Department of Computer Science,
University College London,
Gower Street,
London WC1E 6BT.

A Visibility Field for Ray Tracing

Jesper Mortensen Mel Slater Pankaj Khanna Insu Yu
(j.mortensen | m.slater | p.khanna | i.yu) @cs.ucl.ac.uk

Department of Computer Science, University College London, Gower Street, London WC1E 6BT

Abstract

This paper exploits a type of visibility data structure, similar to a Virtual Light Field, for accelerated ray tracing. The data structure supports constant time ray access to a very small but conservative potential visibility set (PVS) of the original surfaces in the scene; only these have to be searched for a final intersection determination. The visibility field employed is constructed by choosing a regular point subdivision over a hemisphere, to obtain a set of directions, and then corresponding to each direction there is a rectangular grid of parallel beams. Each beam references a set of identifiers corresponding to objects that intersect it. The beam corresponding to any ray can be looked up in small constant time, and the set of objects corresponding to the beam can then be searched for intersection with the ray. This final step can be carried out with any conventional ray traversal technique, and in particular we use a BSP tree, but only with partitioning planes perpendicular to the direction of the beam. This approach trades off rendering speed for significant memory usage and pre-processing time. An early implementation of this technique is described and initial results for a specific class of scenes are presented and shown to compare favorably with an approach that has been around for quite a while, the single-ray Coherent Ray Tracing approach of Wald and Slusallek et al.

1. Introduction

There have been significant advances towards real-time ray tracing in recent years, through the exploitation of algorithms that are tailor made to perform well on today's graphics hardware [PKG97] [CHH02] [WSB*03] [PBMH02]. We present initial results on a novel strategy to reduce ray-object intersection traversal time. Every ray tracing algorithm has to deal with the problem of ray-object traversal – that is, to find for any ray the nearest surface that it intersects. This problem has received a great deal of attention since the introduction of ray tracing into computer graphics [App68] [Whi80]. All successful methods rely on a data structure that when traversed by a ray, delivers a set of objects, the candidate set, or the potentially visible set, that the ray may intersect. In this paper we present a modification of this standard approach. We exploit a 4-dimensional data structure, which is a kind of light field that instead of storing radiance stores object identifiers. The data structure is a special instance of a 'virtual light field' (VLF) [Ano04], that we call VLF-RT in this paper. A ray is used as an array look-up index into the VLF-RT data structure, and immediately delivers a set of candidate objects for ray-object intersection testing. That set of objects, a tiny fraction of the original number in the scene, may be traversed linearly or by any other method.

We discuss the background literature and state-of-the art in Section 2. The VLF data structure for ray tracing is presented in Section 3. In Section 4 we motivate and describe the use of BSP trees for the final traversal of the potentially visible set (PVS) returned from the ray lookup into the VLF-RT. Implementation details are given in Section 5. Results in the form of comparisons of Coherent Ray Tracing (CRT) and the VLF-RT method are presented in Section 6, and conclusions in Section 7, including a discussion of how dynamic scene changes are very simple in this method. In this paper we concentrate only on 'classical' ray tracing as described by Whitted [Whi80]. For compatibility with CRT all our examples are limited to polygons, and in our main results, triangles. However, the method is not bound to polygonal objects.

2. Background

Ray tracing was the first type of global illumination algorithm introduced into computer graphics [Whi80]. It very simply and elegantly supports shadows, specular reflection and transmission, and also solves the problem of visibility. It does not correctly handle light paths that involve both diffuse and specular reflections, and this will not be considered in this paper. The overall benefits of ray tracing have been discussed many times, for example [Gla89] for a general overview and standard algorithms, and [WSBW01] for potential benefits as compared to the standard graphics pipeline. In the original paper Whitted pointed out that the vast amount of the time to produce a ray traced image is taken up by ray-object intersection calculations.

Many techniques have been developed to try to reduce this time. These can be classified into object-space subdivision and ray-space subdivision methods. The former constructs a scene space subdivision, such that each cell in

the subdivision references a relatively small set of objects, and ray traversal through this subdivision is relatively simple and fast. For any given ray the vast majority of objects are therefore never tested for intersection – only those that are picked up by the ray traversal scheme are considered as candidates for intersection. Examples of this method include bounding extent hierarchies [KK86] [GS87], direct space subdivision methods – such as uniform space subdivision [FTI86] [Ama84] [CW88], oc-trees [Gla84], and BSP trees [Kap85] [Jan86] [SS92]. It was argued in [SS92] that of these the BSP subdivision scheme results in the fastest ray traversal, with logarithmic time in the number of polygons.

Ray classification schemes on the other hand exploit coherence amongst rays. One example of this was the light buffer [HG86] which efficiently computed intersections for 'shadow feeler' rays. However, a general ray classification approach that applied to the entire ray tracing process was provided by [AK87]. Rays were represented as points in 5D space and a 32-tree of ray space was lazily built as each successive ray was encountered (in fact six 32-trees each representing one of the six faces of a bounding box around the scene). Each cell of a 32-tree represents a set of similar rays, and corresponding to each cell is a candidate set of objects. Every object in the candidate set is such that at least one of the rays in the cell intersects it. In other words a cell of the 32-tree corresponds to a beam in 3D space that intersects a set of objects – the candidate set for the cell. The size of the tree depends on the maximum permitted size of the candidate object set. Now given any new ray, it is filtered down the tree, its candidate set identified, and intersections carried out with these. The VLF-RT algorithm presented in this paper may be thought of as a much more efficient representation of this same idea – since in this case similar rays are also grouped together and each such group of rays has a candidate object set. However, the data structure is much simpler than the 32-tree, and ray-candidate set retrieval is look-up rather than a tree traversal.

Within each of these two broad categories there have been many proposals for further and substantial improvement in ray-object traversal speed. For example, building on the idea of a BSP representation Havran [HKBZ97] [HBZ98] introduced rope trees to further accelerate ray-BSP tree traversal. In addition caching schemes have been introduced to reuse elements of a solution across several views, exploiting a kind of ray-view coherence [WS99] [WDP99].

Advances in processor power and graphics hardware have supported a massive speed up in ray tracing so that today it is possible to attain interactive speed for millions of polygons on clusters of consumer PCs [WSB*03]. This research has relied on space subdivision schemes for fast ray-intersection solutions, in particular BSP trees, together with precise organisation of the overall algorithm to fit the needs of the hardware, and parallel implementation over PC clusters [WKB*02] [BWS03].

The evidence to date suggests that one scheme in particular; coherent ray tracing (CRT) [WSBW01] is the fastest implementation of ray tracing, by possibly several orders of magnitude. This uses a BSP tree space subdivision. The implementation is organized so that most memory accesses fall within the first two caches, which itself resulted in a speed-up by half an order of magnitude as reported in the original paper. Moreover, packets of 4 rays are SIMD traced in parallel. We have also implemented the single ray CRT scheme, and it is with the results of this that we compare our new approach in Section 6.

3. Virtual Light Field Ray Tracing

3.1 Data Structure

The virtual light field data structure was originally inspired by the light field [LH96] [GGSC96] and the type of representation used is similar to that in [CLF98] and also to a data structure used for visibility culling in [CC-OL98]. Whereas light fields typically only store radiance at the first intersection of a ray with an object, Layered Depth Images [SGHZ98] maintain radiance information about each of the surfaces that rays intersect rather than just the first surface, and in that sense the VLF is also similar to LDI. However, in VLF-RT we never store radiance, only object (in fact polygon) identifiers. We have previously used a general VLF data structure for a view independent global illumination solution [Ano04] where radiance information was stored. In this paper we specialize the data structure specifically for ray tracing, and therefore the solution is view-dependent, and requires an order of magnitude less memory. We now describe VLF-RT.

A scene can be enclosed, for example, by a regular cuboid. Suppose this is a cuboid bound by (-1,-1,-1) to (1,1,1). Consider the lower face (at z = -1) bounded by (-1,-1,-1) to (1,1,-1). This is subdivided into $n \times n$ square tiles. Each tile is the base of a beam parallel to the z-axis that extends infinitely (though only the finite part that intersects with the scene is of interest). The set of such $n \times n$ parallel beams is called the *canonical parallel subfield* (PSF). If l points with spherical coordinates $\omega_i = (\theta_i, \phi_i)$ are chosen on the unit hemisphere then l PSFs are defined as rotations of the canonical PSF by rotating the direction into the corresponding spherical point. The rotation can be achieved in any manner that is consistent throughout.

Consider any beam in the canonical PSF. This will intersect a number of surfaces in the scene. The corresponding tile stores this sequence of surface identifiers. The process of finding all the intersections of surfaces with the tiles of the canonical PSF is straightforward. If we consider the special case that all surfaces are planar polygons, then this is similar to polygon rasterisation, and can be implemented very efficiently. Given any other PSF, corresponding to direction ω_i the scene can be rotated such that ω_i is transformed to the (0,0,1) direction and then the rasterisation carried out in the canonical space.

It is critical to choose a parameterisation over the hemisphere so that *no searching* is required in order to find the closest PSF direction to any arbitrary direction – since such ray lookup is a critical operation during ray tracing. The method for placing points on the hemisphere uses a recursive subdivision of a regular tetrahedron, which partitions the hemisphere into triangles. However, fast

constant time lookup is attained for any arbitrary point on the hemisphere in order to find the closest stored point to any given point on the hemisphere. A method that achieved this was described in [Sla02].

The (finite) set of given PSF directions is denoted Ω_l and $\omega \in \Omega_l$ refers to a particular direction. The tiling coordinate system is referenced by (s, t) $s, t = 0,1,\ldots,n\text{-}1$. Hence a tile is referenced as (ω, s, t). The set of identifiers associated with a tile is denoted by $S(\omega, s, t)$.

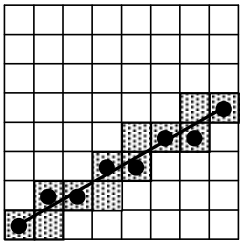

Figure 1: Finding the tiles corresponding to a polygon edge – the correct ones are the shaded tiles, DDA algorithm would produce only the marked tiles.

3.2 Constructing the Data Structure

The application of this data structure to ray tracing is very straightforward. First the data structure as discussed above is constructed. For each PSF ω_i the scene is transformed to the canonical space, and each polygon is orthographically projected to the base of the PSF, and the tiles that it covers computed. This can be achieved by traversing each polygon edge through the tiling to compute the tiles of all the polygon edges, and then filling in the non-edge tiles that are inside the polygon. It is important that the edge-tiling traversal algorithm allow for an 8-connected path, rather than follow a traditional DDA-style algorithm. The difference is shown in Figure 1. Such algorithms are discussed with reference to a 3D context in [CK90] but are easily adapted to 2D. As each tile (s, t) for a polygon is found the polygon identifier is written into $S(\omega_i, s, t)$. By the end of this process for all PSFs the data structure is complete.

3.3 Ray-Object Intersections

Now suppose that all identifiers for all tiles in all PSFs have been computed, and that we require the candidate set of objects for a ray with direction ω and origin (x, y, z). Find the direction in Ω_l that is closest to ω and suppose that this is ω_j. This can be done with a simple array look-up as shown in [Sla02]. There will be a rotation matrix $\mathbf{M}_j$ pre-calculated and stored with the PSF that rotates ω_j into (0,0,1). Then $(x, y, z)\mathbf{M}_j = (x_q, y_q, z_q)$ will be the point in the canonical PSF space that corresponds to (x, y, z) in scene space. In particular the projection $(x_q, y_q, -1)$ will belong to a particular tile. The set of identifiers in that tile forms a PVS set for the ray.

In fact the situation is slightly more complicated. Figure 2 shows a 2D analogue of a condition where a ray would project to more than one tile. In that case (Ray_j) if we only project the origin of the ray to the base of the PSF it would pick up tile 4. However, clearly the appropriate candidate set would be the union of those of tiles 4, 3 and 2. Therefore two points on each ray should be projected – the origin, and an end-point. In the case of shadow feeler rays the end-point is given by the light source position. In the case of primary or secondary rays the point on the ray that intersects with the boundary of the scene may be used. If the projections of these two points are not in the same tile then the appropriate set of candidate objects is the union of all tiles that the ray traverses. For a large enough number of rotations (l) both end-points of the ray will project to the same tile as the angle between a given direction and its nearest PSF tends towards zero.

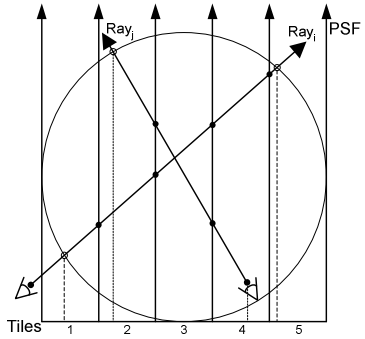

Figure 2: The rays overlap more than one tile.

4. Ray-Tile Traversal

Once a PSF and tile have been identified for a ray the set of polygons referenced must be searched to find the nearest ray-polygon intersection (if any). This is a critical operation. For a sufficient large number of polygons it turns out to be faster to use a BSP tree to traverse the entire set of polygons for every ray compared to a linear search of the polygons within the tiles. The reason is due to the logarithmic performance of the BSP tree and the linear performance in the average number of polygons per tile of the VLF-RT approach. On the other hand the VLF-RT approach does have the advantage that without any ray traversal, and simply with a lookup it is possible to reduce the search space to a very small fraction of its total size.

In order to reduce the linear dependence of the timing on the mean number of polygons per tile a BSP tree could equally be used to search the vastly reduced tile space instead of the entire scene space. In practice, since the direction of greatest variation amongst the polygons within a tile is by construction in the direction of the PSF to which the tile belongs, a BSP tree that subdivided the full 3D space would be wasteful. Instead, only subdivision planes that are perpendicular to the direction of the PSF are used. These subdivisions are mid-point subdivisions along the tile, and also only intersections between a polygon and a subdivision plane that occur within the tile boundary are significant.

5. Implementation Issues

The Visibility Field currently uses the mid-point for the splitting planes in the 1D BSP trees that exist along each tile. The polygon identifiers are pushed to the leaves using

a linearised layout for the BSP trees. This saves on memory, as no internal nodes are explicitly stored, however in some instances time is spent on traversing deeper than strictly necessary. The polygons are clipped to the tile boundaries before testing it against the BSP clipping planes to avoid unnecessary duplication. During BSP traversal the recursion is rolled out using an explicit stack, and the intersection kernel is inlined using a macro.

When intersecting a given ray the nearest PSF is used and the ray is projected onto this PSF using a line rasterisation algorithm similar to the one used for tile rasterisation during initialisation (see Figure 1). Also, the ray segment is limited by the intersections with the scene bounding sphere (see Figure 2, Ray_i), or if the viewpoint is inside the scene only the far intersection is used (see Figure 2, Ray_j). The intersected tile list (eg. tiles 1-5 for Ray_i in Figure 2) is traversed in front to back order and the ray segment is limited by the ray tile intersection points, such that early termination can be applied as soon as an intersection is found in a tile. Note that the directional discrepancy between the rays and the PSF has been exaggerated in Figure 2 to illustrate the point more clearly; in practice only a small number of tiles are intersected depending on the tile resolution and directional density.

The single ray CRT approach implemented follows [WSBW01]. It uses the spatial median for BSP splitting planes, and the optimized triangle layout described in [Wal04]. As in the VLF-RT case, the BSP traversal's recursion is rolled out using an explicit stack, and the intersection kernel is directly inlined using a macro. All other framework code such as ray creation and shading etc. is shared between the implementations.

6. Results

6.1 Parameters and Hardware

We compared performance of VLF-RT with the CRT implementation described above. For each method we used the system and parameters that were fastest. In the case of the BSP tree for CRT, a parameterisation must be chosen – specifically the maximum depth of the tree, and the ideal maximum number of polygons allowed per leaf-node (subject to the maximum depth). In order to determine these parameters we ran a series of pre-test experiments with the scenes described in the sections below, to determine the best combination of depth and leaf size, and these were used in the comparative performance tests.

Similarly, tile-BSP tree sizes needed to be determined for VLF-RT and the same strategy was used. Also we could vary the tile resolutions, already knowing that greater resolution would result in faster times. However, of interest is the graph of performance by number of triangles, and this is discussed in the next section. All timings were carried out on a dual 2.8Ghz Xeon workstation with 3GB memory, using only one processor.

6.2 Performance Curves

We are interested in ray-traversal speed as the number of polygons n increases. For this purpose we have created an artificial scene with uniformly distributed random triangles, see Figure 5. Figure 3 shows the frame time, averaged over about 15 frames for increasing number of triangles viewed from two viewpoints one outside the scene and one from inside.

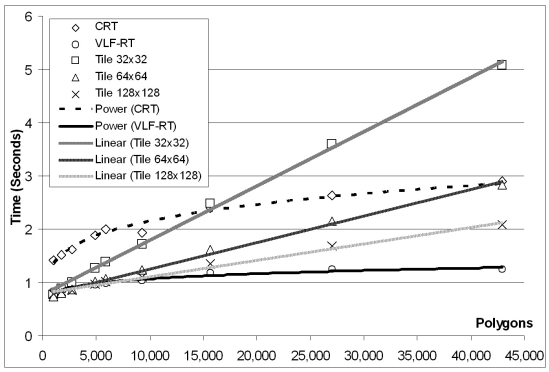

Figure 3: Average time to render a 512×512 image (un-shaded primary rays only) of scenes with increasing numbers of uniformly distributed random triangles.

The results show that the pure linear tiling approaches scale linearly in the average polygons per tile. So, even the tile 128×128 parameterisation will be slower than CRT for some n>45.000. However, it is obvious that the VLF-RT approach using BSP trees scales better than CRT, for this type of scene. This comes at a cost though; memory usages for these scenes range between 25MB to 1GB, and pre-processing times range from 3-20 minutes.

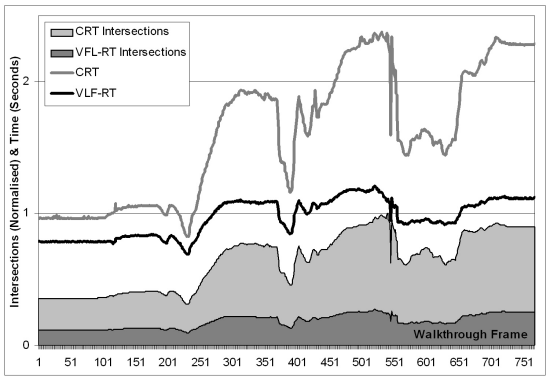

Figure 4: Frame times and normalised intersections for a walkthrough of the 16K scene.

6.3 Walkthrough

In the previous section we discussed performance for static viewpoints, now we consider a walkthrough of one of these scenes to see how performance varies across frames when the viewpoint changes during a walkthrough situation. The scene we will use is the 16K scene shown in Figure 5. Figure 4 shows the walkthrough timings and intersections comparing CRT with VLF-RT. The results show that both methods are sensitive to similar changes in viewed complexity and camera paths though the variations for CRT are much more pronounced. The VLF-RT method results in more stable render times across the frames – this

is due to the fact that though depth complexity is high across successive views of the considered scene, candidate search time is quite low.

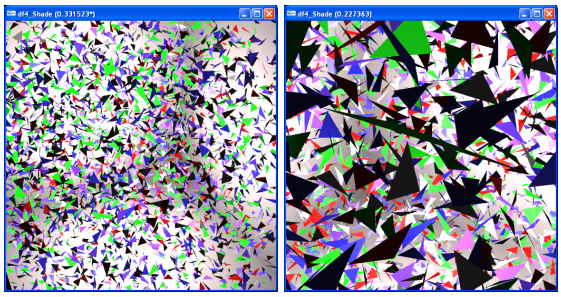

Figure 5: Images of the random scenes. The 9K triangle scene on the left is viewed from outside the scene while the 16K scene on the right is viewed from its interior.

7. Conclusions

In this paper we have introduced a new ray-traversal method, and illustrated its application in classical ray tracing. The method relies on a very fast lookup to obtain a candidate set of polygons for any ray, and then these polygons may be traversed by a BSP tree with partitioning along only one axis. The results suggest (Figure 3) that this method perform better than CRT, for the scene used.

In this paper we have only discussed walkthrough applications. However, real-time ray tracing also demands the possibility of dynamic changes to objects. This is easily achievable with the VLF-RT method. When an object is transformed it must first be deleted from the data structure, then its geometry transformed and inserted back into the data structure. Once these operations have been carried out the ray tracing can be used to render the next frame as usual. In order to delete an object from the VLF-RT data structure, all PSFs are visited, and the object rasterised into the tiling coordinate system as usual, except that in this case the identifiers of the object are removed rather than added. Then the polygon's geometry is transformed, and reinserted into both the tiling and BSP structures. This represents ongoing work and will be reported subsequently.

Another line of ongoing work is improving the BSP splitting plane heuristic moving away from the simple spatial-median approach. Also a SIMD implementation tracing 2x2 ray bundles is underway, and results from this will be reported on in the near future.

Acknowledgements

This research is funded by the UK EPSRC, grant number GR/R13685/01. Mel Slater is supported by an EPSRC Senior Research Fellowship. Thanks to Ingo Wald and Carsten Benthin for helpful suggestions on real time ray tracing.

References

[App68] APPEL., A.: Some techniques for shading machine renderings of solids. *SJCC*, 27–45, 1968.

[Ama84] AMANATIDES, J.: Ray Tracing with Cones, *Computer Graphics* (SIGGRAPH), 1884, 18, 129-135.

[AK87] ARVO, J. AND KIRK, D.: Fast Ray Tracing by Ray Classification, *Computer Graphics* (SIGGRAPH), 1987, 21(4), 55-64.

[BWS03] BENTHIN, C., WALD, I., SLUSALLEK, P.: A Scalable Approach to Interactive Global Illumination, *Eurographics 2003* 22(3) P. Brunet and D. Fellner (eds).

[CLF98] CAMAHORT, E., LERIOS, A., FUSSELL, D.: Uniformly Sampled Light Fields, *Rendering Techniques 1998*: 117-130.

[CHH02] CARR, N.A., HALL, J.D., HART, J.C.: The Ray Engine, *Proc. Graphics Hardware 2002*, September 2002.

[CC-OL98] CHRYSANTHOU, Y., COHEN-OR, D., LISCHINSKI, D.: Fast Approximate Quantitative Visibility for Complex Scenes, *Computer Graphics International '98*, Hannover, Germany, June 1998, 220-227.

[CW88] CLEARY, J.G., WYVILL, G. Analysis of an Algorithm for Fast Ray Tracing Using Uniform Space Subdivision, *The Visual Computer*, 1988, 4, 65-83.

[CK90] COHEN, D., KAUFMAN, A: Scan Conversion Algorithms for Linear and Quadratic Objects, in A. Kaufman (ed), *Volume Visualization*, 1990, pp 280-300, IEEE Computer Science Press.

[CPC84] COOK, R.L., PORTER, T., CARPENTER, L.: Distributed ray tracing. *Computer Graphics*, 18(3):137–145, 1984.

[FTI86] FUJIMOTO, A., TANAKA, T., IWATA, K. (1986) ARTS: Accelerated Ray-Tracing System, *IEEE CG&A* 6(4), 16-26.

[GS87] GOLDSMITH, J. and SALMON, J. (1987) Automatic Creation of Object Hierarchy for Ray Tracing, *IEEE CG&A* 7(5), 14-20.

[Gla84] GLASSNER, A.S.: Space Subdivision for Fast Ray Tracing, *IEEE CG&A*, 1984, 4(10), 15-22.

[Gla89] GLASSNER, A.: *An Introduction to Raytracing*. Academic Press, 1989.

[GGSC96] GORTLER, S., GRZESZCZUK, R., SZELISKI, R., COHEN, M.: The Lumigraph, *Computer Graphics* (SIGGRAPH), Annual Conference Series, 1996, 43-52.

[HG86] HAINES, E.A., GREENBERG, D.P.: The Light Buffer: A Shadow Testing Accelerator, *IEEE CG&A*, 1986, 6(9), 6-16.

[HKBZ97] HAVRAN, V., KOPAL, T., BITTNER, J., ZARA, J.: Fast Robust BSP Traversal Algorithm for Ray Tracing, *Journal of Graphics Tools*, 2(4):15-24,1997.

[HBZ98] HAVRAN, V., BITTNER, J., ZARA, J.: Ray Tracing with Rope Trees, *Proceedings of 13th Spring Conference on Computer Graphics*, 130-139, Budmerice, 1998.

[Jan86] JANSEN, F.: Data Structures for Ray Tracing, in L. Kessener, F. Peters and M. van Lierop (eds) Data Structures for Raster Graphics, *Eurographics Seminar*, 1986, NY Springer-Verlag, 57-73.

[Kap85] KAPLAN, M.R.: Space-tracing, a constant time ray tracer, *Computer Graphics* (SIGGRAPH 85), 1985, State of the Art in Image Synthesis notes.

[KK86] KAY, T.L., KAJIYA, J.T.: Ray tracing complex scenes. *Computer Graphics* (SIGGRAPH) 20(4):269–278, August 1986.

[LMW90] LAMPARTER, B., MULLER, H., WINCKLER, J.: The Ray-z-Buffer—An Approach for Ray Tracing Arbitrarily Large Scenes, *Technical Report*, Universitat Freiburg Institut fur Informatik, Apr. 1990.

[LH96] LEVOY M, HANRAHAN, P.: Light Field Rendering, *Computer Graphics* (SIGGRAPH), Annual Conference Series, 1996, 31-42.

[PBMH02] PURCELL, T.J., BUCK, I., MARK, W.R., HANRAHAN, P.: Ray Tracing on Programmable Graphics Hardware, *ACM Transactions on Graphics*, 21(3), July 2002, 703-712.

[PKG97] PHARR, M., KOLB, C., GERSHBEIN, R.: Rendering Complex Scenes with Memory-Coherent Ray Tracing, *Computer Graphics*, 1997, 31, Annual Conference Series.

[SGHZ98] SHADE, J., GORTLER, S.J., HE, L. AND SZELISKI, R: Layered Depth Images, *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH), 1998.

[SS92] SUNG, K., SHIRLEY, P.: Ray tracing with the BSP tree, *Graphics, Gems III*, pages 271—274, 1992.

[Sla02] SLATER, M. (2002) Constant Time Queries on Uniformly Distributed Points on a Hemisphere, *Journal of Graphics Tools*, 7(1):33-44.

[Ano04] ANONYMOUS: A Virtual Light Field Approach to Global Illumination, *Computer Graphics International*, Crete, Greece, June 16-19, 2004, in press.

[WSBW01] WALD, I., SLUSALLEK, P., BENTHIN, C., WAGNER, M.: Interactive Rendering with Coherent Ray Tracing, *Eurographics* 2001 Proceedings, *Computer Graphics Forum*, 20(3), A. Chalmers and T.-M. Rhyne (eds.), 2001, 153—164.

[WKB*02] WALD, I., KOLLIG, T., BENTHIN, C., KELLER, A., SLUSALLEK, P.: Interactive Global Illumination Using Fast Ray Tracing, *Thirteenth Eurographics Workshop on Rendering*, 2002, P. Debevec and S. Gibson (eds).

[WSB*03] WALD, I., SCHMITTLER, J., BENTHIN, C., SLUSALLEK, P., PURCELL, T.J.: Realtime Ray Tracing and its use for Interactive Global Illumination, STAR, *Eurographics 2003* 22(3) P. Brunet and D. Fellner (eds.).

[WDP99] WALTER, B., DRETTAKIS, G., PARKER, S.: Interactive rendering using render cache, *Rendering Techniques '99*, Eurographics, (D. Lischinski and G.W. Larson, eds.), 19–30, 1999.

[WS99] WARD, G. AND SIMMONS, M.: The Holodeck Ray Cache: An Interactive Rendering System for Global Illumination in Nondiffuse Environments, *ACM Transactions on Graphics*, 1999, 18(4):361-98.

[Whi80] WHITTED, T.: An Improved Illumination Model for Shaded Display, *Communications of the ACM*, 1980, 23(6), 343-349.

[Wal04] WALD, I.: Realtime Ray Tracing and Interactive Global Illumination, PhD thesis, Computer Graphics Group, Saarland University, 2004.