Evaluation of BSP properties for ray-tracing

Vlastimil Havran, Jiří Žára

Czech Technical University, Faculty of Electrical Engineering,
Department of Computer Science and Engineering, Prague, Czech Republic

e-mail: {havran, zara}@sgi.felk.cvut.cz

Abstract

This paper deals with method for evaluation of different Binary Spatial Partitioning techniques. The overall time for ray-tracing is composed of two parts: the time consumed by calculation of the intersections of rays with objects in the scene and the time required for traversal of auxiliary spatial data structures. Different algorithms for construction of bintree have different properties, which influence overall rendering time. We outline a new method for evaluation of bintree qualities for ray tracing purposes by two n-tuples of parameters.

Key words: space subdivision, bintree, BSP tree, spatial data structures, ray tracing

1 Introduction

Ray tracing is commonly used for the photo-realistic rendering nowadays. Its main idea was originally developed by Whitted[13]. He showed that up to 95 % of total rendering time is devoted to the calculation of intersection of rays with objects. It is true for the most naive ray-tracing approach, where the intersection calculations are performed for each ray with all objects in the scene. Since then, a variety of space subdivision and bounding volume schemes has been proposed, that decrease the number of intersection calculations. Firstly, the bounding volume approach has been designed by Whitted[13], Kajiva[5], and Roth[9]. The space subdivision schemes were designed at the second half of nineties (Glassner[2], Fujimoto and Iwate [1], Kunii and Wyvill in 1985[12] etc.), further improved, and combined together[8]. The main representatives of this technique are uniform space subdivision, octree, and binary space partitioning. The improvement is based on the adaptation of data structures in dependence on the object distribution in a space. It includes the regularity, the depth, and the positioning of splitting planes used in spatial data structures. Haines designed a set of testing scenes [4] to measure the contribution of accelerating methods. The methodology of testing by the same scenes is meaningful, but naturally it can cover no properties of acceleration techniques.

The comparison of all algorithms presented in scientific papers is rather impossible because of different implementations and different architectures used. From this reason some papers compare the behavior of different acceleration techniques supposing the rest of the code remains unchanged [7]. Some codes for traversal algorithm have become de facto standards [11], which other papers refer to.

In this paper we show the drawbacks of comparison of two algorithms by their execution times only and we outline a new formalism for this purpose. We demonstrate our approach on adaptive bintree construction and on practical measurements carried out. Section 2 gives basic information about bintree and about its optimization of construction from statistical point of view. The bintree properties are described in section 3. The methodology for comparison of two binary trees is suggested in section 4. Scene characteristics are presented in section 5. The results of measurement are reported in section 6, they are discussed in section 7, and followed by concluding remarks in section 8.

2 Binary space subdivision tree

Binary space subdivision tree (BSP tree, bintree) is a spatial data structure that can be used to solve a variety of geometrical problems. It was initially developed as a means of solving the hidden surface problem [3].

A BSP tree hierarchically subdivides a volume of n-dimensional space containing a collection of objects defined in n-space. The result of the splitting of n-space by hyper-plane is two oriented n-dimensional subspaces. These subspaces are always convex. The subdivision scheme forms a bintree with a root, inner nodes, and leaves.

The leaves of a bintree for purposes of ray-tracing are either occupied by objects or they are vacant. The splitting plane is positioned in a mid-point of the current axis in the original algorithm [2]: the space is split by plane perpendicular to the axis, that is changed regularly in order x, y, and z. If the number of objects in current node falls below a specific threshold or the depth of current node is equal to a given maximal depth, current node is marked as a leaf and no further subdivision is performed. This is the simplest and native termination criterion for construction of bintree. There have been more sophisticated termination criteria suggested [10] that take into account the statistic probability of a ray passing through a voxel. Another improvements concern the positioning and orientation of splitting plane. It need not to be placed exactly at spatial median of current space, but it is more advantageous to put it at a specific position from statistical point of view. Due to the lack of space in this paper we refrain from the details of adaptive construction of the tree proposed by MacDonald and Booth [6] that use *surface area heuristics.*

3 Properties of a bintree for purposes of ray tracing

We have investigated thoroughly the ways for bintree construction and traversal. The bintree properties can be grouped into two parts. The first one includes *static* properties gained by bintree construction and the second one reflects *dynamic* behavior of bintree during the traversal itself. These properties are mutually connected with the scene characteristics over which the bintree is built. The method for bintree construction can significantly influence the usability of this method. Let us recall both property groups and the scene characteristics on condition that for both mentioned termination criteria are used during construction of a bintree. The parameters are marked by following letters:

S .. *specified* before rendering, independent of bintree and real implementation.
D .. *derived* from the **S** parameters and the nature of ray-tracing algorithm itself.
C .. computed from the *construction* of the tree, independent of rendering.
R .. computed from *rendering*, they are dependent on **S**, **D**, and **C**.
T .. *time* parameters dependent on implementation and architecture used.

Some parameters are connected with ray-tracing traversal, some of them are independent of the static properties of bintree, but they depend only on the parameters specified as S, therefore they are marked by D.

It is even possible to enlarge the set of parameters above by other mean and maximal values or to relate the parameters to specific groups of rays, but these extensions are not so important. The overall time required for ray tracing algorithm itself can be expressed (terms are described in Tables 1-3) in a simplified way as follows:

$$\begin{aligned} T_{TR} &= (T_T + T_P).N_{NODES} + T_I.(N_{HPR} + N_{HSECR}) + \\ &\quad +T_{IUN}.(N_{PRIT} + N_{SECRIT} + N_{SRIT} - N_{HPR} - N_{HSECR} - N_{HSR}) + const. \end{aligned} \tag{1}$$

where T_I is the time for performing one ray-object succeeded intersection test
T_{IUN} is the time for performing one ray-object failed intersection test
T_T is the time for performing one traversal step
T_P is the time for decision step in parent node

N_X,N_Y	S	the resolution of the image
N_O	S	the number of scene objects and their distribution in the scene
N_L	S	the number and position of lights in the scene
OP_{CAMERA}	S	the position, the orientation, and other settings of camera
S_{COV}	S,R	percentage of screen coverage
D_{COMPX}	D	depth complexity, i.e., the average number of object primitives that are hit by an arbitrary ray from the viewpoint (see [8])
V_{SCENE}	S,D	the volume taken by the bounding box of the whole scene
V_{BB}	S	the sum of volumes specified by parallelepiped bounding boxes of object primitives
$R_{BBSC} = \frac{V_{BB}}{V_{SCENE}}$	S	the ratio of volumes for bounding boxes to the whole scene
D_{AREC}	S	maximal depth allowed for secondary rays
D_{MAX}	S	maximal depth allowed for construction of bintree
N_{OILF}	S	the minimal number of objects in leaf to stop further subdivision

Table 1 Scene and image characteristics

Total time T_{TR} can be minimized by all R parameters in this equation including the volume of non-empty leaves and by other C parameters, e.g., by the average number of objects in a leaf.

The ratio of T_I to T_T differs for different kinds of objects. For sphere, that is probably the simplest object primitive for intersection computation, the ratio is about one for implementation used in section 5. The ratio of T_I to T_T is about three for triangles. Opposite to the measurements for naive ray tracing [13], the total time for traversing and the time taken by intersection tests is quite comparable for tests performed.

More methods can be used to determine T_I, T_P, T_T, and T_{IUN}. The simplest way is to use performance analyzer, that measures the inclusive and exclusive times devoted to procedures. Another method, which we have also tested, is based on the equation 2. The number of traversal steps and the total time is measured for a set of different bintrees. The unknown variables (times) are then determined by linear regression method of the equation. The results obtained by both methods are very similar.

4 Comparison methodology for ray tracing

The intent of this section is to give a new method for comparison of bintree construction techniques for ray-tracing. The important presupposition for comparison of two methods for bintree construction or implementation is that the measurements are performed under the same conditions. In other words, the parameters denoted by S and thus also by D in previous section have to be equal, i.e., the result of both rendering processes has to be the same image. The difference of some parameters indicates some errors in the implementation. The time (T_{TR}) and the memory (N_{LS}) requirements are the most important aspects of practical applicability of ray-tracing software. We have more choices for comparison of ray-tracing using bintree.

The first possibility is to compare some bintree properties for a given scene by C and R parameters independently on architecture, implementation, and compiler used. The differences of algorithms for bintree construction follow from the (ir)regularity of changing the orientation and the positioning of splitting spaces, the termination criteria, cutting off empty spaces etc. We recommend to use the following septet Δ of the parameters above for this type of comparison:

$$\Delta = < N_{LS}, R_{ETNLS}, N_{ADL}, N_{AOIFL}, R_{FVWV}, N_{RPRT}, N_{AT} > \quad (2)$$

$D_{REACH} \leq D_{MDEPTH}$	C	the maximal depth reached
N_{LS}, $N_{IN} = N_{LS} - 1.0$	C	the number of leaves and inner nodes
N_{ELS}	C	the number of empty leaves (without objects)
$R_{ETNLS} = \frac{N_{ELS}}{N_{LS}}$	C	the ratio of empty leaves to all leaves
N_{RO}	C	the total number of references to objects from all (non-empty) leaves
$N_{ADL} = N_{RO}/N_O - 1.0 (\geq 0.0)$	C	the average number of objects duplication for one object in leaves
$N_{AOIL} = \frac{N_{RO}}{n_{LS}}$	C	the average number of objects in all leaves
$N_{AOIFL} = \frac{N_{RO}}{n_{LS} - n_{ELS}}$	C	the average number of objects in full leaves
V_{EMPTY}	C	the sum of volumes taken by empty leaves
$R_{FVWV} = \frac{V_{SCENE} - V_{EMPTY}}{V_{SCENE}}$	C	the ratio of volumes of full leafs to V_{SCENE}
T_{CB}	T	time required for construction of bintree

Table 2 Static properties of bintree

The second type of comparison concerns the implementation of ray-tracing source code. In this case the parameters S,D,C, and R remains unchanged, parameters denoted by T characterize the quality of implementation, the optimization efficiency of a compiler, or the performance of the architecture tested for ray tracing. For the overall evaluation of ray-tracer performance by triplet Λ other parameters from S,D,C, and R should also be taken into consideration.

$$\Lambda = < T_{CB}, T_{TR}, \frac{T_{TT}}{T_{TR}} >, \tag{3}$$

where T_{TT} is the time devoted to traversing of bintree ($T_{TT} = N_{NODES}.(T_T + T_P)$), introduced in formula (2).

5 Test scenes

A selection of data for testing the methods represents an important step in the experimental verification of theoretical conclusions. Since there are many degrees of freedom in the construction of scenes, it is useful to choose data, which have been either already used in other research works or which contain objects and their arrangement typical for certain practical application.

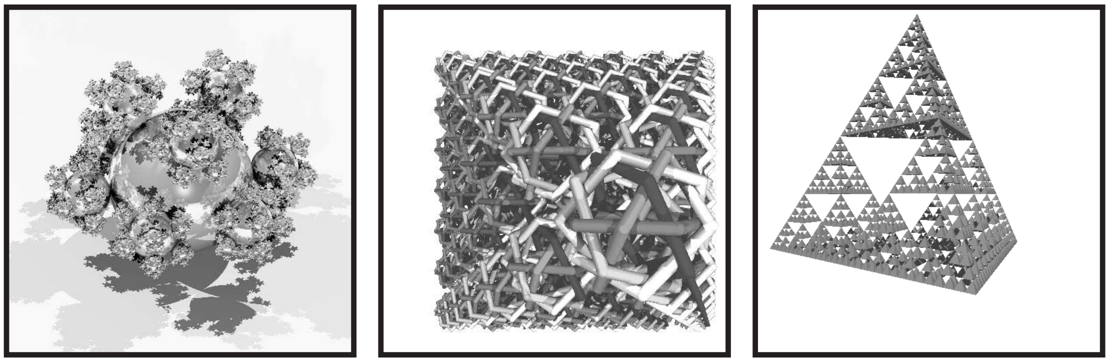

Figure 1: Standard (SPD) scenes – *balls*, *rings*, and *tetra* (resolution 512×512)

$N_{PR} = N_X \times N_Y$	D	the number of primary rays
N_{PRIT}	R	number of intersections tests for all primary rays and object primitives
$N_{HPR} = N_{PR}.S_{COVERAGE}$	D	the number of primary rays hitting the objects
N_{SR}	D	the number of shadow rays
N_{SRIT}	R	the number of intersection tests carried out for all shadow rays
N_{HSR}	D	the number of shadow rays hitting objects
N_{SECR}	D	the number of secondary (reflected + refracted) rays
N_{SECRIT}	R	the number of intersection tests performed for all secondary rays
N_{HSECR}	R	the secondary rays hitting the objects
$N_{RPRT} = \frac{N_{PRIT}+N_{SRIT}+N_{SECRIT}}{N_{HPR}+N_{HSR}+N_{HSECR}}$	R	the ratio of all intersection tests performed to minimal intersection tests
N_{VLS}	R	the number of all visited leaves
N_{VELS}	R	the number of visited empty leaves
$N_{VFLS} = N_{TRLS} - N_{TRELS}$	R	the number of visited non-empty leaves
N_{INODES}	R	the number of traversed inner nodes
$N_{NODES} = N_{INODES} + N_{VLS}$	R	the number of all traversed nodes including inner ones, empty, and non-empty leaves
$N_{AT} = \frac{N_{NODES}}{N_{PR}+N_{SR}+N_{SECR}}$	R	the average number of traversed nodes per one ray (primary, secondary, shadow)
T_{TR}	T	time required for the rendering itself of the image on a specific bintree

Table 3 Dynamic properties of bintree

That is the reason why we have used six test scenes organized into two groups (see Table 4). The first three scenes (*balls, rings, tetra*) represent **Standard** data known in graphics community as standard ray tracing benchmark data (SPD) introduced by Haines [4]. Due to the simplification of our implementation all cylinders in scene *rings* have been approximated by polygons.

The second group of test data consists of **Specific** scenes coming from user applications. The scene *fluid* is the result of visualization of a fluid dynamics simulation, the *m–fluid* contains the same data as *fluid* plus one toroid and one large mirroring plane at the bottom. The scene *room* originates from CAD system and represents an example of naturally looking 3D scene.

6 Results of measurement

In this section we compare two methods for the construction of bintree by two n-tuples of parameters. The first method (1) is the algorithm [11] with splitting plane orientation changing in cyclic order; its position lies in the mid-point. In addition to surface area heuristics, the second method (2) uses cutting off the empty spaces in both on the outside and the inside of currently processed node. Due to the lack of space the methods are not discussed in detail in this paper.

Hardware and programming tools

The measurement reported above have been carried out on Power Challenge XL. It is multiprocessor machine with 12 CPU MIPS R10000 /195 MHz, 2048 MB RAM, and the swap space with 1x2GB, 5(2)x4GB SCSI disks. The internal bandwidth of data bus is 1.2 GB/sec. The internal

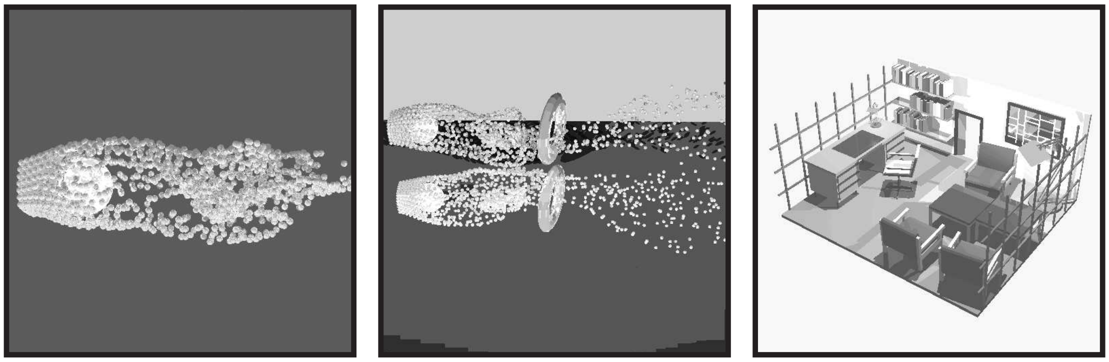

Figure 2: Specific scenes – *fluid*, *m-fluid*, and *room* (resolution 512 × 512)

characteristics	test scenes					
	SPD			Specific		
	balls	*rings*	*tetra*	*fluid*	*m-fluid*	*room*
Geometry	spheres	polygons	triangles	spheres	spheres, polygons	polygons
N_O	7382	46201	4096	2514	2899	7788
N_L	3	3	1	4	2	3
S_{COV}[%]	100.0	100.0	19.0	7.26	70.6	40.6
R_{BBSC}[%]	0.19	65.6	12.5	42.0	1.32	54.2
D_{AREC}	4	5	4	5	5	4
D_{MAX}	20	20	20	20	20	20
N_{OILF}	1	3	1	1	1	1

Table 4 Scene and image characteristics for all test scenes

cache is 32 KB and second level cache for each processor is 2 MB. Typical benchmark for one processor is 8.50 SPECint95 and 13.1 SPECfp95. Operating system IRIX 6.2 is clone of UNIX.

The source codes of ray tracer in C-languages have been compiled with optimization switches -O0 by MIPS PRO Compiler. The *user time* (see UNIX) is reported in the time measurements above, which is almost independent of the load of the machine. It varies a little due to the caching between processor and main memory. The time part consumed by traversal code has been determined by performance analyzer.

7 Discussion

We can analyze from Table 5 and 6 the speedup is reached by decreasing the number of intersection tests and by the number of traversal steps. The ratio for methods 1 and 2 in case of percentage parameters (the numbers itself are denoted by + or −) is calculated as the difference of percentage for methods 1 and 2, because it is more convenient for comparison.

We have tested both the algorithms for bintree construction on different architectures, operating systems, and under different compiler optimization switches. We have also suggested the modifications of source code for BSP traversal (see [11]), that decrease the total time for the scenes above up to 25%. The source code optimization is based on fact the time part consumed by traversal is significant. From the lack of space we cannot show all details of programming techniques and their impact on the overall performance of ray-tracer, for instance the passing of

properties	test scenes								
	balls			*rings*			*tetra*		
	1	2	$\frac{2}{1}$[%]	1	2	$\frac{2}{1}$[%]	1	2	$\frac{2}{1}$[%]
N_{LS}	3244	19892	613	153456	78677	51	96056	11322	11.8
R_{ETNLS}[%]	32.3	25.4	-6.9	15.9	25.5	$+9.7$	28.6	25.9	-2.7
N_{ADL}[%]	130.7	284.7	$+154.0$	4505	1337	-3168	7400	1052	-6348
N_{AOIFL}	7.75	1.91	24.6	16.48	11.33	68.8	4.48	5.62	12.5
R_{FVWV}[%]	43.3	28.8	-14.5	23.4	8.10	-15.3	6.54	3.12	-3.42
N_{RPRT}	34.8	22.3	63.3	117.8	86.1	73.1	21.8	13.33	61.1
N_{AT}	26.2	4.35	16.6	56.51	34.5	61.0	36.3	17.4	47.9
$T_{CB}[sec]$	0.86	2.46	286	44.0	24.7	56.0	5.31	1.17	22.0
$T_{TR}[sec]$	128.7	43.8	34.0	612.0	340.6	55.7	21.0	8.31	39.6
T_{TT}/T_{TR}[%]	51.6	69.9	$+18.3$	17.8	16.4	-1.4	49.5	60.5	$+11.0$

Table 5 The n-tuple Δ and Λ for SPD scenes

parameters between the functions in C-language. Let us remark at least, the total rendering time can be decreased to 65% on average by source program modifications and by compiler switches combined together.

8 Conclusions

In this paper, we have outlined the basic requirements and conditions needed for meaningful comparison of two ray tracing methods.

The properties for evaluation of bintree for ray tracing purposes have been declared. We have chosen two n-tuples of the most important parameters that should be used by graphics community scientists to compare the ray tracing algorithms using modifications of bintree algorithm. Septet Δ includes mainly the relative ratio parameters independent of implementation and characterizing the quality of bintree for ray tracing purposes. Triplet Λ characterizes the quality of implementations assuming that the same algorithm is used for bintree construction. Triplet Λ allows to compare the quality of different implementations, the performance of architecture used for measurement, and the impact of optimization reached by compilers.

Similar methodology should be used for evaluation and comparison of other spatial data structures used for acceleration of ray tracing or other graphics algorithms in future. We agree with the model of the standard benchmarking data introduced by Haines as the input for algorithms, but we draw up the requirements put on the resulting format of scientific papers to be valuable for further usage, because simplified approach given by comparison of execution times and two next parameters only is rather inadequate.

References

[1]Fujimoto A. and Iwata K. Accelerated ray tracing. In *Proceedings Computer Graphics International '86*, pages 41–65, 1986.

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

[3]Fuchs H., Kedem M.Z., and Naylor B. On visible surface generation by a priori tree structures. In *Proceedings of SIGGRAPH '80*, volume 14, pages 124–133, July 1980.

properties	test scenes								
	fluid			*m-fluid*			*room*		
	1	2	$\frac{2}{1}$[%]	1	2	$\frac{2}{1}$[%]	1	2	$\frac{2}{1}$[%]
N_{LS}	28619	15912	55.6	18565	16727	90.1	167834	27925	16.6
R_{ETNLS}[%]	11.3	27.7	+16.4	16.4	28.0	+11.6	27.5	18.4	−9.1
N_{ADL}[%]	2709	1486	−1223	1631	1121	−510	13172	1483	−11689
N_{AOIFL}	2.78	3.47	124.8	3.23	2.94	91.0	8.49	5.41	63.7
R_{FVWV}[%]	37.2	3.70	−33.5	42.8	1.21	−41.6	12.2	40.6	+28.4
N_{RPRT}	10.0	4.0	40.0	10.1	3.3	32.7	63.0	19.3	30.6
N_{AT}	21.1	16.1	76.9	24.3	16.7	68.7	64.7	32.3	49.9
$T_{CB}[sec]$	1.16	1.57	135.3	1.11	1.64	148.8	14.56	3.06	21.0
$T_{TR}[sec]$	20.6	14.1	68.1	63.8	36.6	57.4	277.0	71.9	26.0
T_{TT}/T_{TR}[%]	75.1	82.3	+7.2	61.8	80.2	18.4	26.8	41.1	+14.3

Table 6 The n-tuple Δ and Λ for specific scenes

[4]Eric A. Haines. Standard Procedural Database (SPD) package. *3D/EYE*, 1992. Available via ftp://ftp.eye.com/pub/graphics/SPD.

[5]Kajiya JT. New techniques for ray tracing procedurally defined objects. In *Proceedings SIGGRAPH '83*, volume 17, pages 91–102, 1983.

[6]J. David MacDonald and Kellogg S. Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer*, 6(3):153–166, June 1990.

[7]Endl R. and Sommer M. Classification of ray-generators in uniform subdivisions and octrees for ray tracing. *COMPUTER GRAPHICS forum*, 13(1):3–19, 1994.

[8]Isaac D. Scherson and Elisha Caspary. Data structures and the time complexity of ray tracing. *The Visual Computer*, 3(4):201–213, December 1987.

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

[10]K. R. Subramanian and Donald S. Fussell. Automatic termination criteria for ray tracing hierarchies. In *Proceedings of Graphics Interface '91*, pages 93–100, June 1991.

[11]Kelvin Sung and Peter Shirley. Ray tracing with the BSP tree. In David Kirk, editor, *Graphics Gems III*, pages 271–274. Academic Press, San Diego, 1992. includes code.

[12]Kunii T.L. and Wyvill G. CSG and ray tracing using functional primitives. In *Proceedings of Computer Graphics International '85*, pages 137–152, 1985.

[13]T. Whitted. An improved illumination model for shaded display. *CACM*, pages 343–349, June 1980.