DE-tree ray tracing and box-query performance data.

Miguel R. Zuniga
New Generation Software Corporation
NGSC Report No 2005ZF1 (Updates 2004ZN1)
mike@newgs.com
Feb 2005.

1 Update Information.

This report updates the earlier report (2004ZN1) with:

1. Requested performance data on more modern CPUs and compilers, including performance of the Java version of FORT.
2. Performance data on the BART benchmark, with comparisons to the OpenRT framework and other systems.
3. Initial collision detection and dynamic box query performance.
4. R&D directions under consideration.

The three sections toward the end of this report ("Pentium 4 Performance and th BART benchmark.", "Collision detection and dynamic box queries", and "R&D directions under consideration") are not included in report No. 2004ZN1.

2 Introduction, testing framework, and results.

Tables 1 and 2 below include performance data of the new ray shooting algorithm (RSA) called the DE-tree. The data is from rendering the standard procedural database (SPD) scenes. This data structure is in part of interest because it achieves efficiency in the four major performance metrics of robustness, search speed, memory usage, and RSA build time. Simultaneously achiving these goals is highly desirable but utill now this been elusive and one can find recent statements of future research focusing on it (e.g. [Hurl02] concerning Intel's own ray tracer).

The SPD is described in [Hain87] and has an associated web page; the measures of performance presented are from [Hav00B]. These tables include data for the DE-tree as the RSA of the FORT ray tracing program. Data for other RSAs in these tables are derived from [Hav00A] and [Havr02]. The OSAH+AT kd-tree information is from line 45 of the tables in Appendix E of [Hav00A]. The OSAH+AT+SC kd-tree information is from [Havr02]. These RSAs are selected as they are the best performing RSAs we could find in the published literature. The BES RSA is the best efficiency scheme (BES) program winner (based on ray tracing time) within GOLEM rendering system *prior* to the innovations that lead to the OSAH+AT kd-tree development. The RSAs that competed in the BES program included BSP trees, kd-trees, octrees, bounding volume hierarchies, uniform grids, and adaptive grids; and results for all these RSA are included in [Hav00A] and on the GOLEM website. Table 1 includes averages for the key performance parameters of the ten SPD scenes of Group 4 and of Group 5 (the SPD scenes of roughly 10^4 and 10^5primitives). Data for the uniform grid (UG) is included in this table simply because of its popularity and its small RSA build time. Table 2 includes data for individual scenes within G^5. For the sake of recording computational complexity, we have provided an extra row in table for a larger scenario (rings40) which appears as the "toughest" scenario for the DE-tree based FORT and which will be discussed in a subsequent section. Further data and pictures can be obtained from the GOLEM homepage and the FORT homepage.

For the FORT experiments that produced the data below, the details of the rendering are strictly done according to the rules stated in the SPD Readme.txt file (e.g. 513 x 513 primary rays, plus an additional four levels of reflection, and ray queries are performed one query at a time). FORT does not use shadow buffer or light buffer algorithms, primitive caching, or spatial or temporal coherence. It does not use vectorization, MMX, or SSE, etc. FORT uses explicit box-ray intersection tests only for some parts of the gears scenario, i.e. ray-primitive intersection tests are not generally preceded by ray-bounding-box tests. The DE-tree nodes are 24 bytes in size. With Np being the number of graphics primitives, there are Np − 1 interior tree nodes and Np leaf nodes used. Each leaf node has a direct pointer to a graphics primitive (field *obj*). The DE-tree search algorithm is called with initial arguments *mint* and *maxt* - the initial minimum and maximum ray-object distance limits - to be the same values for all shadow rays and the same values for all other rays.

2.1 Test hardware and compiler

The FORT experiment data is collected on a Gateway SOLO 9300 laptop with a Mobile Pentium III of 700 MHz, 100 MHz FSB, 256 MB RAM running SUSE Linux 8.2, gcc 3.3 at compiler options "-O2 -g" and collecting all ray tracing statistics. Data for the GOLEM BES results are from a Pentium II 350 MHz, 100 MHz FSB. Data for the OSAH+AT kd-tree and OSAH+AT+SC kd-tree results are from a Pentium II 466 MHz. The spec corporation (www.specbench.org) maintains online performance metrics for similar computers.

2.2 Symbols for tables 1 and 2.

Tb - the time (in seconds) to build the RSA.
Tr - the time (in seconds) to perform all ray tracing calculations once the RSA is created.
Nts - the average number of nodes accessed per ray.
Ner - the total number of references to objects in elementary nodes of the RSA DS.
Ng - the number of generic nodes in the RSA (usually the number of interior nodes for tree data structures)
Ne - the maximum number of elementary nodes in the RSA DS (usually the number of leaf nodes for tree data structures).
Np - the umber of graphics primitives in the SPD scene.
Ritm - the ratio of ray-object intersection tests performed to the minimum number of intersection tests (Ritm $>$ 1.0 and also Nit = Ritm * Rsi).
UG - The uniform grid RSA.
AG - The adaptive grid RSA.
BSP/kd - The BSP tree or kd-tree RSAs built using AHTC (see [Hav00A]).

2.3 Table 1 – Performance Averages for SPD Scenes G4 and G5

RSA	Group	Ng+Ne	Ner	Ritm	Nts	Tb	Tr
DE-tree	G4	15271	7636	5.48	25.97	0.02	9.56
Kd-tree ATC	G4	16634	19515	15.1	24.47	2.02	26.7
Kd-tree OSAH+TA	G4	33811	25879	12.3	26.7	1.2	15.1
Kd-tree OSAH+TA+SC	G4	34925	22824	9.8	26.8	2.6	14.8
UG	G4	38369	36013	283.9	13.97	0.39	100.4
DE-tree	G5	197761	98881	5.62	32.57	0.39	12
Kd-tree ATC	G5	46227	147514	41.69	26.99	22.26	40.4
Kd-tree OSAH+TA	G5	501817	367546	12.35	35.22	22.75	19.1
Kd-tree OSAH+TA+SC	G5	525099	328496	9.6	35.1	66.34	18.5
UG	G5	471705	385702	754.9	28.8	5.67	282.3

2.4 Table 2 – Performance Comparison for Individual Scenes within SPD G5

RSA	SPD scene	Np	Ng+Ne	Ner	Ritm	Nts	Tb	Tr
DE-tree	balls5	66431	132861	66431	3.14	30.45	0.26	11.65
BES (BSP/kd)	balls5	66430	14261	84927	46.33	27.48	16.77	42
Kd-tree OSAH+TA	balls5	66430	160187	173040	13.5	35.17	14.8	18.2
DE-tree	gears9	106435	212869	106435	3.83	27.64	0.44	24.34
BES (BSP/kd)	gears9	106435	43629	163635	9.85	19.53	22.97	40.59
Kd-tree OSAH+TA	gears9	106435	784291	757083	7.56	26.16	33.56	30.69
DE-tree	jacks5	42129	84257	42129	7.24	38.28	0.13	4.88
BES (BSP/kd)	jacks5	42129	72297	132833	33.52	38.63	12.47	31.82
Kd-tree OSAH+TA	jacks5	42129	386409	288851	20.08	53.98	12.55	20.78
DE-tree SMI	lattice29	105300	210599	105300	4.31	48.8	0.37	22.49
BES (UG)	lattice29	105300	531441	749107	14.09	9.23	8.54	46.39
Kd-tree OSAH+TA	lattice29	105300	1442715	802648	5	54.68	29.3	30.09
DE-tree	mount8	131076	262151	131076	4.82	21.59	0.5	4.78
BES (BSP/kd)	mount8	131076	26643	153865	18.6	18.9	25.82	25.14
Kd-tree OSAH+TA	mount8	131076	399809	182738	6.11	20.84	23.81	12
DE-tree	rings17	107101	214201	107101	8.59	70.94	0.46	34.73
BES (BSP/kd)	rings17	107101	54311	201209	39.12	40.25	23.55	106.61
Kd-tree OSAH+TA	rings17	107101	608701	614753	17.94	54.63	27.46	50.63
DE-tree	rings40	1328401	2656801	1328401	9.09	82.97	7.27	41.56
DE-tree	sombrero4	130050	260099	130050	5.02	28.74	0.53	2.93
BES (BSP/kd)	sombrero4	130050	36659	135207	41.7	19.17	27.63	6.9
Kd-tree OSAH+TA	sombrero4	130050	464257	171667	6.85	20.06	24.14	3
DE-tree	teapot40	103680	207359	103680	6.62	31.73	0.39	8
BES (BSP/kd)	teapot40	103680	37186	181085	57.4	30.48	21.77	23.85
Kd-tree OSAH+TA	teapot40	103680	525445	435017	12.77	38.29	22.37	10.76
DE-tree	tetra8	65536	131071	65536	9.24	15.78	0.19	1.41
BES (BSP/kd)	tetra8	65536	38195	65536	86.38	20.17	10.61	3.57
Kd-tree OSAH+TA	tetra8	65536	48264	65536	10.12	22	8.95	1.95
DE-tree	tree15	131071	262141	131071	3.36	11.71	0.6	4.83
BES (AG)	tree15	131071	393011	182880	79.01	30.31	370.43	43.38
Kd-tree OSAH+TA	tree15	131071	145845	184136	23.56	20.44	30.54	12.51

2.5 In summary of the data above we state some observations for the DE-tree

1. Its build time can be orders of magnitude faster than competing RSAs. On the SPD G4 and G5 groups, it is at least several times faster to build than any RSA tested with the GOLEM system.
2. Its build time is proportional to Np * log (Np) and is relatively independent of the scenario.
3. Its average number of nodes visited per query ray ranges from about log(Np) (tree scenarios) to about 4.5*log(Np) (rings scenario). This is obviously scenario dependent though robust.
4. The average number of ray-primitive intersection tests performed per ray using the DE-tree is significantly reduced. Its Ritm metric is more then 40% smaller then that of best know competitor for this metric (the OSAH+TA+SC kd-tree). The improvement is more than a factor of two over all other competitors.
5. Its total number of nodes and references to objects (Ne + Ng + Ner) is not only fixed given Np, but on average it is several times less than any RSA that also has good search performance. item Its performance metrics concerning build, search, memory and number of intersection tests performance are much better than the uniform grid's even in cases where the scene topology is ideal for uniform grids.
6. Its performance on the collective set of presented measures is an improvement over the other data structures.

3 Initial Pentium 4 Performance and the BART benchmark.

Table 3 below includes timing performance of the DE-tree based FORT on a 1.8 GHz Pentium 4 machine with 256 MBytes of PC2100 (DDR 266) RAM, and 512 KBytes of L2 cache memory. GCC 3.4.3 with optimization flags "-03 -mfpmath=sse -mmmx -msse' was used for the "GCC" runs; IBM JDK 1.4.2 with appletviewer arguments "-J-Xms100M -J-Xmx100M -J-ea" was used for the Java runs. The C++ software is identical to that used for generating the data of tables 1 and 2 except for the correction of one bug of small significance to performance; the method of running FORT is also indentical. The data for scenes in common with the earlier results in tables 1 and 2 indicate that the auspicious early results are not coincidentally limited to Pentium III systems, a particular compiler, not are they do to unusual memory management and hacks that may be performed in "C" style languages. There are extra cache misses and unavailable optimization in the Java version do to the lack of memory contiguity in 3D arrays and other objects as only 100% standard JSE 1.4 software was used (e.g. real-time java specification compliant software was not used).

Table 3 includes data for the museum7 scenes of BART (Benchmark for Animated Ray Tracing [Lext00]). The reported data is the average over all 300 time steps of the animation. The performance for this animation is also known for the leading ray tracing rendering system OpenRT, and [Wald02] and [Wald04] report that OpenRT renders this animation at an average of 0.45 frames/second on a dual AMD Athlon 1800+ System. This corresponds to about 4.4 seconds for one CPU - or about 20 percent faster than FORT's time of 5.41 seconds. At this stage of development FORT is not using the ray coherence nor the SSE optimized ray-triangle intersection tests introduced by Wald et all ([Wald01] and used in OpenRT, but the dramatic improvements that can be obtained from these techniques will likely force their incorporation into future versions of FORT.

Performance on the museum8 scene is coincidentally reported for a recent animation system by Larson and Akenine-Möller [Lars03]. Though their reported rendering time without using image subsampling is is more than four times slower than what we report for FORT, their search tree's reconstruction time is roughly an order of magnitude faster than FORT's 0.17 seconds tree creation time. This advantage is slightly dampened by the fact that they must prepare in advance five copies of the search tree. Nontheless, as both their system and FORT are bounding volume higherarchies, they have demonstrated that it is at least worth wile investigating tree reconstruction techniques for certain kinds of animations, and related techniques are being considered for future versoins of the DE-tree and FORT.

3.1 Symbols for Table 3.

Tb - the time (in seconds) to build the RSA.
Tr - the time (in seconds) to perform all ray tracing calculations once the RSA is created.
Nts - the average number of nodes accessed per ray.

3.2 Table 3 – Pentium IV Performance of FORT on the SPD and BART museum.

Scene	Compiler	Nts	Tb	Tr
SPD Balls5	GCC	30.47	0.08	3.51
SPD Teapot40	GCC	28.79	0.11	2.34
SPD G5	GCC	32.79	0.12	3.19
BART Museum7	GCC	52.16	0.03	5.41
BART Museum8	GCC	57.34	0.17	6.15
SPD Balls5	IBM Java 1.4.2	30.66	0.42	6.54
SPD Teapot40	IBM Java 1.4.2	28.85	0.48	4.29

4 Initial collision detection and dynamic box query performance.

The DE-tree data structure also supports box queries without any modification to the node structure or even the tree topology as used for ray queries. Consequently, even the same *instances* of the data structure can be used not just for visibility determination and ray tracing, but also collision detection and related calculations.

For collision detection we have not found published benchmarks that are as encompassing, conclusive, or widely used as the SPD is for ray tracing, so currently we are reproducing some published experiments that appear to be widely cited while also being recent and could have some claim as still being the "state-of-the-art".

Table 4 has data which allows for comparisons to the maximally traverse spheres experiments described in chapter 5 of the thesis of Stefan Gottschalk [Gott00]. In these experiments two tesselated spheres of identical size but random orientation are incrementally translated in space and all contact points (all of the intersecting triangles) are reported. The data in our Table 4 can be compared to information presented in section 5.3.3 of that thesis, and in particular to the the data in figures 5.21, 5.22 and 5.23 of the same thesis. The plot of figure 5.23 indicates the contact cost in terms of (bounding volume tests / contacts) for three compared data structures. This cost appears constant as a function of the number of polygons, and from the graph we estimate them to be approximately eighteen, sixty-two, and sixty-five for the OBB-tree, the AABB-tree, and the sphere-tree respectively. In the current version of the DE-tree, such a measure does not exactly exist, but we know that the numerical cost to visit one DE-tree node during box queries is less than the cost of determining one AABB bounding volume test, and since the number of nodes visited by the DE-tree is 42.45 for the largest scene, the DE-tree's performance is likely to be better than the AABB-tree. The OBB-tree has less bounding volume tests than the DE-tree has node visitations, but the cost of its tests are even higher than the corresponding costs in AABB-trees. Finally, it is possible reduce the computational complexity of box-queries with the DE-tree by not using our current method of performing one box query at a time, but instead using the Bounding Volume Test Tree used by [Gott00] with the AABB-tree, the OBB-tree, and the sphere-tree. [1].

4.1 Table 4 – Box query performance in the maximally traverse spheres experiments.

Np	Nq	Nc	Nb	Nnv	Nlv	Anvc	Anvf	Tb	Ts	Tu
1008200	1008200	2657	3689	3109475	18024	42.45	2.98	1.37	0.17	0.17
524288	524288	1956	2726	1646612	13168	40.63	3	0.67	0.1	0.19
131072	131072	1022	1351	434461	6641	37.63	3.04	0.16	0.02	0.15
32768	32768	413	693	117518	3346	35.81	3.18	0.04	0.01	0.31

4.2 Symbols for Table 4.

Np - The number of triangles in each of the two tesselated spheres.
Nq - The total number of box queries.
Nc - The total number of distinct contacts found.
Nb - The total number of contacts.
Nnv- The total number of tree nodes visited.
Ancv - The average number of tree nodes visited per contact-finding query.
Anvf - The average number of tree nodes visited per query not finding contacts.
Tb - The time in seconds required to build the search tree.
Ts - The total search time in seconds for all Nq box queries.
Tu - The average box query time in microseconds.

[1]The Bounding Volume Test Tree appears to be similar to the presented in [?], and thus we have another alternative structure to do this with

References

[Hurl02] J. Hurley, A. Kapustin, A. Reshetov, and A. Soupikov, “Fast Ray Tracing for Modern General Purpose CPU.” *Proc. Graphicon 2002.*

[Hav00A] Vlastimil Havran, ”Heuristic Ray Shooting Algorithms.” Ph.D. Thesis, Faculty of Electrical Engineering, Czech Technical University, Prague. November 2000.

[Hav00B] V.Havran and W. Purgathofer, ”Comparison methodology for ray shooting algorithms.” Tech. report TR-186-2-00-20, Vienna University of Technology, Nov. 2000.

[Havr02] V.Havran and J.Bittner, ”On Improving kd-Trees for Ray Shooting.” *Journal of WSCG*, 10(1), pp. 209–217, February 2002.

[Lext00] Jonas Lext, Ulf Assarsson, and Tomas Möller, “BART: A Benchmark for Animated Ray Tracing”, Technical Report, Dept. of Computer Engineering, Chalmers University of Technology, Götteorg, Sweden, May 2000. Available at http://www.ce.chalmers.se/BART.

[Hain87] Eric A. Haines, ‘A proposal for standard graphics environments.”, *IEEE CG&A*, 7(11):3-5, Nov. 1987. Available at www.acm.org/pubs/tog/resources/SPD.

[Wald01] Ingo Wald, Carsten Benthin, Markus Wagner and Philipp Slusallek, ”Interactive Rendering with Coherent Ray Tracing.” *Proc. EUROGRAPHICS 2001*, Alan Chalmers and Theresa–Marie Rhyne, editors. Blackwell Publishers, Oxford.

[Wald02] Ingo Wald, Carsten Benthin, and Philipp Slusallek, ”A Simple and Practical Method for Interactive Ray Tracing of Dynamic Scenes”, Eurographics Symposium on Rendering (2004), W.H.Jensen and A. Keller, editors.

[Wald04] Ingo Wald, “Realtime Ray Tracing and Interactive Global Illumination”, PhD Thesis, Saarland University, 2004.

[Lars03] Thomas Larsson and Tomas Akenine-Möller, “Strategies for bounding volume hierarchy updates for ray tracing of deformable models.”February 21, 2003.

[Lars03B] Thomas Larsson and Tomas Akenine-Möller, “Efficient collision detection for deformed models by morphing”,The Visual Computer,February 2003.

[Bergen] Gino Van Den Bergen, “Efficient collision detection of complex deformable dodels using AABB trees”, J. Graphics Tools (2(4), 1997.

[Hubb96] P.M. Hubbard, “Approxiamting polyhedra with spheres for time-critical collision detection.” ACM Transaction on Graphics, 13(3), July 1996.

[Gott00] Stefan Gottschalk, “Collision Queries using Oriented Bounding Boxes”, PhD Thesis, Univ. of North Carolina at Chapel Hill, 2000.

GOLEM homepage: www.cgg.cvut.cz/GOLEM.
FORT homepage: www.newgs.com/FORT.
OpenRT homepage: www.OpenRT.de.