Volume xx (*200y*), *Number 3 pp. 1–7*

A Formal Framework for Analysis and Comparison of Ray Casting Acceleration Techniques

J. Revelles, C. Ureña, M. Lastra, R. Montes

Dpt. Lenguajes y Sistemas Informáticos, E.T.S.I. Informática, University of Granada, Spain
e-mail: [jrevelle,almagro,mlastral,rosana]@ugr.es
URL: http://giig.ugr.es

Abstract

The ray-scene intersection test is the most costly process when a scene is rendered. This process may be improved using any strategy to be able to speed-up it. Generally, any strategy used is based on the building of a spatial indexing in the scene domain or in the rays domain. However, there is no theory to formalize these techniques. In this paper, an acceleration techniques formalization is proposed. This formalization allows an optimizer to be specified according to the spatial index used. Furthermore, a formalization of optimizer composition is presented. Finally, we present an expression which allows to compare several optimizers, and select the one with best performances. This formalization is based on the graphics objects theory and claims to be a generalization to all optimizers which use spatial indexing.

Keywords: Graphics object theory, spatial indexing, ray casting, acceleration techniques.

1. Introduction

Usually the programs based on ray tracing include acceleration techniques in order to improve the ray-scene intersection test. Several works have been proposed on this subject. Glassner[10], Fujimoto and Iwata[8] presented techniques based on both uniform and non-uniform spatial subdivision, using a regular 3D grid and an octree respectively. Both strategies are based on subdivision of a 3D region which includes the whole scene. In these techniques it is necessary to design an algorithm for computing the intersection between a ray and the spatial structure.

Arvo and Kirk[3] proposed another technique called "ray classification". It consists of partitioning the five-dimensional space of rays into small regions which are encoded as 5D hypercubes. Each hypercube is associated with a list of scene objects that are totally or partially inside.

Haines and Greenberg[13] proposed a strategy to improve the shadow test, via the light buffer. This technique is only used in ray tracing for spot or directional light sources.

Based on the above techniques, some effort has been devoted to develop new algorithms to traverse spatial indexing schemes. In order to improve the traversing process using a regular 3D grid, several works were proposed[1, 8, 7]. Other papers were presented describing improvements to the basic octree traversal algorithm [5, 7, 9, 15, 17, 20].

There are also available techniques which employ other schemes such as the hierarchical bounding volumes[16, 12] and binary trees[22].

As it has been mentioned above, a lot of effort has been employed in the development of efficient solutions to the problem of ray-scene intersection test. However, there is no theory that formalizes the behavior of a generic optimizer.

In this paper we propose an abstract model of a generic optimizer as a function which selects a set of candidate objects for a given scene and a given ray. We also propose a model of optimizer based on spatial indexing and two strategies to obtain new optimizers by composing other optimizers.

The paper is organized as follows: In section 2 several concepts, definitions, and notations are given. In section 3, an abstract characterization of a strict optimizer and an optimizer are presented. Section 4 presents several compositions

of optimizers, and the recursive optimizers are presented. In section 5, an efficiency measurement is proposed. Finally we include our conclusions and planned future work.

2. Concepts, Definitions and Notations

We focus on optimizers that improve the ray-object intersection test using a spatial index scheme. This scheme is a representation of the spatial distribution of scene objects. It is composed of a set of voxels, which will be called here *volumetric objects*. Each volumetric object contains information of a subset of scene objects, which are contained in it. The ray-object intersection test performs the test with each volumetric object and returns a list of scene objects.

To formalize the optimizer behavior, we use the graphic objects theory [23, 24].

2.1. Graphic Object Theory

A graphic object o is a pair (μ,α), in which: μ is a function called presence function defined as $\mu:\mathbb{R}^3\rightarrow P$, where P is a presence domain, and α is a function called aspect function with domain in $\mathbb{R}^3$ and range in T, where T is called aspect domain.

We adopt as presence domain a subset of $\mathbb{R}$, more concretely, $P=\{0,1\}\subseteq\mathbb{R}$. Using this presence domain, a graphic object is equivalent to an arbitrary set of points in space.

The aspect domain T is not defined because it is not necessary in the current framework. We only need the spatial region occupied by a graphic object.

The set of all graphic objects is denoted by $\mathcal{O}$.

For each graphic object $o\in\mathcal{O}$, we define the spatial region $\mathrm{Vol}(o)\subseteq\mathbb{R}^3$ as the set of all points $p\in\mathbb{R}^3$ such that $\mu(p)=1$. Formally it is:

$$\forall o\in\mathcal{O},\ \mathrm{Vol}(o)=\{p\in\mathbb{R}^3\ \mid\ \mu(p)=1\} \tag{1}$$

where μ is the presence function of o

The null or empty graphic object, denoted by ϕ, is the unique graphic object satisfying the following property

$$\forall p\in\mathbb{R}^3\ \ \mu(p)=0$$

where $\phi=(\mu,\alpha)$. This graphic object fulfils $\mathrm{Vol}(\phi)=\Phi$ (Φ denotes the empty set of points).

The presence domain (that is, the set $P=\{0,1\}$), satisfies the properties of a boolean algebra. This presence domain includes operators such as the *union* ($\vee$), the *intersection* ($\wedge$), and the *complement* ($\sim$). For any pair of values $a,b\in\{0,1\}$ the following expressions are fulfilled:

$$\begin{aligned} a\vee b &= \mathrm{Max}(a,b)\\ a\wedge b &= \mathrm{Min}(a,b)\\ \sim a &= 1-a \end{aligned}$$

where Min and Max have the usual meaning. Thereupon, the set of graphic objects $\mathcal{O}$ inherits this boolean algebra structure.

The union of two graphic objects, $o_1=(\mu_1,\alpha_1)$ and $o_2=(\mu_2,\alpha_2)$, is an object $o=(\mu,\alpha)$ whose presence function μ satisfies the following expression:

$$\forall p\in\mathbb{R}^3\ \ \mu(p)=\mu_1(p)\vee\mu_2(p) \tag{2}$$

The graphic object o can be written as $o_1\cup o_2$.

The intersection of two graphic objects, $o_1=(\mu_1,\alpha_1)$ and $o_2=(\mu_2,\alpha_2)$, is an object $o=(\mu,\alpha)$ whose presence function μ satisfies the following expression:

$$\forall p\in\mathbb{R}^3\ \ \mu(p)=\mu_1(p)\wedge\mu_2(p) \tag{3}$$

The graphic object o can be written as $o_1\cap o_2$.

For any graphic object $o_1=(\mu_1,\phi_1)$ its complement is a graphic object $o=(\mu,\alpha)$ such that:

$$\forall p\in\mathbb{R}^3\ \ \mu(p)=\sim\mu_1(p) \tag{4}$$

The graphic object o will be written as $\sim o_1$

3. Optimizer Abstract Characterization

3.1. Rays

When a optimizer is used into a rendering system, its behavior can be understood as a function which returns a subset of candidate objects for a given ray and a scene. The number of returned scene objects must be lesser than the number of objects in the scene, in order to reduce the number of ray-object intersection tests.

We can define a ray as a graphic object. A ray r is a graphic object (μ,α) such that exists a unique point $q\in\mathbb{R}^3$, and a unique direction vector $u\in\nabla$ such that:

$$\forall\, p\in\mathbb{R}^3\quad \mu(p)\ =\ \begin{cases} 1 & \text{if } \exists t\in\mathbb{R}^+\ \mid\ p=q+tu\\ 0 & \text{otherwise}\end{cases} \tag{5}$$

where $\mathbb{R}^+\subseteq\mathbb{R}$ is the subset of real values strictly greater than zero, and ∇ is the set of unit length vectors in $\mathbb{R}^3$. The point q is the *origin* of the ray, and the vector u is the *direction* of the ray. Every ray has associated a unique origin and a unique directional vector.

From the above definition, we deduce that the volume of a ray is an infinite half-line in $\mathbb{R}^3$. The set of all rays is denoted by $\mathcal{R}$.

A ray and a graphic object may have some points in common, that is, they may *intersect*. When this happens, we can measure the distance from the origin to the nearest common point, and this will be a positive real value. When no intersection occurs, we say that this distance is *infinite*.

In order to formalize this concept, we define the set $\mathbb{R}^*$ as $\mathbb{R}^*=\mathbb{R}\cup\{\infty\}$, where ∞ is any element that it is not included in $\mathbb{R}$. This value is used to denote a infinity distance. By definition, any value $x\in\mathbb{R}$ holds $x<\infty$.

3.2. Intersecting Rays and Objects

Let $r \in \mathcal{R}$, be a ray, and let $o \in \mathcal{O}$ be a graphic object, we define $S(r,o)$ as follows:

$$S(r,o) \;=\; \{\, t \in \mathbb{R} \;\mid\; \mu(q+tu) = 1 \,\} \tag{6}$$

where μ is the presence function of o, q is the origin of r, and u is the direction vector of r. When $S(r,o) \neq \Phi$, an intersection occurs between the ray and this graphic object. When $S(r,o) = \Phi$ no intersection occurs.

Function S returns the set of distances from the origin to all points in the ray which belongs also to the volume of the object. In fact, we only need the lowest one of these real values.

We define the function I with the same domain of S and values in $\mathbb{R}^*$. For each $r \in \mathbb{R}$, and $o \in \mathcal{O}$ it holds:

$$I(r,o) \;=\; \begin{cases} \inf(S(r,o)) & \text{if } S(r,o) \neq \Phi \\ \infty & \text{if } S(r,o) = \Phi \end{cases} \tag{7}$$

where *inf* denotes the *infimum* of a set of real values, which is always defined even for graphics object whose volume is not a closed region.

The main interest of the above definitions consists of determining which graphic objects in a given scene are intersected by a given ray.

In what follows, we will use the symbol $\mathcal{S}$ to mean the set of all scenes.

3.3. Objects Intersected by a Ray

Let $r \in \mathcal{R}$ be a ray, and let $s \in \mathcal{S}$ be a scene, we define $C(r,s)$ as the set of graphic objects intersected by r, as follows:

$$C(r,s) \;=\; \{\, o \in s \;\mid\; I(r,o) \neq \infty \,\} \tag{8}$$

$C(r,s)$ will contain the graphic objects in s intersected by r. Therefore, the condition $C(r,s) \subseteq s$ holds.

We also want to know the nearest intersected graphic objects with respect to the ray origin. Let $r \in \mathcal{R}$ be a ray, and let $s \in \mathcal{S}$ be a scene, we define $C_n(r,s)$ as the set of nearest graphic objects intersected by r as follows:

$$\begin{aligned} C_n(r,s) \;=\; & \{o \in s \;\mid\; I(r,o) \neq \infty \\ & \wedge\; \nexists o' \in s \;\mid\; I(r,o') < I(r,o)\} \end{aligned} \tag{9}$$

The expression $C_n(r,s) \subseteq s$ is also satisfied.

3.4. Strict Optimizer

An optimizer reduces the number of candidate objects for the intersection test. Obviously when a reduced set of scene objects is obtained, we get an improvement in terms of execution time.

Let A be a function with domain in $\mathcal{R} \times \mathcal{S}$ and values in $\mathcal{S}$. A is a strict optimizer if and only if it fulfils the following condition:

$$\forall r \in \mathcal{R},\ \forall s \in \mathcal{S},\quad C(r,s) \;\subseteq\; A(r,s) \;\subseteq\; s \tag{10}$$

In other words, a strict optimizer selects a subset of the scene objects. The optimizer A yields a set of objects intersected by a ray, and possibly, other objects which are not intersected. The best optimizer is one which holds $C(r,s) = A(r,s)$, whereas the worse optimizer is one which always holds $A(r,s) = s$, that is, it always yields the whole scene.

3.5. Optimizer

There are applications where we only need the nearest object intersected by a ray (or the nearest objects, because it may happen that there are more than one at minimum intersection distance). In order to model this requirement we introduce the definition of an optimizer. An optimizer is a function of the same class that a strict optimizer. However the condition we impose to the set of returned objects is weaker, and thus the class of optimizers contains the class of strict optimizers.

Let A^+ be a function with domain in $\mathcal{R} \times \mathcal{S}$ and values in $\mathcal{S}$. A^+ is an optimizer if and only if fulfils the following condition:

$$\forall r \in \mathcal{R},\ \forall s \in \mathcal{S},\quad C_n(r,s) \;\subseteq\; A^+(r,s) \;\subseteq\; s \tag{11}$$

It is easy to prove that any strict optimizer is an optimizer by using the relation $C_n(r,s) \subseteq C(r,s)$ which always holds.

3.6. Spatial Representation

A spatial representation, from now on SR, is a set of graphic objects. These graphic objects will be called *volumetric objects* or *voxels*. The set of all possible spatial representations will be called $\mathcal{E}$. When the only difference between two spatial representations is consists in their aspect functions, we consider both spatial representations equivalent.

Let m be a function with domain in $\mathcal{S}$ and values in $\mathcal{E}$, this function m is a *spatial indexing method* (from now on SIM) if and only if for any given scene $s = \{o_1, o_2, \ldots, o_n\}$ and any given SR $e = \{v_1, v_2, \ldots, v_n\}$ the following equality is satisfied:

$$\bigcup_{i=1}^{n} o_i \;\subseteq\; \bigcup_{j=1}^{m} v_j \tag{12}$$

This set of graphic objects is usually simpler than the original scene, in the sense that the ray object intersection test can be done faster for volumetric objects than for original scene objects. This property is essential for ray casting speed-up, because we can intersect the ray with volumetric objects and discard the scene objects which are included in volumetric objects not intersected by the ray.

In order to determine the ray-scene intersection test, a

function to obtain the intersection between a ray and a SR must be defined.

Let $r \in \mathcal{R}$ be a ray, let $s \in \mathcal{S}$ be a scene, and let $e \in \mathcal{E}$ be a SR, we consider that when an intersection occurs between an object $o \in s$ and a volumetric object, this volumetric object is also intersected by the ray r.

This set is noted as $\Psi(r,s,e)$. This set is more formally defined as follows:

$$\Psi(r,s,e) \;=\; \{\, o \in s \mid \exists v \in e \;/\; v \cap o \neq \phi \;\wedge\; v \cap r \neq \phi \,\} \quad (13)$$

Obviously, $\Psi(r,s,e) \subseteq s$ is always satisfied.

3.7. Optimizer Based On Spatial Indexing

There are many different classes of optimizers. Our attention will be focused on a sub-type or category. This sub-type will be called *optimizers based on spatial indexing*.

Let A be an optimizer. A is an optimizer based on spatial indexing if and only if the following property is fulfilled:

$$\exists!\, m \in \mathcal{M} \;/\; \forall s \in \mathcal{S}, r \in \mathcal{R},\; A(r,s) \;=\; \Psi(r,s,m(s)) \quad (14)$$

Note that for each optimizer based on spatial indexing, there is a unique SIM associated to it (as can be deduced from the above condition).

When an optimizer of this category is implemented, one SIM must be implemented as well. That is, the necessary algorithm to build $m(s)$ must be designed and implemented. Normally, a data structure residing in memory for the SR $m(s)$ must be created.

After that it is possible to process a wide set of rays. For each one ray we must compute which voxels are intersected. From this set of voxels we obtain the set of objects intersecting them. The function Ψ models this algorithm.

4. Composing Optimizers

When an optimizer is used the main goal is to obtain an efficient SR. That is, for a given scene, an optimizer must be selected having into account the objects distribution in the scene. Due to scene complexity, it is not always easy to select the most appropriated optimizer. In this case, it would be interesting to make a partition of the scene. Each partition can be processed by using a different optimizer. In short, we have several optimizers applied to one single scene.

This problem was called by Glassner as *the problem of a teapot inside a stadium*. That is, a very complex and relatively small set of objects inside a very simple and big one. In these cases, the available spatial representations were not as fast as expected. A possible solution was to consider some strategy to compose two o more different spatial representations, as was pointed out in[11] as future efforts.

The main goal is to determine which optimizers are appropriated to use for a given scene[4, 2].

In cases for which it is not easy to find the optimizer which has the best performances, we propose two ways to compose optimizers:

- *Sequential*: This is very useful when, for a given scene, several optimizers will have better performances than a unique optimizer. From an initial SIM, the SR is built. In those voxels with a relatively great number of objects, a secondary SR is applied.
- *Parallel*: We can use this when, for a given scene, there is uncertainty or doubt to determine the best optimizer. The main goal is to execute in parallel or concurrently several optimizers (simple or composed sequentially). For each ray and each optimizer, a subset of intersected objects are returned. The final result is the intersection of all objects subset.

In the following sections, the above described optimizers are formally defined.

4.1. Sequential Composition

As it was mentioned in the previous section, the main goal is to separate the complex scene into simpler subscenes. With this purpose in mind, a new optimizer may be applied for each volumetric object $v \in m(s)$.

We will define the sequential composition as follows:

Let A_1 be an optimizer based on spatial indexing, let m be associated to A_1, and let A_2 be any optimizer. For any ray $r \in \mathcal{R}$ and any scene $s \in \mathcal{S}$, the result obtained when m is applied to s is a spatial representation $m(s)$ including n volumetric objects or voxels as follows:

$$m(s) \;=\; \{v_1, v_2, \ldots, v_n\} \quad (15)$$

In these conditions, we say that A_σ is the sequential composition of A_1 and A_2 (noted as $A_\sigma = A_1 \mid A_2$) if and only if the A_σ is an optimizer which holds the following condition:

$$A_\sigma(r,s) \;=\; \bigcup_{i=1}^{n} A_2(r, e_i) \quad (16)$$

where e_i is the subscene of s including objects which intersect v_i, that is:

$$e_i \;=\; \{o \in s \mid o \cap v_i \neq \phi\} \quad (17)$$

It is easy to prove that the sequential composition is not commutative nor associative in general.

However, when A_2 is also an optimizer based on spatial indexing then the following two conditions hold:

- $A_1 \mid A_2$ is an optimizer based on spatial indexing.
- For any optimizer A_3, it holds that

$$A_1 \mid (A_2 \mid A_3) \;=\; (A_1 \mid A_2) \mid A_3 \quad (18)$$

This formalism can be used to obtain formal models of several optimizers previously proposed by several authors [18, 19, 14, 4, 2, 21, 6].

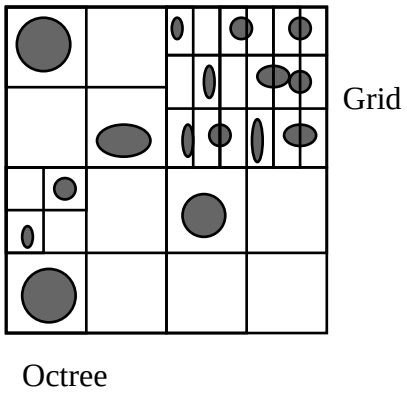

Figure 1: *An example of sequential composition.*

One example for a sequential composition is shown at figure 1. This figure shows us a composition of an octree and a 3D grid.

4.2. Parallel Composition

When we have a complex scene and no optimizer is known as the best one to reduce the cost in terms of execution time, a parallel composition is very useful. This implies the concurrently or parallel execution of two or more optimizers. We will define the parallel composition of two optimizers as follows:

Let A_1 and A_2 be two optimizers, let $r \in \mathcal{R}$ be a ray, and let $s \in \mathcal{S}$ be a scene, We say that A_π is the parallel composition of A_1 and A_2 (noted as $A_\pi = A_1 \parallel A_2$) if and only if A_π is the optimizer which fulfils the following condition:

$$A_\pi(r,s) \;=\; A_1(r,s) \;\cap\; A_2(r,s) \tag{19}$$

An example of a parallel composition is shown at figure 2. Here we show two optimizers based on spatial indexing. The first one is based on a 3D Grid (A_G), and the second is based on an Octree (A_O). This figure shows us that A_O returns less objects than A_G. That is:

$$Card(A_G(r,s)) \;>\; Card(A_O(r,s))$$

In this case, the parallel composition is very useful because a reduced number of objects is returned in a relatively great number of rays for this scene.

4.3. Recursive Optimizers

There are many strategies available to build a SR based on a hierarchical partitioning of the scene. An example of this are the octrees [10, 8], the binary trees[22], the bounding volumes hierarchy[12]. All these optimizers may be described as *recursive optimizers*.

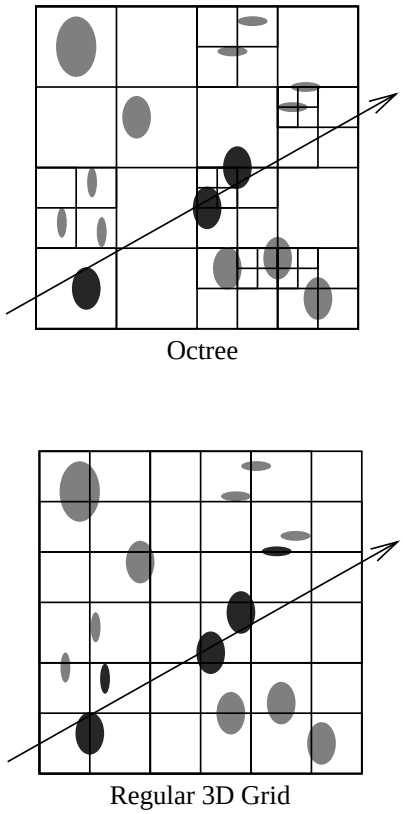

Figure 2: *An example of parallel composition.*

This definition is very connected with the recursion concept and the sequential composition. With these premises, a recursive optimizer can be defined as follows:

Let A be an optimizer based on spatial indexing, let B be also other optimizer based on spatial indexing. A is a recursive optimizer if and only if it satisfies:

$$A \;=\; B \mid A \tag{20}$$

An appropriated example of this group of optimizers is an octree. An octree can be seen as a sequential composition of regular grid (containing $2 \times 2 \times 2$ voxels) with itself.

5. An Optimization Efficiency Measurement

Using the above definitions and results, an optimization efficiency measurement can be defined. This measurement consists of computing the number of objects that an optimizer is capable to reject for a set of rays and a given scene.

A way to get this measurement is to use a random distribution of rays for a given optimizer, and for the whole scene, and after this to compute the relative gain in efficiency. This

computation can be formally expressed by introducing:

$$M(A^+, s, P) = \int_{r \in \mathbb{R}} \frac{Card(A^+(s,r))}{Card(s)} dP(r) \qquad (21)$$

where *Card* is the function which returns the number of elements which are in a set, and $P(r)$ is a probability measure function which models the probability distribution of the rays to be processed.

Probability measure P depends on the usage of the optimizer in a rendering system, this is, different distributions of rays can be obtained. For instance, when a simple ray casting is applied, in most cases the rays start from the observer and reach a particular surface point of the scene. However, when other algorithm ray-based is applied, the rays may start from the light sources.

6. Conclusions and Future Work

In this work, a formal model of optimizer is proposed. This formal model is shown as a function that reduces the number of candidate objects for the ray-object intersection test. Moreover, a model for optimizers based on spatial indexing has been proposed.

Two formal models of composed optimizer have been presented: the sequential and parallel composition, in addition to the recursive optimizers.

In this formalism, a measure function to study the performances of any optimizer with respect to other one was proposed.

As future work, we are planning to produce definitions of concrete optimizers by applying this formal framework.

With respect to the efficiency measurement, we are also planning to study this function. Normally, this is based on a uniform distribution. In terms of implementation, we have into account the rendering algorithm applied and a reduced sample of the all rays is considered to obtain an approximate estimation of this measure function. Obviously, from two given optimizers and when this measure is known, one of these optimizers will be more suitable than another.

In short, an optimizer will be better than another one when it is capable to reduce the average number of candidate objects.

This measurement can be useful to select the best optimizer. Notice that in this case, this measurement does not take into account whether the optimizer is based on spatial index or not. It only obtains the performance of an optimizer with respect to a null optimizer. It will also allows to compare the performances of two optimizers.

Acknowledgements

Special thanks to Juan Carlos Torres for his contribution to this work. This work has been supported by a grant coded as TIC98-0973-C03-01 from the Committee for Science and Technology of the Spanish Government (CICYT).

References

1. J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *EUROGRAPHICS'87*, pages 3–10, Amsterdam, 1987.

2. J. Arvo. Ray tracing with meta-hierarchies. In *SIGGRAPH'90*, volume 15(6), pages 56–62, 1990.

3. J. Arvo and D. Kirk. Fast ray tracing by ray classification. *Computer Graphics*, 21(4):55–64, 1987.

4. J. Arvo and D. Kirk. *A Survey of Acceleration Techniques*, chapter chapter 6. An Introduction to Ray Tracing, pages 201–262. Academic Press, San Diego, 1989.

5. S. Coquillart. An improvement of the ray-tracing algorithm. In *EUROGRAPHICS'85*, pages 77–88, 1985.

6. N.D. Cuong. *An Exploration of Coherence-based Acceleration Methods Using the Ray Tracing Kernel G/GX*. PhD thesis, TU-Dresden, Germany, 1997.

7. R. Endl and M. Sommer. Classification of ray-generators in uniform subdivisions and octrees for ray tracing. *Computer Graphics Forum*, 13(1):3–20, 1994.

8. A. Fujimoto and K. Iwata. Arts: Accelerated ray tracing system. *IEEE Computer Graphics & Applications*, 6(4):16–26, 1986.

9. I. Gargantini and H.H. Atkinson. Ray tracing and octree: Numerical evaluation of the first intersection. *Computer Graphics Forum*, 12(4):199–210, 1993.

10. A.S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics & Applications*, 4(10):15–22, 1984.

11. A.S. Glassner. *An Introduction to Ray Tracing*. Academic Press, 1989.

12. J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics & Applications*, 7(5):14–20, 1987.

13. E. Haines and D.P. Greenberg. The light buffer: A shadow testing accelerator. *IEEE Computer Graphics & Applications*, 6(9):6–16, 1986.

14. D. Kirk and J. Arvo. The ray tracing kernel. In *Proceedings of Ausgraph'88*, pages 75–82, 1988.

15. J. Revelles, C. Ureña, and M. Lastra. An efficient parametric algorithm for octree traversal. In *The 8th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Media 2000, Plzen (Rep. Chequia)*, ISBN: 80-7082-612-6, pages 212–219, 2000.

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

17. H. Samet. Implementing ray tracing with octrees and neighbor finding. *Computer & Graphics*, 13(4):445–460, 1989.

18. I.D. Scherson and E. Caspary. Data structures and the time complexity of ray tracing. *Visual Computer*, 3:201–213, 1987.

19. J.M. Snyder and A.H. Barr. Ray tracing complex models containing surface tessellations. In *SIGGRAPH'87*, pages 119–128, 1987.

20. J. Spackman and P.J. Willis. The smart navigation of a ray through an oct-tree. *Computer & Graphics*, 15(2):185–194, 1991.

21. N. Stolte and R. Caubet. Discrete ray-tracing of huge voxel spaces. In *EUROGRAPHICS'95*, pages 383–394, 1995.

22. K. Sung. *Ray Tracing with the BSP Tree*, pages 271–274. Academic Press, 1992.

23. J.C. Torres. *Abstract Representation of Graphics Systems. Graphic Object Theory*. PhD thesis, Department of Lenguajes y Sistemas Informaicos, University of Granada (Spain), 1992.

24. J.C. Torres and B. Clares. Graphics objects: A mathematical abstract model for computer graphics. *Computer Graphics Forum*, 12(5):311–328, 1993.