis z yields 3D cells called supercells. To take advantage of the spatial coherence of the scene, a spatial subdivision, named *depth*

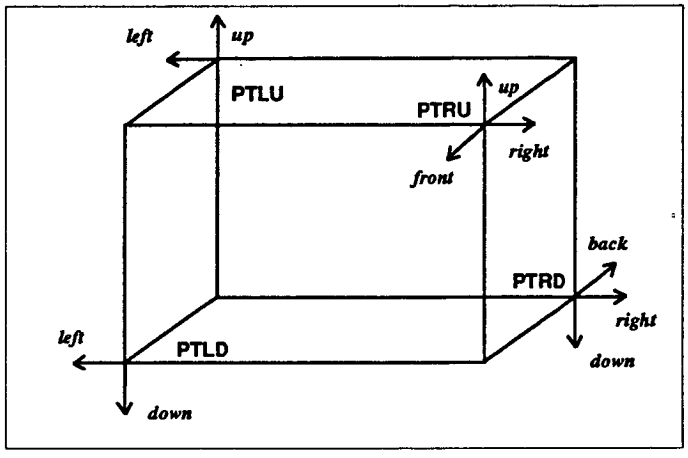

Figure 3: Connectivity using corner pointers

partioning, is accomplished on the super-cells. Indeed, since each super-cell contains primitives which are distributed along the z axis, the faces of the bounding boxes of these primitives which are perpendicular to the z axis, are used to divide the relevant super-cell into cells. A region associated which each processor is then a set of cells. The cell connectivity is assured by means of pointers (figure 3).

PTLU, PTRU, PTRD, PTLD name each corner used in our method and up, down, front, back, left and right are the names of the associated pointers according to the direction of the adjacency relation. With this technique, we can follow the path of a ray across the cell structure.

The search for the next cell along a ray path involves the computation of intersections of this ray with the faces of a cell. Since cells are polyhedra, this may be time consuming. We have resolved this problem by expressing a ray in two coordinate systems. The first one is the eye coordinate system and the second is such that a cell becomes a rectangular parallelepiped whose faces are perpendicular to the axes (figure 4). This solution is also used by a processor, to know which processor is controlling the next region along the ray path.

3.3.1 Determining the entry cell

When a processor receives a ray, it must be able to determine the cell of its own region which is pierced by this ray. To do that, each processor localizes two particular cells during the phase of subdivision into cells of its own region. These are the right-up and left-low cells as shown in figure 5. Thanks to their connectivity pointers, these two cells allow the determination of the cell pierced by the received ray (figure 6).

3.4 Distributing the computation of the pixels intensity

For reasons of simplicity, we use the illumination model proposed by Whitted [28], an extension to Cook's is straightforward. The contribution of each ray to the intensity of a pixel is computed by each of the processors and transmitted to the host processor which uses it to update the frame memory. To do that, the data structure of a ray message must contain the cumulated product of the specular reflection and transparency coefficients K_s and K_t of the objects intersected by all

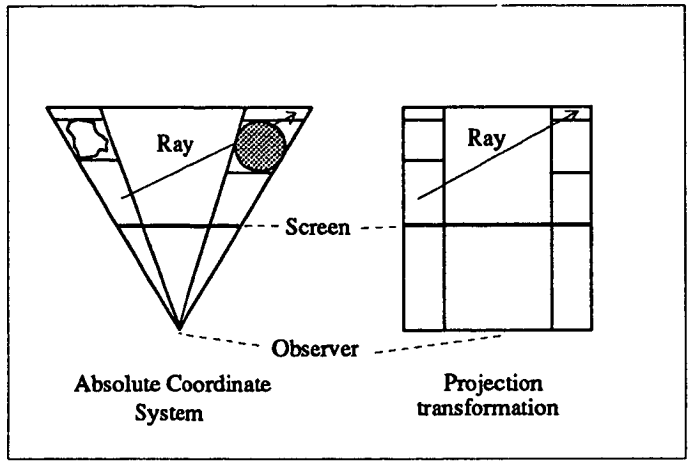

Figure 4: Coordinate system

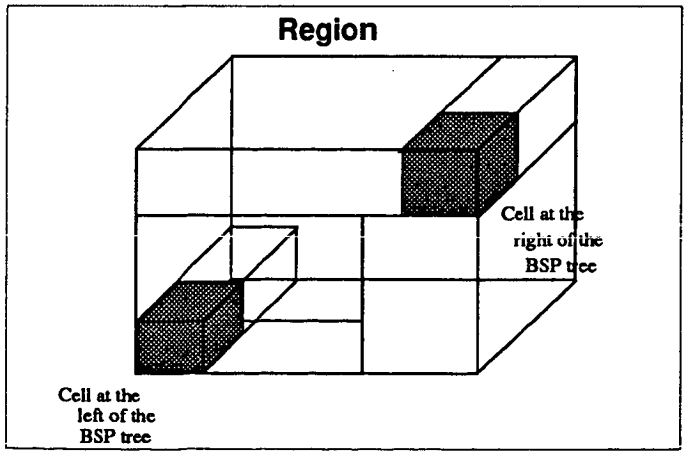

Figure 5: Right-up and left-low cells

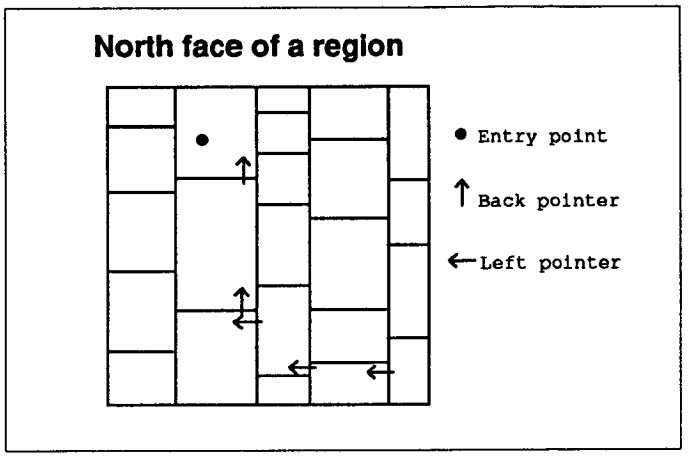

Figure 6: Computing the entry cell

the intermediary rays generated by a primary ray. This avoids the return messages of intensity contribution.

3.5 Interprocessor communication

Our algorithm uses a set of processes. Each process communicates with another one by means of messages. These processes are located on the nodes of the iPSC and on the host processor. The process associated with this latter controls all the input/output operations. Whereas, one process is associated with each node processor and does two tasks :

1. Synthesis task.
2. Communication task.

3.5.1 Process associated with the host processor

This process controls all the input/output operations as for example the reading of the database. It subdivides the space into regions and distributes them among the node processors. It synchronizes the running of the synthesis process of the node processors. After doing that, it waits for the reception of the messages of intensity contribution, coming from the node processors, to update the frame buffer.

3.5.2 Process associated with the Node processors

Figure 7 illustrates the communication between node processes.

- **Synthesis task**

  The node process receives its region and its associated subtree, transmitted by the host processor, and it subdivides it into 3D cells. It then shoots its own primary rays. Some rays may be sent to the processors controlling the regions along their path. This sending

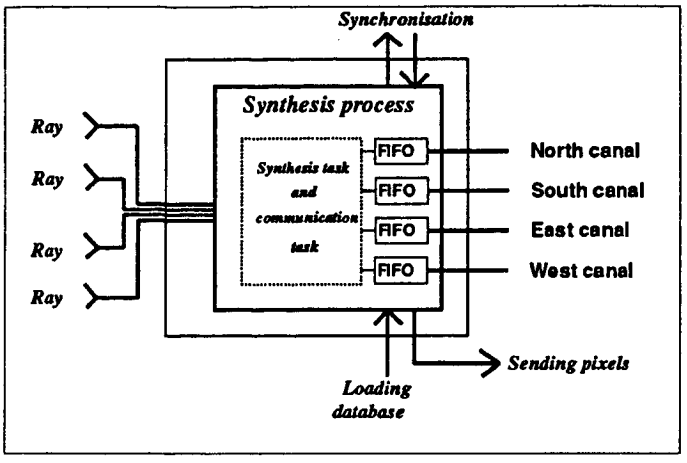

Figure 7: Communication between node processes.

is performed asynchronously by using FIFO queues and asynchronous system call *send()* in order to avoid the locking of the synthesis task. But if FIFO queue is already full, the synthesis task remains locked until there is an entry in the FIFO queue. To avoid the saturation of the FIFO queue, the synthesis task computes in priority the rays sent by the neighbouring processors.

Once, the fraction of the pixel intensity is computed by the synthesis task, this latter stores it in a queue. As soon as this queue becomes full, it is transmitted to the host processor. This saves time since communication between a node and the host processor is time-consuming. The synthesis task is described by the following algorithm:

```
main()
{
  while(image_not_finished()) {

     /* Processing of primary rays */

     generate_primary_ray(r)
     evaluate_ray(r);

     /* Processing of rays coming from */
     /* the neighbouring processors    */

     while(ray_to_read())
        read_ray(r);
        compute_entry_cell(r)
        evaluate_ray(r);
     }
   }
}
```

where :

evaluate_ray(r) processes a ray according to its type. Secondary or light rays may be shot.

compute_entry_cell(r) computes the entry cell according to the entry point in the region.

image_not_finished() this procedure corresponds to the implementation of the termination algorithm.

- **Communication task**

The node process has a second task which manages the four *FIFO* queues used by the synthesis task. There is one queue for each face of a region except for the front and back face. When the synthesis task wants to put a ray in a queue, the communication task is called to try to send the oldest ray message picked in the four FIFO queues in order to avoid the saturation of these queues. FIFO queues can store 128 rays under the message form given by:

```
typedef struct
  {
  type      kr;      /* kind of ray (primary, */
                     /* light or secondary ray */
  point     ori;     /* origin of the ray */
  vecteur   dir;     /* direction of the ray */
  point     end;     /* outgoing point */
  short     depth;   /* depth of the ray */
  short     ipix;    /* pixel coordinate */
  short     jpix;    /*    "            "   */
  short     face;    /* face containing the */
                     /* outgoing point */
  double    maxlum;  /* distance between the origin */
                     /* of the ray and the light source */
  double    coeff;   /* Cumulated product of the Ks, Kt */
  double    att;     /* Attenuation factor */
  } rayon;
```

3.6 Termination algorithm

We have chosen to implement the termination algorithm proposed by Dijkstra [9]. To do that, we configure the iPSC into a virtual ring on which a token moves, whose color may be white or black. At the initialisation phase, processor 0 possesses the token. After having completed a tour round the ring, if the token remains white the termination algorithm is then effective. Otherwise, processor 0 must emit another white token. This latter is not immediately passed from a processor to its neighbour. Indeed, the synthesis process consist of two phases. In the first phase, the primary rays and the received rays are simultaneously processed. The second phase starts when all the primary rays are treated. During this phase, a node processor has to consume only the rays coming from its neighbouring processors. It is in the second phase that the token moves from processor to processor. A processor transmits the token (whatever its color) to its neighbor only if it has nothing to do. When a processor P receives the token, two cases have to be considered:

1. If the token is black, processor P transmits it to the next processor in the ring.

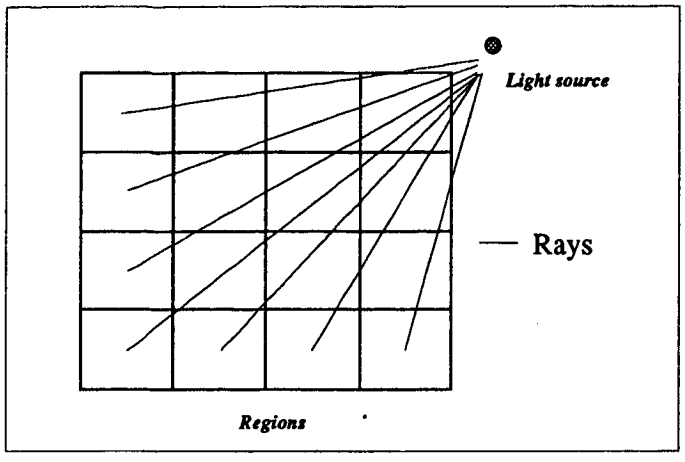

Figure 8: Light rays path.

2. If the token is white, it is transmitted to the next processor in the ring, only if processor P has not sent a message to a previous processor in the ring before receiving the token. Otherwise processor P modifies the color of the token before transmitting it to the next processor in the ring.

Each processor transmits the token to the next processor only if it has finished to compute its primary rays. The goal of this technique is to minimize the number of tokens. First results have shown that only a dozen is enough to detect the termination of our parallel algorithm.

3.7 Load distribution

The first tests have been performed with a scene of one hundred objects and with a resolution of 256 x 256 pixels. Our present parallel algorithm seems slower than Roth's (where the database is duplicated in each processor and the pixels are distributed among all the processors). In fact, the algorithm is very efficient at the beginning of the synthesis phase. After that, its efficiency decreases rapidly. The state to which converges the algorithm is such that 75 percent of the process are locked due to the saturation of the FIFO queues. This means that the load is not uniformly distributed. This load imbalance is mainly dûe to the processing of the light rays which all converge to the light sources, yielding then a lot of messages (figure 8). In the following, we propose a method of performing a roughly uniform load distribution.

3.7.1 Static load distribution

As mentioned previously, dynamic load redistribution [10,22] yields a lot of messages and consequently a lot of computations of ray-boundary intersection. This degrades dramatically the performance of a distributed machine. We think it is more realistic to perform a load distribution staticly, that is before starting the synthesis phase. The proposed redistribution method consists in sub-sampling the image in order to represent a set of coherent rays by only one ray generated by ray-tracing the sub-sampled image. The algorithm consists of two steps. In the first step, the primary rays corresponding to this sub-sampling and all the derived secondary and light (in case of non use of light cones) rays are computed. In the second step, the pyramidal bounding box of

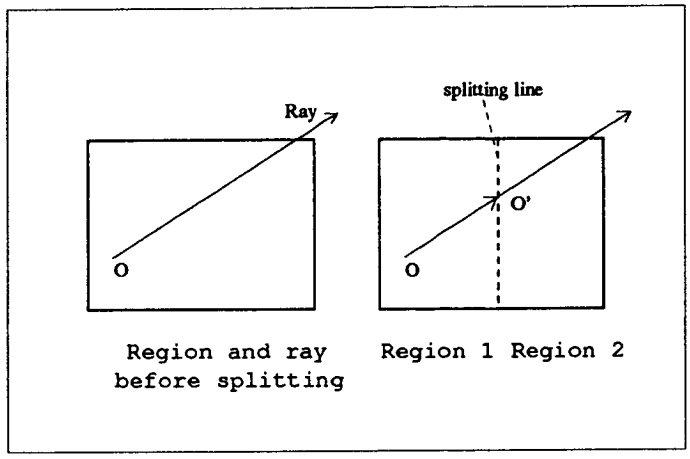

Figure 9: Splitting a ray into two rays.

the scene is adaptively subdivided into regions; each one contains roughly the same number of rays. In fact this subdivision is accomplished by means of a 2D binary space partitioning technique [12] which is performed on the screen plane. Indeed, all the rays are projected in perspective on the screen plane and the 3D regions are represented by their associated pixels areas described in the previous section. During the BSP step, a ray may belong to two adjacent regions created by a BSP splitting line. This ray is then split into two rays (Fig. 9).

The result of the BSP subdivision is firstly a set of rays including the original rays computed in the first step and secondly a set of regions containing roughly the same number of rays. We can now define the load metric. It is only determined by the number of rays to be processed, since objects are supposed to be of the same kind and since ray tracing in a 3D region is nearly independent of the number of objects [2]. Since a region may be adjacent to several others (Fig. 10), we obtain an adjacency graph which represents the interprocessors communications. It is important to suitably map this graph on the hypercube topology in order to minimize the messages routing. An other important point is the choice of a good sub-sampling. This latter is not obvious. For example, it must be regular and must depend on the image resolution. This choice is under investigation.

Another problem arises when a light source lies within the scene. In this case, the processor controlling the region containing it, as well as its neighbours, become rapidly saturated (figure 11). A solution to this problem, is to delete the light ray messages involved by this kind of source, as shown in the following subsection.

3.7.2 Deleting the light ray messages

The aim is to allow each node processor to process locally its light rays. To do that, it must know all the objects potentially intersected by the light rays which originate from its own region. These light rays form a light volume. This latter may be approximated by a cone whose apex is a light source and which contains the spherical bounding box of the relevant regions (figure 12). A small part of the database (which is a subtree in our case) is associated with a cone which is henceforth called light cone. Thus at each region, corresponds as many light cones as light sources. In fact, each light cone is truncated by a plane which subdivides the space into two half-spaces. A leaf of the subtree, associated with a light cone, is such that its spherical bounding box intersects both

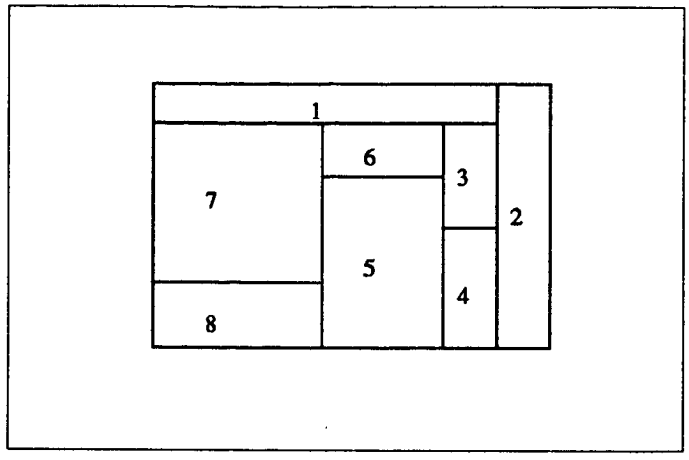

Figure 10: Result of the BSP.

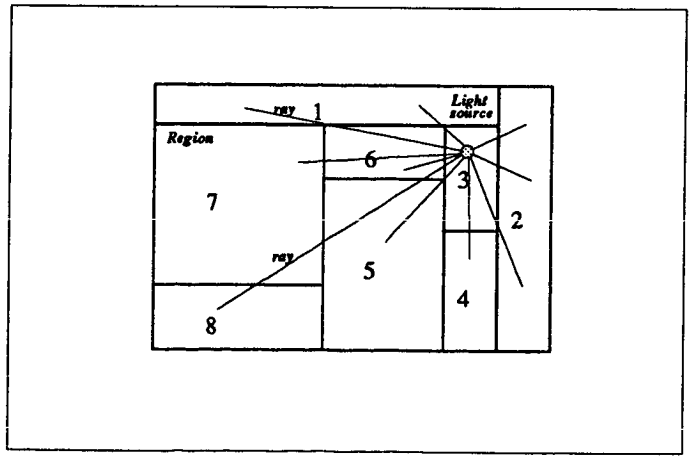

Figure 11: Network congestion due to light source within the scene.

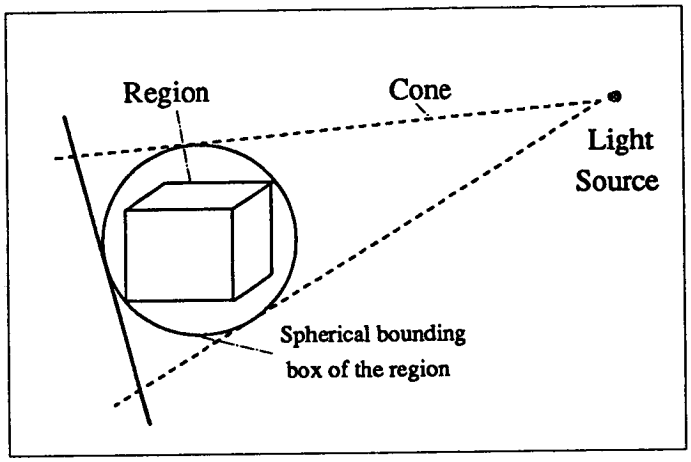

Figure 12: Light cone.

the light cone and the half-space containing the light source. Spherical bounding boxes are used since sphere-cone intersection is easy to compute [1].

An alternative is to use a pyramid (called light pyramid) to fit the light volume more accurately. The construction of this pyramid is described in figure 13.

A coordinate system (S, X_l, Y_l, Z_l) is associated with a light source located at point S. It is named light coordinate system. The Z_l axis passes through the center of gravity C of a region. Let V be a vertex of a region. The three unit vectors of the light coordinate system axes are then:

$$\begin{aligned} X_l &= \frac{\vec{SV} \wedge \vec{SC}}{\| \vec{SV} \wedge \vec{SC} \|} \\ Z_l &= \frac{\vec{SC}}{\| \vec{SC} \|} \\ Y_l &= X_l \wedge Z_l \end{aligned}$$

Let (x_l^i, y_l^i, z_l^i) be the coordinates of the eight vertices i that define a region. The light pyramid is then constructed by five faces:

- One face which is perpendicular to the Z_l axis and which contains the point $P(0, 0, Z)$ such that:

$$Z = \max_{i=1}^{8}(z_l^i)$$

- Two faces making the angles α_1 and β_1 with the (x_l, z_l) plane such that:

$$tan(\alpha_1) = | \max_{i=1}^{8}(\frac{y_l^i}{z_l^i}) |$$

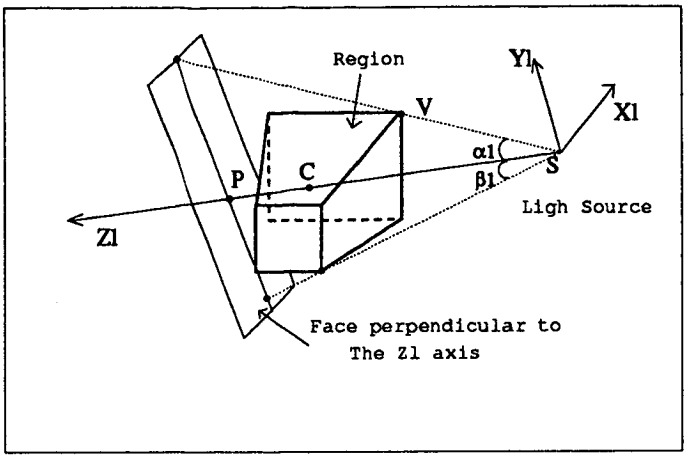

Figure 13: Light pyramid

$$tan(\beta_1) = |\min_{i=1}^{8}(\frac{y_l^i}{z_l^i})|$$

- Two faces making the angles α_2 and β_2 with the (y_l, z_l) plane such that:

$$tan(\alpha_2) = |\max_{i=1}^{8}(\frac{x_l^i}{z_l^i})|$$

$$tan(\beta_2) = |\min_{i=1}^{8}(\frac{x_l^i}{z_l^i})|$$

A leaf of the subtree associated with a light pyramid is such that it's parallelepipedic bounding box intersects the light pyramid.

The light pyramid is more efficient than the light cone because of the shape of the region associated with each processor. Generally, the depth of a region is much greater than its width or its height. Thereby the size of the spherical bounding box is very important. Consequently, the number of objects contains in the light cone would be larger than those contained in a light pyramid.

4 Results

The coding of the algorithm with a static load balancing has been written in C language on an iPSC hypercube available at IRISA. Figure 14 shows the image of the scene which database has been provided by E. Haines [16]. Its characteristics are given by table 1.

Table 2 shows times of synthesis process. Preprocessing time is not showed and take few minutes essentially due to the communication between the host and the nodes. An interesting result is that the estimated number of rays associated with each processor by subsampling the image, is close to the exact one processed by this processor. Figure 15 gives, for each processor, the number of rays

Image	Balls
Number of primitives	92
Number of light sources	3
Maximum depth of ray	5
Kind of objects	sphere parallelepiped
Resolution	128 x 128

Table 1: Characteristics of test image.

Figure 14: Balls

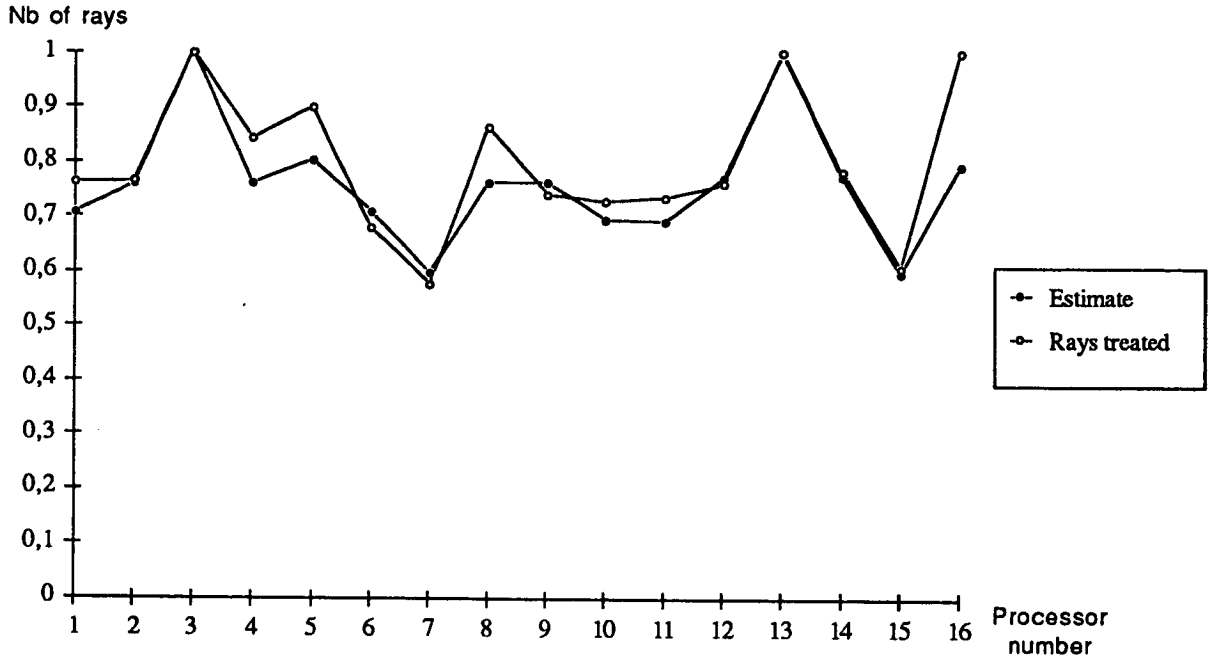

Figure 15: Number of ray estimated and treated by each processor

Processors	Times (s)	Speedup	Efficiency
1	13941	1	1
4	4621	3.02	0.75
16	2262	6.16	0.39
32	1263	11.04	0.35

Table 2: Results

estimated by subsampling and the exact number of rays processed by each of them (the results are normalized). The pixel area is subdivided into a set of 8x8 windows. Subsampling consists then in choosing one pixel in each window. In spite of this, tests have shown that some processors (less than 25 percent) remain less loaded that the other ones. This means that the load metric described previously, must be refined. This refinement is under investigation.

5 Conclusion

We have presented a parallel space tracing algorithm whose implementation on an iPSC hypercube has allowed us to raise all the problems due to a distributed machine. Only one part of the solutions of these problems have been implemented whereas the other part (light pyramids) is being implemented. Our main goal was to reduce considerably the number of messages in order to accelerate the algorithm and to avoid the crucial problem of deadlock. Indeed, primary ray messages have been deleted and light pyramid have been proposed to avoid light ray messages when the light sources are in the scene. The proposed static load distribution yields good results but must be improved by refining the load metric.

References

[1] J. Amanatides. Ray tracing with cones. *Computer Graphics*, 18(3):129–135, July 1984.

[2] B. Arnaldi, T. Priol, and K. Bouatouch. A new space subdivision method for ray tracing csg modelled scenes. *The Visual Computer*, 3(2):98–108, August 1987.

[3] K. Bouatouch, M.O Madani, T. Priol, and B. Arnaldi. A new algorithm of space tracing using a csg model. In *EUROGRAPHICS'87 Conference Proceeding*, pages 65–78, Centre for Mathematics and Computer Science, August 1987.

[4] C. Bouville, R. Brusq, J.L. Dubois, and I. Marchal. Synthèse d'images par lancer de rayons: algorithmes et architecture. In *Premier Colloque Image*, pages 683–696, May 1984.

[5] J.G Cleary, B.M Wyvill, G.M. Birtwistle, and R. Vatti. Multiprocessor ray tracing. *Computer Graphics Forum*, 5(1):3–12, March 1986.

[6] M. Cohen and D. Greenberg. The hemi-cube, a radiosity solution for complex environments. *ACM Computer graphics*, 19(3), 1985.

[7] R.L Cook. Stochastic sampling in computer graphics. In *Siggraph'86 Tutorial*, SIGGRAPH, 1986.

[8] R.L. Cook and K.E. Torrance. A reflectance model for computer graphics. *ACM transactions on graphics*, 1(1):7–24, January 1982.

[9] E.W Dijkstra, W.H.J Feijen, and A.J.M Van Gasteren. Derivation of a termination detection algorithm for distributed computation. *Inf. Proc. Letters*, 16:217–219, June 1983.

[10] M. Dippe and J. Swensen. An adaptive subdivision algorithm and parallel architecture for realistic image synthesis. *Computer Graphics*, 18(3):149–158, July 1984.

[11] M.A.Z. Dippé and E.H Wold. Antialiasing through stochastic sampling. *ACM Computer Graphics*, 19(3), 1985.

[12] H. Fuchs. On visible surface generation by a priori tree structure. In *SIGGRAPH'80 Conference Proceeding*, pages 149–158, July 1980.

[13] A. Fujimoto, T. Tanaka, and K. Iawata. Arts : accelerated ray tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, April 1986.

[14] A. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

[15] J. Goldsmith and J. Salmon. *A Ray Tracing System for the Hypercube.* Technical Report, California Institute of Technology, 1985.

[16] E. Haines. Introduction to ray tracing. SIGGRAPH'87.

[17] A. Roy Hall and Donald P. Greenberg. A testbed for realistic image synthesis. *IEEE Computer Graphics and Applications*, 3(8):10–20, November 1983.

[18] J.T. Kajiya. Anisotropic reflection model. *ACM Computer Graphics*, 19(3), 1985.

[19] J.T. Kajiya. The rendering equation. In *Proceedings of SIGGRAPH'86 in computer graphics*, pages 143–150, SIGGRAPH, August 1987.

[20] M. R. Kaplan. Space-tracing, a constant time ray tracer. In *SIGGRAPH'85 tutorial on the uses of spatial coherence in ray tracing*, 1985.

[21] M.E. Lee, R.A Redner, and S.P. Uselton. Statiscally optimized sampling for distributed ray tracing. *ACM Computer graphics*, 19(3), 1985.

[22] K. Nemoto and T. Omachi. An adaptative subdivision by sliding boundary surfaces for fast ray tracing. *Graphics Interface*, 43–48, 1986.

[23] H. Nishimura, H. Ohno, T. Kawata, I. Shirakawa, and K. Omuira. Links-1: a parallel pipelined multimicrocomputer system for image creation. In *Proc. of the 10th Symp. on Computer Architecture*, pages 387–394, 1983.

[24] S.D Roth. Ray casting for modeling solids. *Computer Graphics and Image Processing*, 18(2):109–144, February 1982.

[25] S. Rubin and T. Whitted. A three-dimensional representation for fast rendering of complex scenes. *Computer Graphics*, 14(3):110–116, July 1980.

[26] R.B Tilove and A.A.G Requicha. Closure of boolean operations on geometric entities. *Computer Aided Design*, 12(5):219–220, September 1980.

[27] K.E. Torrance and E.M. Sparrow. Theory for off-specular reflection from roughened surfaces. *Journal of Optical Society of America*, 57(9):1105–1114, September 1967.

[28] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23:343–349, June 1980.

[29] G. Wyvill and T.L. Kunii. A functional model for constructive solid geometry. *The Visual Computer*, 1(1):3–14, July 1985.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique