The Visual Computer, 2006 (Proceedings of Pacific Graphics)

Johannes Günther · Heiko Friedrich · Hans-Peter Seidel · Philipp Slusallek

Interactive Ray Tracing of Skinned Animations

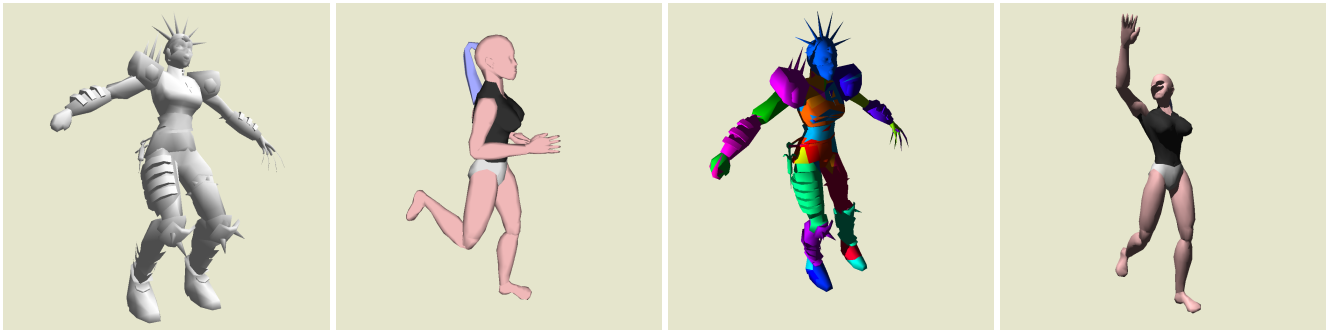


Fig. 1 The "UT 2003" and "Cally" example scenes demonstrating interactively skinned meshes. These dynamic scenes are ray traced between 4 (including shadows, right two images) and 15 frames per second (with a simple shader, left two images) at 1024×1024 pixels on a single CPU.

Abstract Recent high-performance ray tracing implementations have already achieved interactive performance on a single PC even for highly complex scenes. However, so far these approaches have been limited to mostly static scenes due to the high cost of updating the necessary spatial index structures after modifying scene geometry. In this paper, we present an approach that avoids these updates almost completely for the case of skinned models as typically used in computer games. We assume that the characters are built from meshes with an underlying skeleton structure, where the set of joint angles defines the character's pose and determines the skinning parameters. Based on a sampling of the possible pose space we build a static fuzzy kd-tree for each skeleton segment in a fast preprocessing step. These fuzzy kd-trees are then organized into a top-level kd-tree. Together with the skeleton's affine transformations this multi-level kd-tree allows fast and efficient scene traversal at runtime, while arbitrary combinations of animation sequences can be applied interactively to the joint angles. We achieve a real-time ray tracing performance of up to 15 frames per second at 1024×1024 resolution even on a single processor core.

Keywords Ray tracing · Fuzzy kd-tree · Dynamic scenes

J. Günther · H.-P. Seidel
MPI Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
E-mail: {guenther, hpseidel}@mpi-inf.mpg.de

H. Friedrich · P. Slusallek
Saarland University
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
E-mail: {friedrich, slusallek}@graphics.cs.uni-sb.de

1 Introduction

Interactive image synthesis and real-time visualization is today a matter-of-course. Rasterization hardware is continuously using new tricks to support more advanced shading and lighting effects. Moreover, ray tracing – famous for its superior rendering quality – has finally become fast enough for real-time applications. Even highly complex scenes with millions of primitives can be ray traced at real-time frame rates [RSH05].

A feature that is still not supported to its full extent in real-time ray tracing is *dynamic scenes*. Unfortunately, many applications require at least the interactive rendering of dynamically changing objects, and computer games even demand at least 60 Hz. Real-time rendering of animations is still the domain of graphics hardware because of its immense hardware acceleration, e.g. for vertex shaders. With more than 48 pipelines [ATI] at present, GPUs provide massive parallel computation power. Furthermore, rasterization does not depend on precomputed acceleration structures for rendering, because every polygon is rendered in every frame anyway.

However, ray tracing, first introduced into the field of computer graphics by Appel et al. [App68], must to exploit spatial index structures to speed up the rendering process by minimizing the number of ray-primitive intersections. All current spatial index structures used in ray tracing, such as grids [CWBV83, AW87], octrees [Gla84], bounding volume hierarchies [RW80] (BVHs), and kd-trees [Jan86], must be rebuilt when objects change. Depending on the scene and the type of index structure, the reconstruction can take up to several minutes, which of course is not tolerable for in-

teractive applications [Hav01]. Especially highly optimized kd-trees [WH06] built with *surface area heuristics* [MB89] tend to have extremely long construction times.

In recent years, kd-trees have been shown to be an efficient acceleration structure for static scenes, which can yield excellent ray tracing performance [HPP00]. One main advantage of kd-trees is that they subdivide space into irregularly sized areas [PH04, Hav01] and thus they also adapt to unevenly distributed scene-primitives very well (compared to grids). Additionally, the simple traversal algorithm of kd-trees can be mapped efficiently to modern processors (compared to octrees and BVH) [Wal04]. As a consequence, most modern real-time ray tracing systems implement kd-trees, including CPU-based systems [WSBW01, RSH05], ray tracing hardware [SWS02, WSS05], and GPU-based ray tracers [FS05].

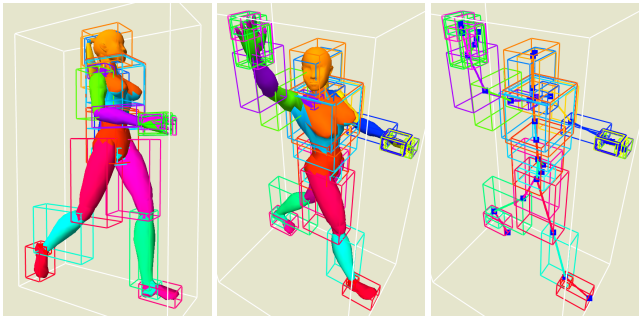


Fig. 2 To ray trace a frame of an animation we first build a small kd-tree over the current bounding boxes of each bone object. These boxes are shown for three configurations of the "Cally" scene. Rays hitting a box get inversely transformed into the local coordinate system of this box and traverse its fuzzy kd-tree.

2 Related work

Real-time ray tracing has a short history and not much research has been done in the area of interactive ray tracing dynamic scenes yet.

In 2001 first practical results towards real-time ray tracing of static scenes on commodity PCs were presented by Wald et al. [WSBW01]. An extensive study of acceleration structures by Havran [Hav01] showed that kd-trees perform best for a large number of test scenes.

Lext et al. [LAM01] proposed a novel algorithm to efficiently ray trace animated objects that undergo simple affine transformations such as rotation and translation. To do so, they used a hierarchy of oriented bounding boxes over the objects, with each object having its own grid as the acceleration structure. Two years later, Wald et al. [WBS03] introduced the concept of a two-level kd-tree to ray trace the same class of dynamic objects as Lext but with kd-trees.

Reihard et al. [RSH00] proposed the usage of *interactive grids* to render scenes with random motion like particle- and turbulence systems. In order to achieve interactive results they used an SGI Onyx 2000 with 32 processors.

Recently, Günther et al. [GFW+06] proposed a motion decomposition approach to ray trace animations. A motion clustering algorithm identifies regions with *coherent motion* that can be expressed as an affine transformation of the vertices of the scene. After subtraction of this common transformation for each cluster, the residual motion is handled by a so-called fuzzy kd-tree. A fuzzy kd-tree is not directly built over the scene primitives but over axis-aligned fuzzy boxes, which bound the space where an primitive can be located during the animation. Although real-time rendering frame rates can be achieved with this approach, the algorithm has the disadvantage that the complete animation sequence has to be known in advance. Furthermore, the motion clustering is a costly preprocessing step. In this paper, we extend the idea of motion decomposition to skinned animations.

Very recently it was demonstrated that ray tracing of dynamic scenes at interactive frame rates can also be accomplished with grids [RSH00, WIK+06], bounding volume hierarchies (BVHs) [WBS06, LYTM06, CHCH06], or hybrid data structures [WMS06, WK06]. These acceleration structures can be rebuilt or updated faster than kd-trees when geometry changes. However, they are still inferior in ray tracing performance in the general case.

3 Ray tracing of skinned animations

Although kd-trees are widely considered as the most efficient acceleration structure for ray tracing, their large construction time heavily complicates its utilization for dynamic scenes. Thus our goal is to avoid rebuilding the kd-tree while still retaining its efficiency as an acceleration structure.

The idea is to compensate for most of the motion of the scene by transforming rays instead of the geometry. For this purpose, we adapt the concept of two-level kd-trees of [WBS03]. The residual motion of the scene that cannot be compensated is handled separately by fuzzy kd-trees similar to the approach in [GFW+06].

We extend the approaches of previous work by supporting a more flexible type of animation, specifically skinned meshes. By exploiting the underlying bone information, we can efficiently construct the fuzzy kd-trees once in a preprocessing phase. During ray tracing of the skinned animation these fuzzy kd-trees do not need to be updated.

Before describing our work in detail, we will briefly review the concept of two-level kd-trees.

3.1 Two-level kd-trees

If an animated scene consists of several moving rigid objects, separate kd-trees can be built, one for each object. This has the advantage that these kd-trees can be pre-built and remain valid during the course of the animation.

The absolute motion of all objects is accounted for by a top-level kd-tree. Only this kd-tree needs to be updated every frame as it is built over the current bounding boxes of all

objects. This way the top-level kd-tree adapts to the changing positions and orientations of moving objects (Figure 2). However, this rebuilding can be done very quickly because the number of moving objects is usually rather small (less than 100), allowing interactive frame rates.

To render a new frame, ray traversal starts in the top-level kd-tree. For every intersected bounding box the ray is transformed into the local coordinate system of that box and traversal continues at the root node of the corresponding second-level kd-tree.

With such a two-level kd-tree hierarchy it is possible to interactively ray trace animated objects that just undergo simple affine transformations. A distinct example for this type of animation is the BART robots scene [LAM00].

3.2 Vertex skinning

Nevertheless, for more flexible animation types the two-level kd-tree scheme alone is not sufficient. Our goal is to ray trace skinned animations because they provide many advantages. Skinned meshes are a powerful tool for content creation in computer games and animated movies. Artists create meshes, e.g. an avatar for a game such as the "UT 2003" model in Figure 1 and attach a bone model as well as physics to it. Using this additional information they can then apply deformations to the mesh. This type of motion is a good approximation of many real scenarios and is also common for animated synthetic datasets.

A skinned mesh is animated with the help of an underlying skeleton, usually referred to as skeleton subspace deformation [MTLT88, MTT91]. Each vertex of the mesh is influenced by one or more bones. For example, vertices forming an arm move together with the corresponding arm bone. Additionally, vertices near the joints of two bones are influenced by both bones, resulting in a smooth skin surface without cracks or other artifacts near the joints.

Mathematically, bones are described by transformations and standard skinning is a simple weighted interpolation:

$$v' = \sum_{i=1}^{n} w_i X_i v, \quad \text{with} \sum_{i=1}^{n} w_i = 1$$

The value n is the number of bones influencing the skinned position v'. X_i is the current transformation of bone i and w_i is the associated scalar weight (or influence) of that bone for the vertex v in the rest pose of the mesh. The rest pose is the original mesh configuration modeled by an artist without applied skinning.

If all vertices were to be influenced by only one bone ($n = 1$) we would have the situation of simple affine-only transformations where the two-level kd-tree scheme can be applied.

However, it is an important observation that even when $n > 1$ the interpolated motion of a vertex in the *local coordinate system* of a carefully chosen bone is typically rather small. The reason is that usually vertices are significantly influenced by neighboring bones only.

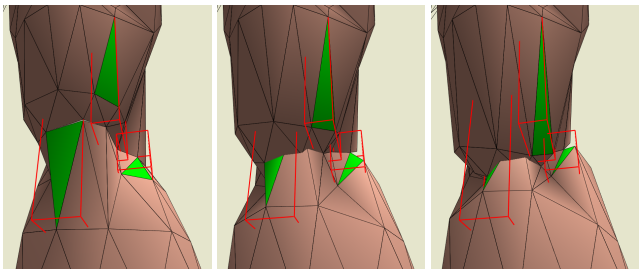


Fig. 3 The residual motion of each triangle (green) is bounded by a fuzzy box (red). As the fuzzy kd-tree is built over these fuzzy boxes instead of the triangles it is valid for all poses of the skinned model.

This small residual motion of the vertices and triangles is then bounded by so-called *fuzzy boxes*. A fuzzy box is an axis-aligned bounding box in the local coordinate system that contains the valid positions of a triangle. For each bone, a fuzzy kd-tree [GFW+06] is then built over the fuzzy boxes rather than the triangles. This has to be done only once in a preprocessing step, because the local movement of the triangles does not invalidate the fuzzy kd-tree (see also Figure 3).

To render a new frame the pose of the skeleton is updated as well as the vertex positions. Then the world coordinate bounding boxes of each bone and associated triangles are calculated. Finally, we can apply the two-level kd-tree scheme and rebuild the small top-level kd-tree over the bone fuzzy kd-trees.

3.3 Stochastic fuzzy box computation

For the correctness of this algorithm it is crucial that the fuzzy boxes used to build the fuzzy kd-trees are conservative. Otherwise it might happen that triangles get culled during traversal of the fuzzy kd-trees and that rays erroneously miss these triangles, which will result in holes in the mesh.

Additionally, we also want to optimize ray tracing performance. Thus it is preferable to have small fuzzy boxes, because the surface area of the fuzzy boxes is proportional to the probability that a random ray will hit this box [MB89].

To estimate a set of small, but sufficiently conservative fuzzy boxes, we sample the pose space during the preprocessing phase. The full pose space can be sampled by rotating each bone arbitrarily. However, tighter fuzzy boxes and thus better ray tracing performance can be achieved by restricting the bone rotation relative to its parent bone. Additional information from the rendering application – such as joint limits – can be used to restrict the pose space, because arbitrary bone rotations are quite unnatural for most models. Another possibility is to only sample the different animation sequences – such as walking, running, or kicking – used e.g. by a game application.

Note that the size of the fuzzy boxes depends not only on the pose space but also on the choice of the bone a triangle is assigned to. In order to find the optimal bone, we do an exhaustive search by calculating the fuzzy box in every

bone's local coordinate system. The fuzzy box of a triangle is the union of the fuzzy boxes of its vertices.

The fuzzy boxes of all vertices are calculated by sampling as follows. For each sample (i.e. a pose of the skeleton), every vertex is skinned and subsequently transformed into the local coordinate space of each bone. The fuzzy box of a vertex in a bone's local coordinate system is then the axis-aligned bounding box of its samples, i.e. its transformed positions.

pose space	#samples	fuzzy area	avg. #tris per leaf	fps
arbitrary bone rotations	1000	747k	4.1	7.7
applying joint limits	245	580k	3.4	10.2
many animation sequences	124	552k	3.2	11.6
just one animation sequence	62	515k	3.1	12.0

Table 1 Influence of the valid pose space on the fuzzy box surface area, the quality of the fuzzy kd-trees and subsequently the rendering performance for the "UT 2003" scene. Restricting the pose space – at least by joint limits – greatly improves ray tracing performance.

4 Implementation and results

We implemented our ray tracing engine with C++ and use SSE [Int02] for optimizing inner loops. The kd-tree traversal is loosely based on *multi-level ray tracing* and the *inverse frustum culling* approach of Reshetov et al. [RSH05] with a ray packet size of 4×4. The ray-triangle intersection test is basically the same as described in [Wal04] for packets of rays. To manage the skeleton and skinning parameters of the models, we use the open source library Cal3D [Cal]. With Cal3D we can also smoothly blend different animation sequences as well as interpolate between key frames (see Figure 4). All measurements are done on a workstation equipped with two AMD Opteron 2.4 GHz processors and a GeForce 6800 GT PCIe graphics board.

4.1 GPU accelerated display

In real-time ray tracing implementations it was common to store the color of each computed pixel of the current frame in the main memory of the PC. After all pixel calculations were done, an OpenGL or DirectX call was made to transfer the image to the graphics hardware for display. With a float frame buffer of full-screen resolution, this transfer can easily be a bottleneck – measurements have shown that it can slow down the overall rendering process by up to 25%.

To ease this problem, we use the OpenGL *pixel buffer objects* extension to directly store computed pixel values in the texture memory of the graphics board. Besides reducing the display overhead to only 4 ms, this approach has the additional advantage that color channel swizzling or tone mapping (see e.g. [GWWH03]) can be efficiently performed by the pixel shader pipelines of the GPU. Merely moving the swizzling and float-to-int conversion of the color channels from the CPU to the GPU significantly increases the frame rate.

4.2 Fuzzy box estimation

To evaluate the trade-off between the flexibility of posing a skinned mesh and ray tracing performance, we restricted the valid pose space in several steps. Arbitrary rotations of the bones on the one hand allow the greatest freedom for a user to interact with the model. On the other hand, it is often meaningful to apply joint limits to the skeleton to avoid unnatural mesh deformations. The pose space can be further restricted by allowing only certain types of (predefined) movements (such as walking or waving), which is often the case in game applications.

Table 1 shows our measurements for all these pose space configurations. With increasing freedom of the bone rotations, the fuzzy boxes become larger. This leads to more overlap of the fuzzy boxes, which decreases the efficiency of the fuzzy kd-tree. More intersections need to be calculated, resulting in decreased ray tracing performance.

The accumulated surface area of the fuzzy boxes of all triangles is a good measure for the expected ray tracing performance because it is proportional to the number of intersections of a random ray with these boxes [MB89]. Note that the absolute value of the surface area is meaningless, because it directly depends on the scale of the scene. One measure for the efficiency of kd-trees is the average number of triangles per kd-tree leaf. Higher numbers account for larger overlap of fuzzy boxes, because it is not meaningful to split the overlapping area during kd-tree creation – every child node would contain the same triangles – and we cannot save intersection tests. Finally, we verified our expectations by measuring the achieved frames per second for ray tracing the rest pose with the same viewpoint for all the different pose space configurations.

All these measures show that restricting the pose space improves the ray tracing performance. The biggest performance gain can be obtained by applying joint limits.

4.3 Performance depending on mesh size

To evaluate the preprocessing as well as the rendering performance of our ray tracing implementation we measured the pose space sampling time, the time to construct the fuzzy kd-trees, and the achieved frames per second for two scenes. We were not able to acquire models with a high number of polygons that include the necessary skeleton and skinning parameters. To still be able to estimate the performance of our system for scenes with a high number of triangles, we decided to iteratively subdivide our low-resolution models.

Table 2 summarizes our results. All numbers were measured at a screen resolution of 1024×1024 pixels on a single CPU with a simple diffuse shader. Even with hundreds of thousands of triangles we achieved reasonable fast preprocessing within seconds to a few minutes. As expected,

scene	#tris	preprocessing time in seconds sampling	building kd-trees	fps
Cally	3k	0.63	0.65	9.5
Cally	10k	1.80	2.28	7.1
Cally	30k	5.21	8.77	5.7
Cally	90k	16.4	34.5	3.6
Cally	271k	50.4	150	2.0
UT 2003	2k	0.59	0.39	15.5
UT 2003	7k	1.23	1.67	12.3
UT 2003	22k	3.08	6.41	7.4
UT 2003	67k	8.63	23.8	4.0
UT 2003	200k	29.7	88.9	1.9

Table 2 Measured performance data of our example scenes for different subdivision levels. Even with hundreds of thousands of triangles we achieved reasonable fast preprocessing within a few minutes and still ray trace at interactive frame rates.

the sampling time was roughly linear in the number of triangles. For ray tracing dynamic scenes, we achieved a rendering performance of up to 15 frames per second. Furthermore, interactive frame rates were still possible for large scenes.

Günther et al. [GFW+06] compared their fuzzy kd-tree for a predefined animation with highly optimized kd-trees which were pre-built for every time-step. Our results are congruent with their reported numbers: The more flexible fuzzy kd-trees degrade the rendering performance approximately by a reasonable factor of two.

5 Discussion and conclusions

We have presented a novel algorithmic approach to ray trace skinned meshes efficiently. We use a small top-level kd-tree and bone transformations to account for the dominating deformations of the mesh. As the residual motion of triangles can be handled by fuzzy kd-trees, we avoid almost completely the costly reconstruction of acceleration structures. As a result, we are able to achieve interactive frame rates even for high polygonal models.

Supporting skinned meshes rather than only predefined animations as in [GFW+06] has several advantages:

- The preprocessing time is greatly reduced. For predefined animations an expensive clustering step is necessary to find coherent moving triangles.
- We save a lot of memory because we just need one base mesh for the entire animation, plus the corresponding bones and skinning information – in contrast to many meshes, one for every time-step.
- The most important advantage is the gained flexibility: With skinned meshes it is no problem to smoothly interpolate between key frames or to blend several animation sequences. Even direct user interaction is possible if we sample the (optionally limited) pose space.

6 Future work

One direction for future work is the investigation of ray tracing even more general types of animations, such as blending meshes or meshes deformed by physical simulations. For example, ray tracing the cloth worn by a game character is not yet possible with our method.

Under-sampling of the pose space can lead to underestimation of the fuzzy boxes and consequently to rendering artifacts. Thus instead of sampling the pose space it might be desirable to find the extent of the fuzzy boxes analytically or by using affine arithmetic. Nevertheless, in practice we noticed no such artifacts when using a moderate number of samples (i.e. one sample per key frame, see also Table 1).

With the advancements in ray traversal and intersection performance, the actual *shading* of the hit-points already became the bottleneck. Thus it is not yet possible to run a fully featured ray tracing system at interactive frame rates on a single CPU. Advanced ray tracing effects, such as glossy reflections, soft shadows, or global illumination still require the combined power of many processors – even with static scenes. Thus it is desirable to further research the acceleration of distribution ray tracing and shading operations.

It has been shown by Yoon and Manocha [YM06] that a cache-efficient layout of BVHs significantly increases ray tracing performance. Certainly the same idea should be applicable to (fuzzy) kd-trees.

It would also be interesting to compare our approach of using approved kd-trees for ray tracing dynamic scenes to other acceleration structures, such as grids or BVHs that can be quickly updated.

References

App68. Arthur Appel. Some techniques for shading machine renderings of solids. In *AFIPS Conference Proceedings*, volume 32, pages 37–45, Washington, DC, 1968. Thompson Book Company. Proceedings of the Spring Joint Computing Conference (SJCC).

ATI. ATI. The ATI Homepage. http://www.ati.com/.

AW87. John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In *Proceedings of Eurographics*, pages 3–10. Elsevier Science Publishers, Amsterdam, North-Holland, 1987.

Cal. Cal3D. 3D Character Animation Library. https://gna.org/projects/cal3d/.

CHCH06. Nathan A. Carr, Jared Hoberock, Keenan Crane, and John C. Hart. Fast GPU ray tracing of dynamic meshes using geometry images. In *Proceedings of Graphics Interface*. A.K. Peters, 2006.

CWBV83. John Cleary, Brian Wyvill, Graham Birtwistle, and Reddy Vatti. A parallel ray tracing computer. In *Proceedings XI Association of Simula Users Conference*, pages 77–80, 1983.

FS05. Tim Foley and Jeremy Sugerman. KD-tree acceleration structures for a GPU raytracer. In *HWWS '05 Proceedings*, pages 15–22, New York, NY, USA, 2005. ACM Press.

GFW+06. Johannes Günther, Heiko Friedrich, Ingo Wald, Hans-Peter Seidel, and Philipp Slusallek. Ray tracing animated scenes using motion decomposition. *Computer Graphics Forum*, 25(3), September 2006. (Proceedings of Eurographics).

Gla84. Andrew S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

Fig. 4 "Cally" featuring self shadowing in different poses.

GWWH03. Nolan Goodnight, Rui Wang, Cliff Woolley, and Greg Humphreys. Interactive time-dependent tone mapping using programmable graphics hardware. In Per H. Christensen and Daniel Cohen-Or, editors, *Proceedings of the 2003 Eurographics Symposium on Rendering*, pages 26–37, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

Hav01. Vlastimil Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Faculty of Electrical Engineering, Czech Technical University in Prague, 2001.

HPP00. Vlastimil Havran, Jan Prikryl, and Werner Purgathofer. Statistical comparison of ray-shooting efficiency schemes. Technical Report TR-186-2-00-14, Department of Computer Science, Czech Technical University; Vienna University of Technology, July 2000.

Int02. Intel Corp. Intel Pentium III Streaming SIMD Extensions. http://developer.intel.com/vtune/cbts/simd.htm, 2002.

Jan86. Frederik W. Jansen. Data structures for ray tracing. In *Proceedings of the workshop on Data structures for Raster Graphics*, pages 57–73, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

LAM00. Jonas Lext, Ulf Assarsson, and Tomas Möller. BART: A benchmark for animated ray tracing. Technical report, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, May 2000. Available at http://www.ce.chalmers.se/BART/.

LAM01. Jonas Lext and Tomas Akenine-Möller. Towards rapid reconstruction for animated ray tracing. In *Eurographics 2001 – Short Presentations*, pages 311–318, 2001.

LYTM06. Christian Lauterbach, Sung-Eui Yoon, David Tuft, and Dinesh Manocha. RT-DEFORM interactive ray tracing of dynamic scenes using BVHs. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, September 2006.

MB89. J. David MacDonald and Kellogg S. Booth. Heuristics for ray tracing using space subdivision. In *Graphics Interface Proceedings 1989*, pages 152–163, Wellesley, MA, USA, June 1989. A.K. Peters, Ltd.

MTLT88. Nadia Magnenat-Thalmann, Richard Laperrière, and Daniel Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics interface '88*, pages 26–33, Toronto, Ont., Canada, Canada, 1988. Canadian Information Processing Society.

MTT91. Nadia Magnenat-Thalmann and Daniel Thalmann. *Human body deformations using joint-dependent local operators and finite-element theory*, pages 243–262. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1991.

PH04. Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufman, 2004.

RSH00. Erik Reinhard, Brian Smits, and Chuck Hansen. Dynamic acceleration structures for interactive ray tracing. In *Proceedings of the Eurographics Workshop on Rendering*, pages 299–306, Brno, Czech Republic, June 2000.

RSH05. Alexander Reshetov, Alexei Soupikov, and Jim Hurley. Multi-level ray tracing algorithm. *ACM Transaction of Graphics*, 24(3):1176–1185, 2005. (Proceedings of ACM SIGGRAPH).

RW80. Steve M. Rubin and Turner Whitted. A three-dimensional representation for fast rendering of complex scenes. *Computer Graphics*, 14(3):110–116, July 1980.

SWS02. Jörg Schmittler, Ingo Wald, and Philipp Slusallek. SaarCOR – A Hardware Architecture for Ray Tracing. In *Proceedings of the ACM SIGGRAPH/Eurographics Conference on Graphics Hardware*, pages 27–36, 2002.

Wal04. Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Saarland University, 2004.

WBS03. Ingo Wald, Carsten Benthin, and Philipp Slusallek. Distributed interactive ray tracing of dynamic scenes. In *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG)*, pages 77–86, 2003.

WBS06. Ingo Wald, Solomon Boulos, and Peter Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. SCI Institute Technical Report UUSCI-2006-015, University of Utah, 2006. (conditionally accepted at ACM Transactions on Graphics, available at http://www.sci.utah.edu/~wald/Publications/webgen/2006/BVH/download/togbvh.pdf).

WH06. Ingo Wald and Vlastimil Havran. On building fast kd-trees for ray tracing, and on doing that in O(N log N). In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, September 2006.

WIK+06. Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, and Steven G. Parker. Ray tracing animated scenes using coherent grid traversal. *ACM Transactions on Graphics*, 25(3):485–493, 2006. (Proceedings of ACM SIGGRAPH).

WK06. Carsten Wächter and Alexander Keller. Instant ray tracing: The bounding interval hierarchy. In *Rendering Techniques 2006, Proceedings of the Eurographics Symposium on Rendering*, pages 139–149, Aire-la-Ville, Switzerland, June 2006. The Eurographics Association.

WMS06. Sven Woop, Gerd Marmitt, and Philipp Slusallek. B-kd trees for hardware accelerated ray tracing of dynamic scenes. In *Proceedings of Graphics Hardware*, 2006.

WSBW01. Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive Rendering with Coherent Ray Tracing. *Computer Graphics Forum*, 20(3):153–164, 2001. (Proceedings of Eurographics).

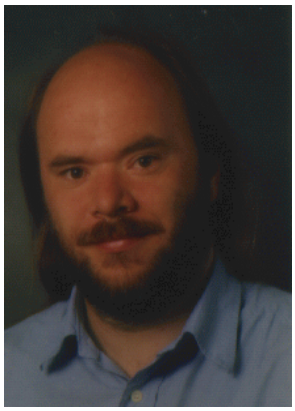
WSS05. Sven Woop, Jörg Schmittler, and Philipp Slusallek. RPU: A Programmable Ray Processing Unit for Realtime Ray Tracing. *Proceedings of ACM SIGGRAPH*, 24(3):434–444, 2005.

YM06. Sung-Eui Yoon and Dinesh Manocha. Cache-efficient layouts of bounding volume hierarchies. *Computer Graphics Forum*, 25(3), September 2006. (Proceedings of Eurographics).

Johannes Günther is currently a PhD student at the Computer Graphics Group at the Max-Planck-Institute for Computer Science in Saarbrücken, Germany. In 2004, he finished his diploma thesis on real-time photon mapping for caustic illumination at the Saarland University. Before 2003 he studied physics and computer science at the Chemnitz University of Technology and worked as an intern at the Technion – Israel Institute of Technology, Haifa. His main research interests are real-time global illumination and ray tracing.

Heiko Friedrich is currently a PhD student at the Computer Graphics Group at Saarland University in Saarbrücken, Germany. In 2004, he finished his Master's thesis on real-time iso-surface ray tracing. Before doing so, he received a Bachelor's degree from Saarland State University in 2002, and worked as an intern at the Imaging and Visualization Department of Siemens Corporate Research in Princeton/USA. His main research interests are real-time volume rendering and ray tracing.

Hans-Peter Seidel studied mathematics, physics and computer science at the University of Tübingen, Germany. He received his PhD in mathematics in 1987, and his habilitation degree in computer science in 1989, both from the University of Tübingen. From 1989, he was an assistant professor at the University of Waterloo, Canada. In 1992, he was appointed to the University of Erlangen-Nürnberg, Germany. Since 1999 he has been director of the Computer Graphics Group at the Max-Planck-Institute for Computer Science and cross-appointed to Saarland University in Saarbrücken, Germany. In his research, Seidel investigates algorithms for 3D image analysis and synthesis. This involves the complete processing chain from data acquisition over geometric modeling to image synthesis. In 2003, Seidel was awarded the Leibniz Preis, the most prestigious German research award, from the German Research Foundation (DFG). Seidel was the first computer graphics researcher to receive this award. In 2004, he was selected as founding chair of the Eurographics Awards Programme.



Philipp Slusallek is full professor at the Computer Graphics Group of Saarland University. In 1998/99, he was visiting assistant professor at the Stanford University graphics laboratory. His current research activities focus on real-time ray tracing on off-the-shelf computers and on designing a hardware architecture for real-time ray tracing. Other research topics include the design of a network-aware multi-media infrastructure, consistent illumination in virtual environments, physically-based and realistic image synthesis, and object-oriented software design.