Two Algorithms for Fast Ray Tracing of Complex Scenes

Andrea Sanna *Paolo Montuschi* *Massimo Rossi*
Dipartimento di Automatica e Informatica,
Politecnico di Torino, corso Duca degli Abruzzi 24, 10129 Torino (Italy)
E-mail: sanna@censa3.polito.it.

Abstract

In this paper we propose two algorithms for fast ray tracing of complex scenes. We test the scene in a pre-processing step with some rays. In this way, we can obtain some information about the coherence of the rays in the scene and about the space coherence among the objects.
The pre-processing step allows us of reducing the complexity of the data base of objects. We can build a sub-set of items, of the whole data base, which have to be considered for the intersection tests of the primary rays. In the same way, we can obtain a list of objects, for each element of the scene, to be tested with the secondary rays and for the shadow computations.
We will show with two practical examples that our methods provide images of quality practically equal to the one obtained with a ray tracer which uses standard optimizations (bounding box hierarchies) but with rendering times up to 7 times faster.

1 Introduction

Ray tracing is a very powerful technique to obtain photo-realistic images [10]. The main drawback of this method is the high computational time required for the rendering. In particular, it has been proved that up the 95% of the whole computational time may be due to the intersection tests among rays and objects [9].
Several methods have been proposed to speed up the ray tracing. Some of them are based on "exact" approaches such as hierarchies of bounding volumes [8] or space subdivision [4]. Other are based on coherence principles [7]. These methods attempt to "guess" the values of some pixels of the scene, without exactly computing them. The priori knowledge of some relationships can be properly used to find good estimates of pixel groups.
In this paper, we propose two algorithms which attempt to combine two principles of coherence together. We use a pre-processing of the scene in order to find some useful properties and relationships. In this way, we can build efficient data structures to reduce the number of the objects to be considered for the intersection tests. We use some test rays traced in a particular way. We can use the information about the path of the rays (ray coherence) and about the relationships among objects (object coherence) to know a

priori what objects have to considered in the intersection tests.
We will show with two examples that our algorithms are able to render scenes from three to seven times faster than a ray tracer which uses bonding box hierarchies, without affecting the whole quality of the images. With our methods only very little details of scenes are missed and the differences between exact and approximated scenes are almost unnoticeable.
The content of the paper is the following: in section 2 we briefly review the main coherence principles used in computer graphics, in section 3 we describe the proposed methods and, finally, in section 4 we show the results of a comparison between our algorithms and a ray tracer with standard optimizations.

2 Previous work

Several methods have been proposed to speed up ray tracing of complex scenes by using techniques based on coherence principles; a survey paper on the coherence in computer graphics can be found in [7]. It is very difficult to provide a non ambiguous definition of coherence but, in general, the term denotes similarities among items or entities. These methods attempt to guess the values of some pixels, reducing in this way the computational cost of the rendering process. Among all, it is important to remember the following kinds of coherence:

- Object coherence. Object coherence is based on some known relationships either among objects or parts of the same object. These relationships can reduce the complexity of some operations such as clipping, sorting, comparison and so on. Some references on this kind of coherence can be found in [2] and [3].

- Image coherence. It is very similar to the object coherence, in fact, an area on the image plane can be "affected" only by a small sub-set of objects belonging to the scene. Space relationships among objects can be seen as plane relationships on the plane of view [6].

- Ray coherence. Ray coherence is based on the principle that similar rays, i.e., rays with similar origin and direction, may have a very similar behavior. In this way, many ray-object intersection tests can be saved and the rendering process improved [1] [3].

- Area coherence. It is likely that adjacent pixels of a raster image have the same, or similar, intensity values. For instance, this kind of coherence can be employed when a group of pixels belongs either to the same object or the same visible surface [2].

Another strategy widely used to speed up ray tracing process is based on hierarchies of bounding volumes; this method is briefly reviewed because used in our algorithms. The idea is of encapsulating the objects of the scene with bounding volumes of simples shape (spheres, rectangular prisms an so on) for which the intersection tests with rays are very fast. If a ray does not strike a bounding volume the intersection test with object included is not necessary. The best part of the rays do not hit an object, therefore, computational time can be saved; for the rays striking a bounding volume two intersection tests have to be done (a test with the bounding volume and a test with the object) but the test with the bounding volume is very fast. To further improve the intersection test procedure, the bounding volumes can be organized in a hierarchical way. Several bounding volumes can be surrounded with a larger volume; this strategy can be recursively repeated until a hierarchy is built. The top of the hierarchy is first tested, then, the hierarchy is recursively descended if necessary. The recursive process is stopped when: there is not intersection or there is an intersection at the bottom level. In this case, the object at the corresponding leaf has to be tested.

3 The algorithms

3.1 Basic idea

Our goal is to reduce the computational cost of the rendering processing. We attempt to identify a reduced set of objects to be considered for the intersection tests. For instance, if we have a scene with one hundred objects and we can detect that no more that ten objects are visible from the observer for each pixel of scene, then, we are able to reduce the complexity of the rendering process.

We can reduce the complexity of a ray tracer by a pre-processing of the scene. We can trace some rays to identify some relationships among the objects. For instance, we can foresee which objects, of the whole data base, are visible from the observer; this information can speed up the intersection tests for the primary rays. Moreover, we can define a sub-set of objects, for each item of the scene, to be considered for intersection with reflected and refracted rays.

3.2 Steps of the algorithm 1

In this section we describe the steps of the algorithm 1 (in the following comparison tables and figures denoted as algo 1). It can be summarized as follows:

1. A hierarchy of bounding volume is built for all the objects of the scene by using the method described in [5]. With this technique, a bounding box at time is considered. Then, the point where the insertion of the box has minimum cost is searched in the hierarchy. The insertion of a box into the hierarchy may modify the size of the "parent node"; in fact, a parent box has to encapsulate all its son boxes. Therefore, the term cost denotes the growing of volume of the parent box in the hierarchy after

the insertion of a son box. We have chosen this method because it is very efficient and simple. Of course, also other techniques of space subdivision [4] or hierarchies of bounding volumes [8] could be used.

2. The scene is pre-processed by tracing some rays. We will describe more in the details the pattern of the pre-processing grid for the rays in section 3.4. We split the screen into cells of size depending of the resolution of the image. The pre-processing rays are traced for the corner pixels of these cells and for the central pixel. Each object hit from a ray is considered and its bounding box is used to build a hierarchy as described in [5]. In this way, we attempt to consider for the primary rays only the objects visible from the observer, i.e., only a reduced set of the whole data base.

3. Some new lists are build for each object of the new hierarchy. A list of neighboring objects is obtained by considering reflections and refraction of the pre-processing rays. A list for each light source is obtained by considering the objects tested for the shadow computation of the pre-processing rays. In this way, we are able to reduce the number of objects considered for the intersection tests for secondary rays and shadow computations. Moreover, if in the occlusion tests of the pre-processing rays we discover that an object "can not see" a light source, we remove this light source from the list for that object. In this way, the complexity of the shadow computation is further reduced. With this strategy we exploit both the space coherence among the objects and ray coherence.

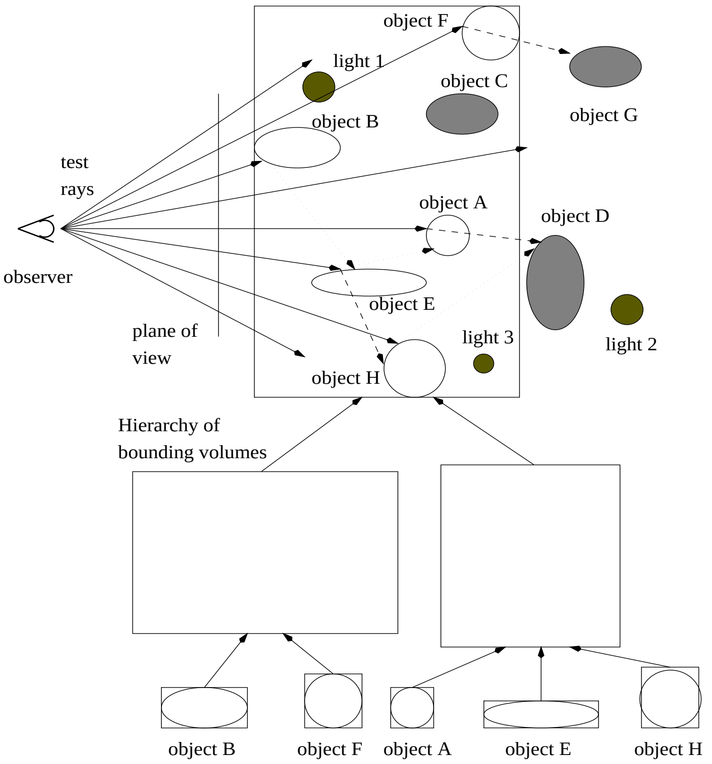
4. In the final step, ray tracing is performed for each pixel not traced in the pre-processing step. Of course, the values of the pre-processed pixels have already been computed in the previous steps of the algorithm.

A pseudocode of pre-processing step is shown in Fig. 1. Let us consider the example in 2D shown in Fig. 2. In Fig. 2 is shown the pre-processing step; test rays are traced from the observer and strike some objects of the scene (the empty objects). These objects are considered to build a new hierarchy of bounding volumes; the other elements will not be considered for the intersection tests with the primary rays in the next phase of the rendering process. For each object is shown the list of neighbors that have to be considered for the intersection tests with the secondary rays (in the example just one reflection/refraction is considered). Moreover, for each object and for each light are shown the elements have to be tested for the shadows.

```
Test_Ray_Algo1(Start_Ray=NULL;Start_Obj=Observer)
{
    for(ray=1;ray<=n_ray_test;ray++)
    {
         obj=Trace_Ray(ray);
        if(obj!=NULL)
        {
            if(Start_Obj==Observer)
            {
                Add_Hierarchy(obj);
                for(light=1;light<=n_light;light++);
                {
                    Detect_Object_Shadow(light);
                    Create_List_Object(light);
                }
            }
            else
                Add_list(obj,Start_Obj);
            if(reflection<=max_reflex)
            {
                Test_Ray_Algo1(reflex,obj);
                Test_Ray_Algo1(trasm,obj);
            }
        }

        else
            return;
    }
    return;
}
```

Figure 1: Pseudocode of Algorithm 1 pre-processing step.



Object	Neigh.
A	D
B	E
E	A H
F	G
H	D

Object	light 1	light 2	light 3
A	A	-	A
B	B	-	B E
E	B E	-	E
F	F	F	-
H	B E H	-	H

Figure 2: Example in 2D.

3.3 Steps of the algorithm 2

Algorithm 2 (in the following denoted as algo 2) is quite similar to algorithm 1. The only difference concerns the objects visible from the observer, i.e., the step 2 of the algorithm. Instead of building a hierarchy of bounding boxes as in the step 2 of algorithm 1, we keep a list of objects to be tested for each cell of the image. In this way, for each cell defined by the pre-processing rays just a few objects have to be tested. If an object is hit into a cell, it is assigned also to the neighboring cells for an interval having center in the considered cell and length depending both by the resolution of the image and the pre-processing step. This strategy attempts to avoid that an object, which should affect two or more neighboring cells is rendered in an incorrectly way.

3.4 Shape of the grid

The pattern of the grid of the pre-processing rays is very important to improve the performances of our algorithms. We have chosen a regular grid with square cells. The pre-processing step depends on the resolution of the image. With a resolution of 320x200 pixels we adopt a step of 6 pixels, while with a resolution of 640x480 pixels we use step 12 and for 1024x768 images we use 18.

It is important to note that the number of the pre-processing rays is constant and independent of the resolution. In fact, the projection on the plane of view of an object is always the same and does not change with the resolution. Therefore, the same "precision" is always obtained with a constant number of rays of test.

To avoid that very thin objects, parallel with the co-ordinate axes, were completely lost in the pre-processing step, we have adopted a fifth pixel per cell. This pixel is placed in the center of the cell; in this way, the probability that an object is not detected in the pre-processing stage decreases. In Fig. 3 the patterns of the cells with and without central pixel are shown.

We Assume to consider only convex objects (non convex objects can often be obtained as union of convex objects) and then we try to find the optimal shape of a cell of the pre-processing grid. If we had assumed a step of n pixels without central pixel, an object affecting an area of a cell of $(n+1)^2-4$ pixels would not be hit by the test rays. Therefore, a large detail of the scene could be lost. The maximum convex area not covered by the test rays into a cell with central pixel is large $\frac{n}{2} * (n+1) - 2$ pixels. Therefore, the probability that an object is missing with cells having a central pixel is halved.

4 Comparison results

We have compared our two algorithms with a ray tracer which uses bounding box hierarchies (algo 0). This ray tracer adopts the method proposed in [5] to build a hierarchy

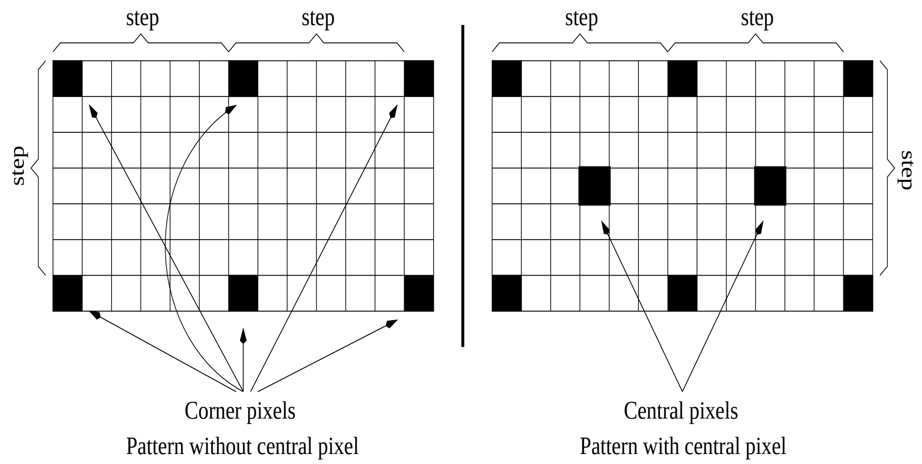


Figure 3: Pattern of the grid

of bounding volumes and speed up the rendering process. All our ray tracers adopt the Phong's light model [10] and process the rays up to the third reflection (refraction). Moreover, we have compared the proposed methods with a ray tracer adopting a principle of area coherence (algo 0x4). This method splits the image into cells of 3x3 pixels (cells overlap in corners) and computes only the pixels at the corners of the cells. The other pixels are obtained as interpolation of the exactly calculated values. With this method, the rendering times is expected to be reduced by a factor four, since just one pixel out of four is exactly computed. The principle of area coherence can be also applied to our algorithms (algo 1x4 and algo 2x4). Two examples have been considered:

- Example 1: a virtual city built with 560 polygons and four light sources.
- Example 2: a scene with 120 random spheres, three planes and eight light sources.

Each image has been rendered with a resolution of 1024x768 pixels in true color on a PC with a P6 133 MHz processor. Rendering times in seconds are shown in table 1. It is important to note that algo 1 is at least 3 times faster than the ray tracer with standard optimizations, while algo 2 is about 7 times faster. The algorithm exploiting the area coherence (algo 0x4) is about 4 times faster than algo 0 but two important things have to be remembered: I) the quality of the images obtained with algo 0x4 is worse than the one obtained with our algorithms. In fact, algo 0x4 works on an image with a resolution lower than the original. We lose just very little details of the scenes employing our methods and we consider the images at the full resolution. The lost details are almost unnoticeable and only an accurate exam of the images shows the differences with the scenes traced with the standard ray tracer. Moreover, if also our algorithms adopt area coherence principle the examples are rendered up to 22 times faster than the standard ray tracer. II) The area coherence principle works like a low-pass filter. In fact, the interpolated pixels are computed as an average between exactly computed values. Therefore, the scenes look like darker than the exact ones. This could be a big drawback in the case the method

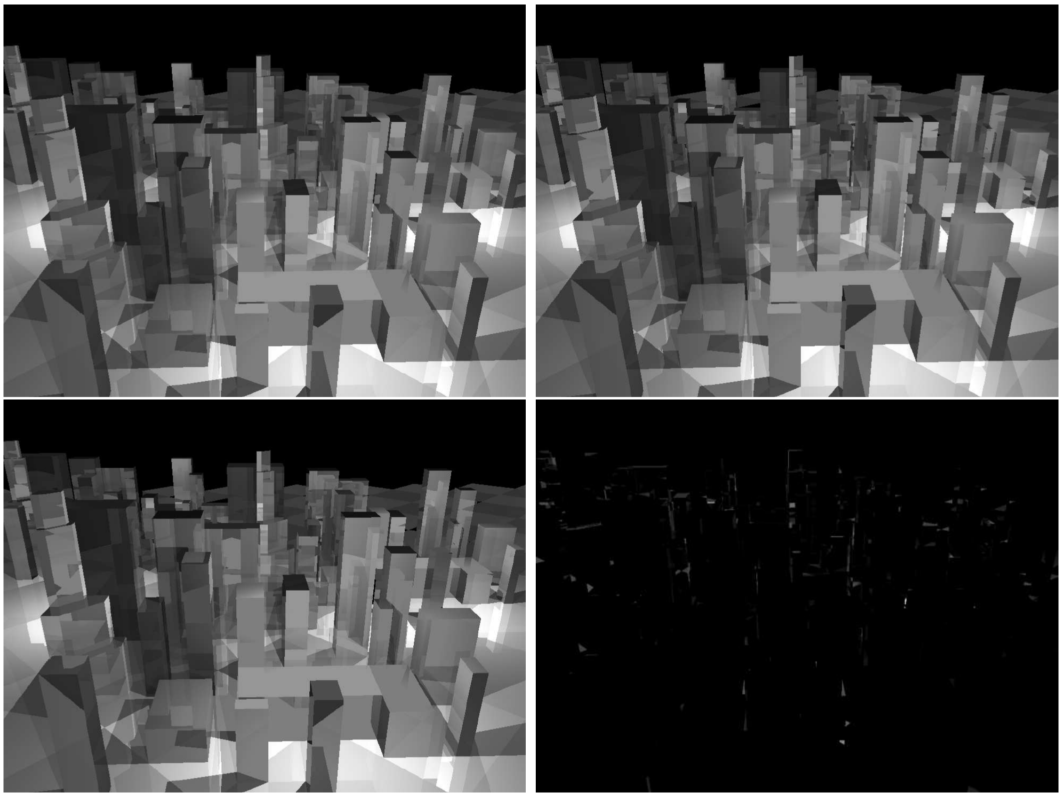


Figure 4: Example 1, 560 polygons.

Table 1: Computation times in seconds

Example	algo 0	algo 0x4	algo 1	algo 1x4	algo 2	algo 2x4
example 1	763	193	251	68	114	34
example 2	423	107	120	33	54	18

were used to render frames of an animation. In this case, the differences among exactly generated frames and approximated frames could be easily noted from an observer. In Fig. 4 and in Fig. 5 the images obtained are shown. The upper left images have been obtained by using the standard ray tracer (algo 0), the upper right images by using the algo 1, while the lower left images by using algo 2. The lower right images represent the differences between the images calculated with algo 0 and algo 2 (the faster of the two proposed algorithms but also the more approximated one) which have been enhanced for easier viewing. A slight drawback of our methods is the memory requirement. Algo 1 and algo 2 require two times the memory used by algo 0 for example 1 (556464 bytes versus 966672 bytes) and three times the memory needed to algo 0 for the example 2 (152820 bytes versus 407636 bytes). Anyway, it is important to stress that the memory required is quite small, at least for a wide range of applications and synthetic scenes, and can not be considered critical for nowadays workstations or PCs.

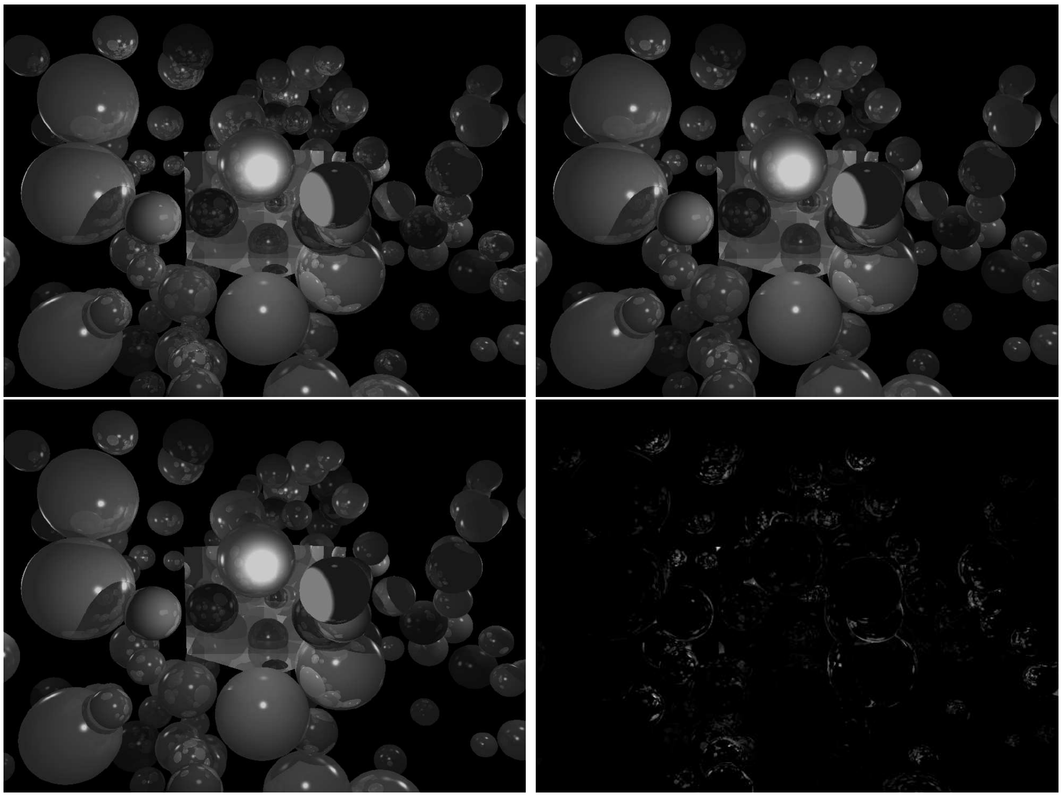


Figure 5: Example 2, 120 spheres and 3 planes.

4.1 Extreme cases

The accuracy of the images produced with proposed algorithms depends on the configuration of the scenes. An extreme case is when either the objects are very thin and are aligned with the axes or the objects are built with very small elements; in this situation, is likely that important details will be lost. To avoid this drawback it is possible to adopt an irregular pattern for the test ray grid, in this way, the probability that regular objects are lost might be reduced, in particular for the first algorithm. The worst case for the memory occupation is when each test ray hits a different object; in this situation the maximum number of lists is built. Moreover, the number of the items of the list is maximum when each reflected/refracted ray strikes an object which has to be added to the neighbor list of the element form which the secondary ray is started. Anyway, the memory requirement is strongly related to the number of test rays (that should be a small sub-set of whole number of traced rays), therefore, by controlling the pre-processing step is possible to limit the occupation memory. The efficiency of our methods mainly depends on two factors: I) resolution of the image; the number of test rays, which require a high computational time, may be not negligible for low resolution images and it may heavily affects the whole rendering time. II) Number of the objects; the algorithms provide the best results when the scenes are densely populated, in fact, in the cases the ratio between number of elements and number of "visible" objects is high and many tests may be saved.

5 Concluding remarks

In this paper we propose two efficient algorithms to speed up the ray tracing of complex scenes. Our methodologies are based on the idea that the complexity of the intersection tests among objects and rays can be reduced with a pre-processing step. In the pre-processing step the images are split into cells of size depending on the resolution of the scene. For each cell, the rays for corner and the central pixels are "exactly" traced. These test rays allow us to build efficient data structures. In this way, we can reduce the data base of the objects to be considered for the intersection with: primary rays, secondary rays and occlusion tests. We have shown that our algorithms can be, for the considered scenes, 3 and 7 times faster than a ray tracer with standard optimizations, respectively and the differences between the images obtained with our method and the exact ones are almost unnoticeable.

References

[1] J. Arvo and D. Kirk, "Fast Ray Tracing by Ray Classification", Computer Graphics 21(4), July 1987, pp. 196-205.

[2] J. Foley, A. Van Dam, S. Feiner and J. Hughes, "Computer Graphics Principles and Practice", Addison-Wesley, 1990.

[3] A. S. Glassner, "An Introduction to Ray Tracing", Academic Press Ltd. 1989.

[4] A. S. Glassner, "Space Subdivision for Fast Ray Tracing", IEEE Computer Graphics and Applications, Oct. 1984, pp. 15-22.

[5] J. Goldsmith and J. Salomon, "Automatic Creation of Object Hierarchies for Ray Tracing", IEEE Computer Graphics & Application, May 1987, pp. 14-20.

[6] S. A. Green and D. J. Paddon, "Exploiting coherence for multiprocessor ray tracing", IEEE Computer Graphics and Applications, Nov. 1989, pp. 12-26.

[7] E. Gröller and W. Purgathofer, "Coherence in Computer Graphics", Institute for Computer Graphics, Technical University Vienna, Internal Report TR-186-2-95-04.

[8] T. L. Kay and J. T. Kajiya, "Ray Tracing Complex Scenes", ACM Computer Graphics, Vol. 20, No. 4, pp. 269-277.

[9] S. M. Rubin and T. Whitted, "A Three-dimensional Representation for Fast Rendering of Complex Scenes", ACM Computer Graphics, Vol. 14, No. 3, pp. 110-116.

[10] D. Thalmann and N. Magnenat-Thalmann, "Image Synthesis: theory and practice", Springer-Verlag Tokyo, 1987.