Benchmarking for Ray Tracing Performance Measurement

Kroeze, JCW, Nort West University, Vanderbijlpark, South Africa, thejcwk@gmail.com
Jordaan, DB, Nort West University, Vanderbijlpark, South Africa, dawid.jordaan@nwu.ac.za
Pretorius, P, North West University, Vanderbijlpark, South Africa, phillip.pretorius@nwu.ac.za

Abstract

During the past thirty years, researchers have greatly sped up the naïve ray tracing algorithm. However, the methods used to measure and compare their diverse algorithms have remained *ad-hoc*. This makes it difficult to gauge the effectiveness of one author's approach versus that of another. Clarity, transparency and the reproducibility of experiments are foundational to proper science, therefore these discrepancies must be addressed. This paper will discuss a variety of sources from the literature to show the flaws in the current research paradigm. It will then propose a new method based on a benchmarking approach. Finally, the results of experimentation with the approach will be discussed and a conclusion will be drawn as to its applicability.

Keywords
Ray Tracing, Computer Graphics, Benchmarking, Performance Measurement.

Introduction
The naïve ray tracing algorithm was first formulated by Turner Whitted (Whitted, 1980). It offered much greater visual realism than the rasterization algorithm, but struggled with performance issues.

In the intervening thirty years, much research has been conducted on ways to correct this problem with the algorithm. Researchers have succeeded in greatly reducing the time needed to ray trace a scene. In recent years, even real-time and interactive ray tracers have seen the light.

However, one aspect of the research remains troubling: the wide variety of experimental methods in use. It is seldom the case that two papers will measure their results in quite the same way. Different scenes, viewpoints, metrics and hardware is used from paper to paper. This makes the comparison of results very difficult and ends up confusing the reader. Since these algorithms are not compared on equal footing, it is hard to see which offer the best, or worst performance. This, in turn, makes it difficult to pick an algorithm for implementation and could be a factor in the lack of acceptance of new ideas in the community.

As scientists, the research community formed around ray tracing should seek to minimize the variables used in an experiment, until ultimately the only variable is the algorithm in use.

This paper will make a contribution to this aim. It will start with a brief discussion of results previously obtained by the author by studying the literature of a specific branch of ray tracing research: the GPU ray tracing approaches.

These results will then be strengthened by taking a sample of papers from a different area of ray tracing research and observing that the same problems are present.

Finally, a solution to the problems discussed here will be presented and tested. This paper will conclude with a discussion about the applicability of the proposed solution to future ray tracing research.

Previous Work

	Carr *et al.* (2002	**Purcell *et al.* (2002)**	**Buck *et al.* (2004)**	**Foley & Sugerman (2005)**	**Carr *et al.* (2006)**	**Huang *et al.* (2006)**
Performance Metric	Rays / second.	SIMD efficiency, traversal steps and intersections.	Ray / triangle intersections per second.	Elapsed milli-seconds and various traversal counts.	Elapsed milli-seconds.	Rays / seconds and inter-sections / ray.
Scenes	"Teapot room", "office" and "soda hall".	"Soda hall", "forest" and "bunny".	"Glassner"[i]	"Robots", "kitchen", "Cornell box" and "Stanford bunny".	"Stanford bunny" and "Mult."	"Desk", "cube", "teapot", "bear", "venus", "simplified bunny", "approximate bunny", "teapot house" and "bunny couple".
GPU	Radeon 8500 / GeForce 3 / GeForce 4 Ti4600	Not stated.	Radeon X800 XT Platinum / GeForce 6800 (Pre-release)	256 MB ATI X800 XT PE	GeForce 7800 GTX (430 MHz clock, 1.2GHz memory clock)	256 MB NVIDIA 6800GT
CPU	Not stated.	Not stated.	3 GHz Pentium 4 (875P Chipset)	Not stated.	2.2 GHz Athlon 3500+	2 x 3.2 GHz Pentium 4
Memory	Not stated.	Not stated.	Not stated.	Not stated.	Not stated.	2 GB
	Horn *et al.* (2007)	**Chen & Liu (2007)**	**Popov *et al.* (2007)**	**Zhou *et al.* (2008)**	**Aila & Lane (2009)**	**Kalojanov & Slusallek (2009)**
Performance Metric	Frames per second and millions of rays / second.	Elapsed seconds and percentage speed-up.	K-d tree statistics, traversal steps and frames per second.	Elapsed seconds and frames per second and speed-up factor.	SIMD efficiency, millions of rays / second and percentage of simulated performance.	Frames per second and milliseconds.

Scenes	"Cornell box", "kitchen", "robots" and "conference".	"Bunny", "dragon" and "easter".	"Shirley6", "bunny", "forest" and "conference".	"Toys", "museum", "robots", "kitchen", "fairy forest" and "dragon".	"Conference", "fairy" and "Sibenik".	"Thai statue", "soda hall", "conference", "dragon", "fairy forest", "sponza", "ruins".
GPU	512 MB Radeon X1900 XTX (650 Mhz clock & 750 Mhz memory clock)	Radeon X300SE	GeForce 8800 GTX	768 MB GeForce 8800 ULTRA	GeForce 285 GTX	1 GB GeForce 280 GTX
CPU	2 x 2.4 GHz Core2 Duo	1.8 GHz Athlon64 3000+	2.6 GHz Opteron	3.7 GHz Xeon	Not stated.	4 x 2.66 GHz Core2 Quad
Memory	Not stated.	Not stated.	Not stated.	Not stated.	Not stated.	Not stated

Table 1: Variation in experimental methods in the GPU ray tracing community (Kroeze, Jordaan & Pretorius, 2010).

Table 1 illustrates the wide variation in experimental methods encountered during a survey of the literature on GPU ray tracers.

Several different hardware platforms are used across the different papers. This is largely due to the increase in hardware performance over the years from 2002 till 2009. However, how can meaningful comparisons be made between algorithms when the hardware is so diverse?

In many cases the hardware configuration used for testing isn't even completely specified. It is a rarity that the size of the primary memory on the test computer is reported, and its bandwidth is never mentioned.

A computer's performance is a complex and subtle thing, it is difficult to describe its relative performance by simply stating the graphics card in use.

In the interest of comparability and repeatability, the configuration used for testing should at least be stated carefully and ideally should be eliminated as a variable during performance testing.

The scenes used for testing are also not standardized, despite there being proposals for two standard sets (Haines, 1987; Lext, Assarsson & Möller, 2001). Since the amount of reflectivity, refractivity, polygon count and the distribution of objects in a scene can affect the performance of a ray tracer, the scenes used to test a ray tracer's performance should be a standard suite with a standard set of viewpoints or animations.

The lack of standardization concerning performance metrics also raises questions. Confusion can easily arise when one paper measures rays traced per second, another measures SIMD efficiency and another measures frames per second. It is at least difficult and at worst impossible to convert between the different performance metrics used.

In order to make experiments comparable and reduce confusion there should be one standard measure of ray tracing performance that each paper can easily use.

Other Ray Tracing Research

The previous section discussed papers published on the topic of GPU ray tracing and highlighted some of the flaws in the current research paradigm. While these results are interesting, they are only applicable to one

subfield of the ray tracing research community. In the interest of fairness, this section will sample six papers from various other fields in the wider ray tracing research community. This will help to determine whether the same problems are present in the wider context of ray tracing.

	Georgiev & Slusallek (2008:121)	**Overbeck, Ramamoorthi & Mark (2008:45 & 47)**	**Wald & Slusallek (2001)**
Performance Metric	Frames per second.	Frames per second & performance benefit.	Frames per second & microseconds per primary ray.
Scenes	"Sponza", "conference" and "soda hall".	"ERW6", "toasters", "fairy" and "rings".	"MGF office", "MGF conference", "MGF theater", "Library", "Soda Floot 5" and "Soda Hall".
GPU	Not stated.	Not stated.	NVIDIA GeForge II GTS
CPU	2 x 2.6 GHz Core2 Duo[ii]	8 x 2.0 GHz Intel Xeon	800 MHz Pentium III[iii]
Memory	Not stated.	Not stated.	256 MB
	Wald, Mark, Günther, Boulos, Ize, Hunt, Parker & Shirley (2007)	**Havran, Herzog & Seidel (2006)**	**Wächter & Keller (2006)**
Performance Metric	Frames per second.	Time to construct the data structure and render the scene along with several other metrics.	Frames per second and milliseconds required to render.
Scenes	"ERW6", "conference", "soda hall", "toys", "runner" and "fairy".	"Conference", "bunny", "armadillo", "dragon", "buddha", "blade", "robots", "museum" and "kitchen".	"Shirley Scene 6", "Dragon", "Buddha", "Kitchen", "Conference", "Bunny", "Car 1" and "Blender Suzanne", BART scenes and "UTAH Fairy Forest".
GPU	Not stated.	Not stated.	Not stated.
CPU	2.6 GHz Opteron	2 x 3800+ AMD Athlon 64[iv]	2.8 GHz Pentium 4HT / 2.6 GHz Opteron[v]
Memory	Not stated.	Not stated.	Not stated.

Table 2: Performance metrics, scenes and hardware used in six ray tracing papers.

Note that the information in table 2 is for the sections of the papers discussing total performance measurements only. Other sections dealing with the performance of specific elements and the like have been ignored.

The papers covered in table 2 are much more homogenous with regards to the performance metrics used. Only one paper (Havran *et al*., 2006) uses a different metric than frames per second. Since these papers are relatively recent, the dominance of a single metric is an encouraging sign that the research community is settling on a common performance metric. This will greatly simplify the comparison of results between papers.

The hardware used is also more homogenous – all of the CPUs (save one) are in the 2.0 GHz to 3.0 GHz range. Unfortunately, the amount of cores in use ranges from one to eight. The single paper from 2001 that uses a slow single-core 800 MHz processor is a reminder that performance metrics should be hardware independent in order to remain relevant into the future and allow researchers with access to poor hardware to fairly represent their findings.

Since the ray tracing algorithm is inherently very parrallel, the amount of cores available is a very important variable. Since the research in question focuses on finding better ray tracing algorithms and not better ray tracing

CPUs, it would be ideal to eliminate this variable from the experiments.

It is also troublesome that the GPU used and the memory installed on the testing platforms is not mentioned at all. Every component in a computer has some effect on its eventual performance. The descriptions of the CPU alone that is often encountered in the literature may be too simple to give an accurate idea as to the performance of the testing platform. Ideally, a performance metric would ignore these complicated issues entirely, making life easier for the researchers involved while also providing a clearer idea about the performance of a given algorithm.

These goals are hard to achieve, since the efficiency of a given computer is a very complex thing to measure or standardize. It would also be impractical to require researchers to use a single, pre-determined hardware platform for their measurements. As such a platform would age, its components would eventually be hard to come by and many researchers would balk at the extra effort it would entail.

Another solution to the problem is needed, one which any researcher can easily implement and that would give consistent and clear results. The next section discusses a possible solution that would meet these criteria.

Solution

Since it would be impossible to standardize the platform used to measure the performance of ray tracing algorithms the performance of the computer performing the tests must be measured and used to characterise the results obtained.

This process is commonly known as “benchmarking” and many commercial programs exist that are able to characterise a computer's performance with a single standard number.

This approach hides all the performance intricacies by showing a single number indicating how fast a testing platform is.

There is a question about which benchmark to use, however. Since the problem domain is already narrowly defined, however, it would be fitting to use a ray tracer as this benchmarking program. It is a logical assumption that most ray tracers have similar performance profiles on the same hardware, since they perform the same type of operations. At least, these performance profiles are likely to be more similar than that of some other benchmark program.

The solution suggested by this paper is to use a simple, unoptimized ray tracer to determine how good a particular testing platform is at the task of ray tracing. An adapted, optimized ray tracer can then be run. The difference between the two can then be measured as a percentage “speed up” obtained. This can then be used with other algorithms to compare them on an equal footing.

This approach should eliminate as many variables as possible with one simple process. The rest of this paper will focus on the development and testing of the proposed solution.

Development

The proposed benchmarking program can be pretty simple. Its only function is to provide an accurate baseline for comparison of algorithms on disparate machines.

However, the program must be representative of the types of operations performed by a ray tracer. This means that it should implement the majority of features provided by ray tracers such as perfect specular reflection, refraction and shading.

Since most ray tracers are implemented in C++, the benchmarking program should make use of this language as well to facilitate fair comparisons.

There are several sets of standard scenes for testing ray tracers. Haines (1987) provides the standard procedural library that has several features designed to test ray tracers. Lext, Assarsson and Möller (2001) expand his file

type definition to include animated scenes and provide several examples that are being used to test ray tracers.

The benchmark program discussed in the paper will implement the neutral file format (NFF) defined by Haines (1987) and use the standard procedural library as test scenes. These were picked for the file format's relative simplicity and the care put into the design of the scenes.

Implementation of parsers for other popular file formats and scenes is left as future work and considered beyond the scope of this paper.

The finalized benchmark program will therefore be a very simple ray tracer written in the C++ programming language. It will simulate a pinhole camera in order to generate a set of rays – no depth of field effects will be added. Each pixel in the final image will get its colour value from a single ray – no anti-aliasing will be performed. Each of these rays will interact with the scene in a physically correct way. In this way they collect information on the light that is reflected on the surface of certain objects, or refracted through them. This information is then propagated through the scene by tracing rays through their reflections and refractions. Finally, this information will then be used to calculate the correct colour for each pixel. The benchmark program will not be optimized in any way, except for the "optimized benchmark" runs where it will only use a simple bounding box optimization.

Limitations

Despire the author's best efforts there are some bugs that remain in the benchmark program. These relate to refraction calculations and shadow rays, leading to some anomalies on the "mount" and "gears" scenes. These errors may be seen by inspecting the additional material accompanying this paper, which contains the "correct" images and the images that the benchmark program generate for each test run.

These errors should not affect the quality of the results, since the calculations are still being performed, just slightly incorrectly. However, this is still a point where the results of this study may be improved.

Testing

The proposed solution was tested in order to determine its suitability as a benchmarking application. The measure of its performance will be the degree to which the metric it provides remains constant across hardware platforms. In order to measure this consistency the r^2 metric will be calculated to determine the correlation between test runs on different platforms.

To this end, the benchmarking program will be executed on several systems to gather the appropriate data. Then, a popular open source ray tracer will be executed on the same system, along with a slightly optimized version of the benchmarking program. The percentage performance increase over the benchmark program will be recorded for each of these latter two programs. The systems the benchmark will be executed on appear below:

	Straylight	**Neolith (Laptop)**	**Monolith**
CPU	4 x 2.6 GHz Intel i5 750	2 x 1.6 GHz Intel Core2 Duo T5500	1 x 3.4 GHz Intel Pentium 4
CPU Cooler	Thermalright MUX-120	Stock	Stock
GPU	NVIDIA GT250	Intel Mobile GM965	ATI Radeon 9600
Memory	4 GB DDR3 1333 MHz	1 GB DDR2 667 MHz	1 GB
Hard Drive	1 TB Seagate SATA 3GB/s	120 GB Seagate Momentus SATA / 1. GB/s 5200 RPM	160 GB Seagate Barracuda Ultra ATA / 100MB 7200 RPM

Table 3: Hardware installed on the three testing systems.

Each of the scenes from the SPD will be rendered by each of the programs. Data will be provided for each such run.

To ensure that there is the minimum resource contention with other processes that might be running on the test systems, the tests will be executed only on Linux systems booted into maintenance (init level 1) mode. In this mode, only the minimum programs are launched, making context-switches and disk contention much less likely.

The benchmark program therefore serves as a type of baseline – its execution time provides the time a vanilla ray tracer will take to render each of the scenes. Each optimized ray tracer can then render the same scenes, on the same hardware and under the same conditions. This provides a percentage performance increase figure that should remain constant across hardware platforms. This should be the case, because comparing the optimized ray tracers with the benchmark eliminates hardware as a variable, since the actual performance of the hardware for the task of ray tracing is captured as the run time for the benchmark program.

Data Analysis

After gathering the test data from each run of the program, they will be related to each other in order to test their level of hardware independence. The run for the basic ray tracer serves as a base-line or a benchmark, while the run for its optimized version serves as a control.

If the hypothesis of this paper is correct, then dividing the running time for each scene by the running time for the benchmark will be more-or-less constant across platforms, i.e. hardware independent.

Results

The results of the data analysis are shown in the tables below. While the performance profiles for the related runs look very similar, there are a couple of anomalies where there is a larger difference between the ratio on one computer and the ratio on another. In general, this is a couple percentage points for the runs comparing the optimized version of the benchmark versus the benchmark itself. Note that these ratios have been scaled to percentages.

Presumably, this is because these runs took much longer to complete than the others. Another hypothesis to test would be whether the error margin increases as the running time of a test does.

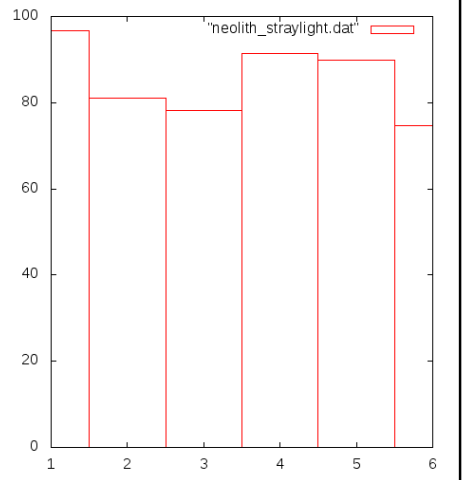

Figure 2: Runtime for optimized benchmark program divided by runtime for benchmark program on the "neolith" platform.

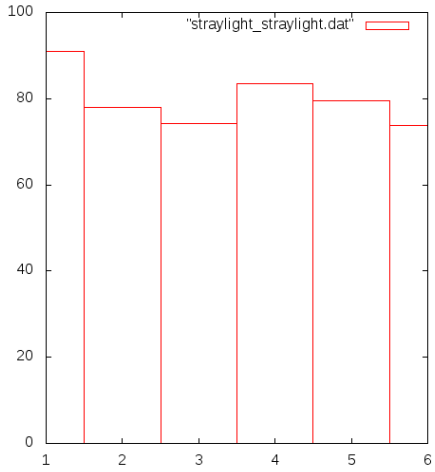

Figure *1*: Runtime for optimized benchmark program divided by runtime for benchmark program on the "straylight" platform.

For the tests comparing POVRay to the benchmark, the error margin is significantly lower – only a fraction of a percentage point at most.

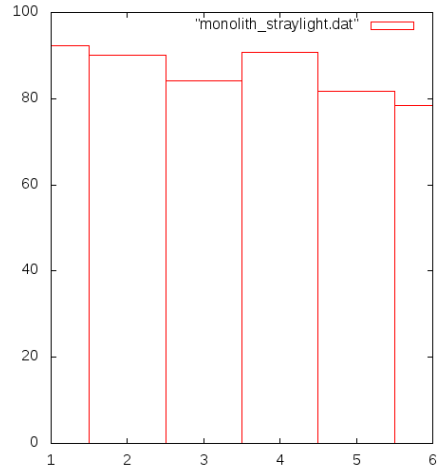

Figure 3: Runtime for optimized benchmark program divided by runtime for benchmark program on the "monolith" platform.

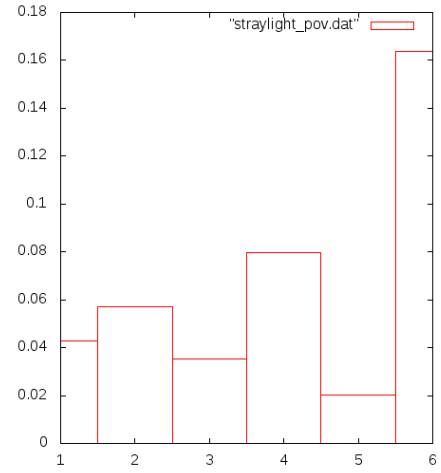

Figure 4: Runtime for POVRay divided by runtime for benchmark program on the "straylight" platform.

The correlation coefficient for the different runs of the optimized benchmark versus the benchmark is 0,8696 between “neolith” and “straylight”, but is only 0,5658 when comparing “monolith” to “straylight”.

It is possible that this discrepancy is caused by the old hardware present in “monolith”. The latter computer only has a Pentium 4, while the other platforms had 64-bit capable, multi-core CPUs. It could be the case that on the longer experimental runs this causes a divergence.

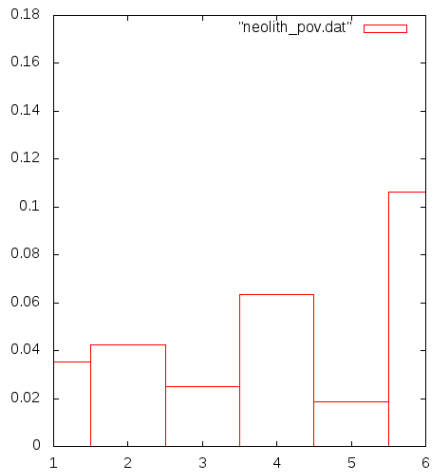

Figure 6: Runtime for POVRay divided by runtime for benchmark program on the "neolith" platform.

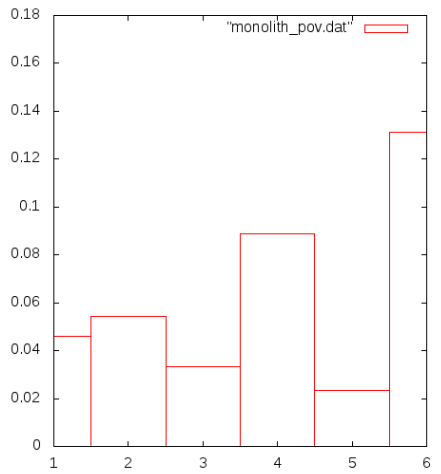

Figure 5: Runtime for POVRay divided by runtime for benchmark program on the "monolith" platform.

The correlation coefficents for the run comparing the benchmark to POVRay fares much better: it is 0,9857 when comparing “neolith” and “straylight”, and 0,9581 when comparing “monolith” and “straylight”.

The correlation coefficient for “monolith” is again lower, although not by much. This supports the theory that the longer the experiment runs – the greater the effect of the discrepancy in hardware.

This is unlikely to be a problem in practise, however. Most ray tracers that are discussed in the literature have much shorter run times than the optimized benchmark for which the correlation coefficients are poor. They are much more likely to behave like POVRay, for which the coefficients are much better.

Conclusions

In conclusion, the results from this study are very promising. At least in the case of faster ray tracers, the benchmark is quite effective at factoring out hardware as a variable in experiments.

Unfortunately, it seems that large hardware discrepancies can still produce unacceptable error margins and low correlation coefficients. Since this problem is contained only to the slower running control experiment in this paper, it does not invalidate the hypothesis.

Hopefully a technique similar to this one can be used in the future to facilitate ray tracing research by providing a hardware independent way of comparing research results.

Future Research

While the results from this study are promising, they are still very much preliminary. Unfortunately, the author only had access to the three computers used in this study. Further experiments on other computers and more varied platforms would serve to either strengthen the conclusion of this paper, or falsify its findings.

As mentioned previously, there are still a number of bugs in the benchmark program to be sorted out. Fixing these issues and noting the effect they have on accuracy will also be worthwhile research to undertake.

Finally, the benchmark program is very limited as to the range of scenes it can load, render and therefore use as comparison. Extending the range of file formats it can handle will go a long way toward providing a standardised platform for ray tracing research

Final Notes

The code used for the benchmark program, as well as all data used, all programs used to organise the data and any other supplementary information may be downloaded from: ftp://philosoraptor.co.za/IBIMA for the purposes of careful review and openn

i It is unclear whether any other scenes were used. However, this scene name is mentioned on page 780 and the performance graphs suggest that only one scene was used.

ii One of the algorithms was tested on a different hardware platform, although the authors of the paper do not provide specifics (Georgiev & Slusallek, 2008:121).

iii The researchers also used the SGI Onyx-3 and the SGI Octance graphics supercomputers for rasterization performance figures (Wald & Slusallek, 2001).

iv While the hardware supported symmetric multiprocessing the authors chose not to take advantage of it (Havran *et al.*, 2006).

v The research contains reference to previous work done on the latter processor (Wächter & Keller, 2006).

References

Aila, T and Laine, S. (2009). 'Understanding the efficiency of ray traversal on GPUs'. Proceedings of the Conference on High Performance Graphics 2009, ISBN: 978-1-60558-603-8, 1-3 August 2009, New Orleans, LA, 145-149.

Buck, I., Foley, T., Horn, D. Sugerman, J., Fatahalian, K., Houston, M. and Hanrahan, P. (2004). 'Brook for GPUs: stream computing on graphics hardware'. ACM SIGGRAPH 2004 Papers, 8-12 August 2004, Los Angeles, CA, 777-786.

Carr, N.A., Hall, J.D. and Hart, J.C. (2002). 'The ray engine'. Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, ISBN: 1-58113-580-7, 1-2 September 2002, Saarbrucken, Germany, 37-46.

Carr, N.A., Hoberock, J., Crane, K. and Hart, J.C. (2006). 'Fast GPU ray tracing of dynamic meshes using geometry images'. Proceedings of Graphics Interface 2006, ISBN: 978-1-56881-308-0, 7-9 June 2006, Québec, Canada, 203-209.

Chen, C.C. and Liu, D.S.M. (2007). 'Use of hardware z-buffered rasterization to accelerate ray tracing'. Proceedings of The 2007 ACM Symposium on Applied Computing, ISBN: 1-59593-480-4, 11-15 March 2007, Seoul, Korea, 1046-1050.

Foley, T. and Sugerman, J. (2005). 'Kd-tree acceleration structures for a GPU raytracer'. Proceedings of the ACM SIGGRAPH / EUROGRAPHICS Conference on Graphics Hardware, ISBN: 1-59593-086-8, 30-31 July 2005, Los Angeles, CA, 15-22.

Georgiev, I. and Slusallek, P. (2008). 'RTfact: generic concepts for flexible and high performance ray tracing'. Proceedings of the 2008 IEEE Symposium on Interactive Ray Tracing, ISBN: 978-1-4244-2741-3, 9-10 August 2008, Los Angeles, CA, 115-122.

Haines, E. (1987). 'A proposal for standard graphics environments'. *IEEE Computer Graphics and Applications*, 7(11), 3-5.

Havran, V., Herzog, R. and Seidel, H.P. (2006). 'On the fast construction of spatial hierarchies for ray tracing'. Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing, ISBN: 1-4244-0693-5, 18-20 September 2006, Salt Lake City, UT, 71-80.

Horn, D.R., Sugerman, J., Houston, M. and Hanrahan, P. (2007). 'Interactive k-D GPU raytracing'. Proceedings of the 2007 Symposium on Interactive Ray Tracing, ISBN: 978-1-59593-628-8, 30 April – 02 May 2007, Seattle, WA, 167-174.

Huang, P., Wang, W., Yang, G. and Wu, E. (2006). 'Traversal fields for ray tracing dynamic scenes'. Proceedings of The ACM Symposium On Virtual Reality Software And Technology, ISBN: 1-59593-321-2, Limassol, Cyprus, 65-74.

Kalojanov, J. and Slusallek, P. (2009). 'A parallel algorithm for construction of uniform grids'. Proceedings of the Conference on High Performance Graphics 2009, ISBN: 978-1-60558-603-8, 1-3 August 2009, New Orleans, LA, 23-28.

Kroeze, J.C.W., Jordaan, D.W. and Pretorius, P. (2010). 'The Advent of GPU Ray Tracing'. Proceedings of IBIMA 15, ISBN: 978-0-9821489-4-5, 6-7 November 2010, Cairo, Egypt.

Lext, J., Assarsson, U., Möller, T. (2001). 'A benchmark for animated ray tracing'. *IEEE Computer Graphics and Applications*, 21(2), 22-31.

Overbeck, R., Ramamoorthi, R. and Mark, W.R. (2008). 'Large ray packets for real-time whitted ray tracing'. Proceedings of the 2008 IEEE Symposium on Interactive Ray Tracing, ISBN: 978-1-4244-2741-3, 9-10 August 2008, Los Angeles, CA, 41-48.

Popov, S., Günther, J., Seidel, H.P. and Slusallek, P. (2007). 'Stackless KD-tree traversal for high performance GPU ray tracing'. *Computer Graphics Forum,* 26(3), 415-424.

Purcell, T.J., Buck, I., Mark, W.R. and Hanrahan, P. (2002). 'Ray tracing on programmable graphics hardware'. *ACM Transactions on Graphics*, 21(3), 268-277.

Wächter, C. and Keller, A. (2006). 'Instant ray tracing: the bounding interval hierarchy'. Proceedings of the Eurographics Symposium on Rendering, ISBN: 3-905673-35-5, 26-28 June 2006, Nicosa, Cyprus , 139-149.

Wald, I. and Slusallek, P. (2001). 'State of the art in interactive ray tracing'. STAR Proceedings of Eurographics 2001, 3-7 September 2001, Manchester, UK.

Wald, I., Mark, W.R., Günther, J., Bolous, S., Ize, T., Hunt, W., Parker, S.G. and Shirley, P. 2007. 'State of the art in ray tracing animated scenes'. STAR Proceedings of Eurographics 2007, ISSN: 1017-4656, 3-7 September 2007, Prague, Czech Republic, 89–116.

Whitted, T. (1980). 'An improved illumination model for shaded display'. *Communications of the ACM*, 23(6), 343-349.

Zhou, K., Hou, Q., Wang, R. and Guo, B. (2008). 'Real-time KD-tree construction on graphics hardware'. ACM SIGGRAPH Asia 2008 Papers, ISSN:0730-0301, 10-13 December 2008, Singapore, Article #126.