ulm university universität
uulm

Two-Level Ray Tracing with Reordering for Highly Complex Scenes

J. Hanika, H.P.A. Lensch, A. Keller

Ulmer Informatik-Berichte

Nr. 2009-11
November 2009

Two-Level Ray Tracing with Reordering for Highly Complex Scenes

J. Hanika[1] and H. P. A. Lensch[1] and A. Keller[2]

[1]{johannes.hanika, hendrik.lensch}@uni-ulm.de, Ulm University, James-Franck-Ring, 89081 Ulm
[2]alex@mental.com, mental images GmbH, Fasanenstrasse 81, 10623 Berlin, Germany

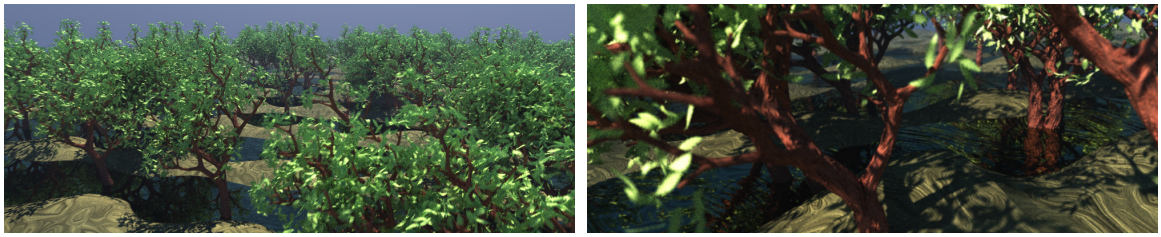


Figure 1: *A forest with 100 trees and geometry shaders including displacements for bark and leaves. Fully tessellated, it would consist of over* $108 \cdot 10^9$ *triangles. Thanks to level of detail on-demand geometry creation and ray sorting, only around 170 million triangles are actually created, without the use of caching. The geometry is created and fully path traced without instancing, evaluating global illumination from the sun and sky, and motion blur at a resolution of* 1920×768 *with 32 samples per pixel in 5:04 minutes on a 2.83 GHz quad core Q9550 (left image).*

Abstract

We introduce a ray tracing architecture which is able to handle highly complex geometry modeled by the classic production approach of surface patches tessellated to micro-polygons, where the number of micro-polygons can exceed the available memory. Two novel techniques allow us to carry out global illumination computations in such scenes and to trace the resulting incoherent sets of rays efficiently. For one, we introduce a technique for building the BVH over tessellated patches in time linear in the number of micro-polygons. Second, we present a two-stage ray tracing system which is highly parallel and minimizes the number of tessellation steps by reordering rays. The technique can accelerate rendering scenes of billions of micro-polygons and objects with complex reflection shaders using deferred shading.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.7]: Three-Dimensional Graphics and RealismRaytracing; Computer Graphics [I.3.3]: Picture/Image GenerationDisplay algorithms.

1. Introduction

In movie production, extreme geometric detail, complex reflection shaders and motion blur are needed to obtain visually convincing images. The Reyes architecture [CCC87] successfully deals with these challenges using a rasterization approach. The use of physically-based ray tracing is getting more and more common in the production industry, partly to be able to easily carry over artists' experiences from real-world lighting design, accepting long render times [Gri09]. To retain the strengths of the Reyes architecture in a general ray tracing setting, we propose a two-level hierarchy approach, using reordering of computations instead of caching. After traversing a top-level hierarchy, rays are sorted to bundle those intersecting the same bounding volume. Any necessary operation to be carried out in this volume, e.g. tessellation or loading a complex shader or BRDF, is thus performed a minimum number of times. This results in significantly improved data locality which allows us to fully ray trace computationally complex (procedural) displacements efficiently, i.e. corresponding to billions of micro-polygons without instancing (see Figure 1). Existing production pipelines can easily be extended to use our method, since the algorithm works on the same two-level data, such as displaced subdivision surfaces and sub-pixel-sized micro-polygons.

Rendering such scenes requires to tessellate the free form

patches or procedural displacements which can be quite expensive with regard to computation and memory consumption. We accelerate ray tracing and global illumination by exploiting the two-level hierarchy of such scenes: The top-level hierarchy (Section 3.1) organizes the list of surface patches. After traversing the top-level, all rays are sorted according to patches they possibly intersect, increasing locality and minimizing the number of tessellation steps. The bottom-level consists of the micro-polygons which are tessellated on-the-fly on demand. The micro-polygons of one patch are diced into a micro-polygon buffer, and a high-quality BVH is constructed in linear time in the number of micro-polygons (Section 3.2), exploiting the regular topology of a diced patch. Furthermore, the number of tessellation steps during rendering is effectively reduced by adapting the level of detail (Section 3.3).

Altogether, the architecture collapses the inherent recursive nature of ray tracing to allow for better vectorization and combines the strengths of tracing ray packets [WBWS01], fast incoherent mono-ray traversal [DHK08], and rasterization: Our technique inherently handles displacements and procedural geometry, supports simple shader writing and large depth complexity. It optimizes the utilization of memory bandwidth and coherence and furthermore is highly parallel.

2. Previous Work

A lot of work has been done to render complex geometry [SBB*06, LYM07, LYTM08] not specialized for the Reyes architecture.

The fundamental assumptions and design principles of the Reyes image rendering architecture have allowed to model and render diverse and complex content, as postulated in the original publication [CCC87]. The concepts were so fundamental, that the many extensions (e.g. [HL90, LV00]) seamlessly complemented the basic architecture. As one of the design principles was to keep expensive ray tracing to a minimum, it is not surprising that the addition of minimal ray tracing turned out to be restrictive. The most recent ray tracing extension was profoundly described in [CFLB06]. With our technique we demonstrate how a ray tracing system can deal with the same complexity in geometry modeling and shading while adding the benefit of simple Monte Carlo-based global illumination computation using path tracing.

Key to our system is the reordering of rays to increase locality for ray tracing massive data which, in rather general settings, has been investigated before [LMW90, PKGH97, NFL07, BBS*09]. Our approach, however, directly benefits from the intrinsic data locality of the common two-level modeling approach: large surface patches in the top-level, and displacements or procedural details and complex shading at the bottom-level. This approach is particularly common in games, for example using parallax occlusion mapping [Tat05]. Ray tracing displaced primitives using tessellations [SB87], caches [PH96] or direct grid-like traversal [SSS00] has been investigated in depth, also on the GPU for geometry images [CHCH06]. The GPU can also be used to dice/tessellate Reyes patches [PO08]. Acceleration structures for ray tracing have been build in complexity below $\mathcal{O}(N \log N)$ before [HMF07]. In Section 3.2 we show that our method is much simpler and matches current hardware better than their method.

In our two-level approach rays are reordered, grouping active rays which potentially intersect the same patch. Similar to the approach taken in the Kilauea render system [KS02] the resulting *(ray, patch)* lists can then be processed in parallel without additional caching strategies.

The Razor architecture [SMD*06] was designed to alleviate similar problems of ray tracing, to obtain better data access and computation patters. It uses current results of that time to accelerate ray tracing for the processors available by then. Today, cache lines (and fetches) get larger and data parallelism is getting wider, GPUs being the extreme example. Thus, linear memory access has become more important, and simple streaming of large blocks is often more efficient than highly recursive tree traversals and lazy builds with a lot of branches. The Razor system does not use displacements or pluggable artist-driven geometry shaders, so it is not optimized to avoid these computations, nor for the excessive level of detail needed for production. The hierarchies used here are also significantly different and more complicated than our algorithm. For their level of detail, they need a set of pairs of kd-trees for every two adjacent levels of detail, which are merged together in one acceleration structure. Additionally, at the lowest level, the vertices are stored in a 5x5 grid, which is created on demand and traversed as in [SSS00].

Our method on the other hand comes along with two levels of hierarchy, has implicit levels of detail (in the upper levels of the bottom-level QBVH), and can be diced, displaced and built with optimal memory access and data parallelism.

There exists an impressive system [PH96, PKGH97], which relies on a set of different caches for the rays, geometry and textures. However, processor architectures have moved on since then and, as we will show, caching does not work well in our setting. Additionally, our system is much simpler and transparently increases locality for rays, textures, BRDF data and geometry.

Level of detail (LOD) has been added to both Reyes [CHPR07] and ray tracing architectures, e.g. using multi-resolution meshes [SMD*06] or simplification [YM06]. For ray tracing, the choice of LOD is commonly based on ray differentials [Ige99]. We show that in the case of our architecture, simpler mechanisms may be used.

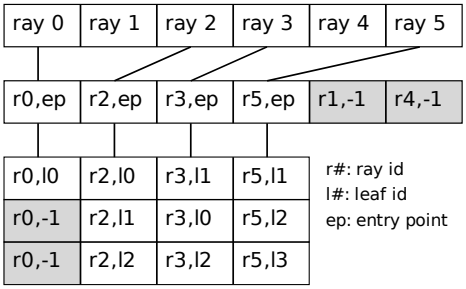


Figure 2: *The buffers used to sort the rays. Top: main buffer holding the actual ray structs, containing information such as hit distance, normal, ray origin, reciprocal direction. This buffer is not sorted and can be used to derive pixel indices. After one iteration of QBVH intersection, the second buffer is filled in parallel with entry points and all inactive rays are removed. Finally, all patches with a possible intersection on the way are stored in the third buffer. In this example, if another ray terminates, enough memory becomes available for each of the three remaining rays to store one more intersection in order to tackle a larger depth complexity, four in this case.*

3. A Two-Level Ray Tracing Hierarchy

Our rendering system follows the two-level modeling approach commonly applied by artists who often create coarse geometry from free form surfaces and refine it by adding geometry shaders and complex reflection shaders to them.

In order to minimize the number of dicing operations we introduce an active ray buffer. Directly after traversing the top-level hierarchy, all potential intersections are sorted by patch id. Then each patches reflection and geometry shader is prepared only once for this iteration. Lastly, all rays associated with the patch are now processed in one block of computation. As new rays might be generated due to recursive ray tracing the loop of top-level traversal, sorting, tessellation and bottom-level processing is iterated as needed. Finally, the result of the first hit point is added to the accumulation buffer.

Partitioning the computation this way greatly facilitates parallelization in each of the four processing steps. Even for global illumination computations where rays are typically incoherent after the first bounce, the explicit sorting step maximizes coherence for further processing.

3.1. Top-Level Hierarchy

Our approach is based on a QBVH [DHK08], whose leaves are the conservative bounding boxes of single patches. Then the ray buffer is filled with all available rays and each ray traverses this top-level hierarchy in turn, which can be executed in parallel by partitioning the ray buffer.

Instead of directly intersecting a ray with patches, a maximum of N first intersections of a ray with leaf bounding boxes are recorded in an array that keeps tuples of the form (`rayid`, `leafid`). In an optimal case all possible intersections would fit into this buffer. Given a number R of rays and a memory size M, N is proportional to M/R. This array is then sorted by `leafid` in order to reduce mulitple accesses to the same leaf node. This approach may resemble [NFL07], however, specific details are not disclosed in their work and only simulated memory traffic statistics are provided.

In another buffer, the last `leafid` is stored as an entry point for all active rays, so the actual ray buffer remains in original order which can be used to implicitly associate pixel positions and rays (see Figure 2).

For each `leafid` in the array, the leaf object is tessellated and the rays corresponding to the `leafid` are traced through the leaf object (see Section 3.2). In a parallel implementation each thread picks the next `leafid` as a task. Writing back intersection results to rays is either serialized by implementing a few locks for larger blocks of rays or, more efficiently, by writing the ray intersections to small buffers for each thread, which are synchronized at the end.

Once all rays are intersected, the top-level traversal is continued using the last `leafid` as an entry point. Since traversal is ordered by ray direction, it is always clear which children have already been processed when stepping up in the hierarchy.

Early termination is realized by intersecting the ray and a leaf bounding box prior to tessellation. The resulting number R' of remaining rays is then used to determine a larger $N' \sim M/R'$ similar to above. This way, the process does not have to be repeated often, as the depth complexity of most scenes (the forest scene in Figure 1 has an overdraw of about 200) is reached quickly.

This scheme enables two more optimizations. First, in the presence of shaders, which require to access large memory blocks (such as measured BRDF data), many rays intersecting the respective surface will have an early out event at the same time and thus the memory does not have to be accessed several times. Second, to further reduce the need for repeated dicing over generations of rays, the early termination event can be used to shade a terminated block of rays, and spawn new ray directions, which can directly be intersected with the already diced originating patch and then be re-injected into the top-level traversal.

3.2. Bottom-Level hierarchy

After the patches which might intersect a set of rays have been found by the top-level hierarchy, they have to be diced and displaced, evaluating geometry shaders. The resulting micro-polygons are stored in the micro-polygon

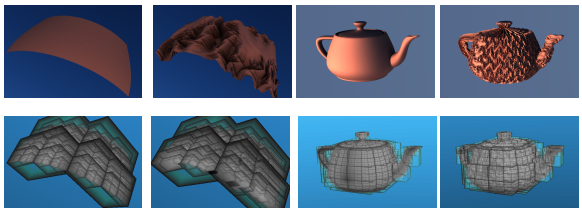


Figure 3: *The top row shows a surface patch and the teapot without (left) and with displacement mapping (right). In the bottom row the bounding volume hierarchies implied by the micro-polygon array topology are visualized by rendering them transparently and darkening their contours. Differences between the hierarchies are difficult to spot, which indicates that reasonable displacement does not much affect the efficiency of the implied acceleration data structure.*

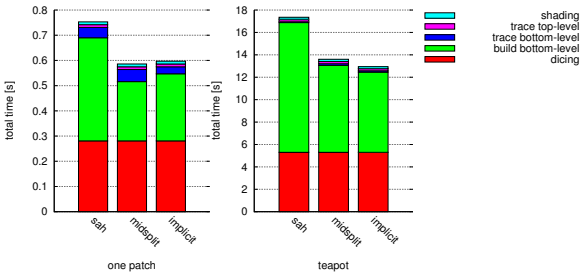


Figure 4: *Timings comparing bottom-level construction strategies. On the left, a full SAH build of the micropolygon BVH is done, in the middle, a spatial median was used as split plane candidate, on the right is implicit construction. The one-patch scene and the teapot is as is Figure 3 (displaced version). These timings have been taken for primary rays only.*

buffer, which represents $2^m \times 2^m$ micro-polygons as a two-dimensional array of $(2^m+1) \times (2^m+1)$ vertices, where each four adjacent vertices define one micro-polygon.

Surface patches must implement a tessellation method, that computes the micro-polygon vertices by either sampling or subdividing a surface patch, applies trimming and displacement, and stores interpolated (s,t) texture coordinates. Vertices are displaced along interpolated per-vertex displacement normals. Afterwards a loop over all micro-polygons evaluates whether or not the micro-polygon is clipped or trimmed. Unless the micro-polygon is discarded, its bounding box, color from texture, and normal by vertex differences are computed and stored.

Such a tessellation method must be aware of the resolution of the micro-polygon buffer. In case of insufficient resolution, surface patches must be split and the parts processed separately.

3.2.1. Construction in Linear Time

The number of $4^m = 2^m \times 2^m$ micro-polygons and their topology suggest using a complete quad-tree of axis-aligned bounding boxes as acceleration hierarchy for ray tracing.

Although in general, complete trees for ray tracing cannot be recommended [Wäc08, Sec. 2.4.1], this concept is very appropriate for tessellated surface patches: Unless the patch is overly curved or extremely displaced, the array topology very well represents spatial proximity as illustrated in Figure 3 and results in fast ray tracing (see Figure 4).

While typically the construction time for a spatial acceleration structure is $\mathcal{O}(n \log n)$ in the number of triangles [WH06, Wal07], our bottom-up construction of the complete quad-tree is linear in the number of nodes $\sum_{i=0}^{m} 4^i \in \mathcal{O}(4^m)$ and thus linear in the number of micro-polygons of one surface patch. In contrast to [HMF07], this can be done without explicit input hierarchy and in a non-recursive manner, thus featuring a better memory access pattern.

Ray tracing the micro-polygon buffer starts by marking all axis-aligned bounding boxes of the hierarchy as empty before calling the tessellation method. After the bounding boxes of the micro-polygons have been determined, the bounding volumes of the inner nodes of the hierarchy are updated in a bottom-up manner, similar to MIP maps. This is in fact similar to the min max MIP map utilized in [CHCH06]. Note that bounding boxes marked as empty do not need to update their parent boxes and also can be handled transparently during ray traversal. Since the memory for the micro-polygon buffer data structure is allocated once for the whole rendering process, it does not make sense to optimize for memory of the empty bounding boxes or omitted micro-polygons. Rays are then intersected with the acceleration structure using single ray traversal. Improved memory access by tracing ray packets did not pay off at this stage, as even these pre-sorted rays for one patch are very incoherent with regard to traversing the bottom-level hierarchy in the case of path tracing. We tessellate down to sub-pixel size and use the boxes of the leaf nodes directly as geometry, as this accuracy is sufficient [DK06].

To assess the quality of the implicit BVH construction for patches, we tested two very simple scenes, to avoid the effect of a complicated top-level hierarchy in the figures. In Figure 4, timings are plotted for a scene containing only one patch, and the displaced teapot scene (see Figure 3). The bottom-level ray tracing time can be slightly improved when using a general SAH build for the teapot (0.119 seconds SAH vs. 0.121 seconds implicit), for the simple one-patch scene, the implicit tree can even be ray traced faster (0.041 seconds SAH vs. 0.028 seconds implicit). This might be also due to the fact that our bottom-level QBVH traversal implementation exploits the special memory layout of the implicit

BVH, using skip lists. It also explains the difference to the ray tracing time of the midsplit tree (0.048 seconds), which results in quite similar topology.

3.3. Level of Detail

A common and very useful technique to increase efficiency when rendering complex models is level of detail (LOD), i.e. the use of simplified geometry in appropriate cases, for example for a tree which is far away and can hardly be seen. We facilitate this by choosing the parameter m from Section 3.2 accordingly.

There are three major problems when LOD is used: First, if two adjacent patches are tessellated in a different LOD, there can be cracks along the boundary of the patches. Second, a mechanism to identify the required LOD is needed. This is mostly done using ray differentials [Ige99], as a local approximation of the distance to the neighboring ray cast from the pixel raster. A third common problem is *popping*, i.e. objects suddenly appearing in more detail, which results in distracting, quick changes in animations. A fourth problem related to ray tracing is *self-intersection*, which is the problem that new rays emerged from a surface erroneously report an intersection with exactly this surface.

The problem of cracks can be avoided by stitching the adjacent geometry together [CFLB06, Sec. 6.6], but in our setting, this is not necessary. As tessellation is sufficiently fine such that the smallest boxes of the BVH can directly be used as primitives, cracks never appear: if the adjacent box is larger, it will span at least the area of the two smaller ones. Of course this requires coarse boxes to use conservative bounds during sampling of the patch. This can be achieved by min-max MIP-maps when using textures. In our case, we use interval arithmetic on the noise function.

Ray differentials are an approximation of the distance to neighboring rays hitting the same surface. In our system, we trace all rays at once. Thus, at first sight, there is no need for such an approximation, since the information about the other rays is at hand. In practice, it is a bit harder since we want to choose the LOD before tessellating a patch. That is, we need to decide for a LOD based on the patch bounding box and a group of rays intersecting this patch. Currently, we solve this by assuming uniform distribution of the ray directions and origins. Consequently, tessellation is done such that the number of resulting voxels is at least equal to the number of rays R intersecting this patch: $m := \min\{k | 4^k \geq R\}$. This is evaluated very fast and has the advantage that in regions of path space with blurry contributions (e.g. after a diffuse bounce), which will have a low ray density, coarse geometry (i.e. low LOD) will be used. On the other hand, specular reflections will keep rays close together, resulting in precise geometry.

This simple and fast non-recursive heuristic proved good enough for our purpose, but can fail if the assumption of uniform ray distribution is violated. Most notably, this is the case if a patch only partly overlaps the viewport. Then the visible part of the patch will have a much higher ray density as our heuristic assumes. This problem could be addressed by a small statistic to estimate the minimum distance of the rays.

If available in the rendering system, individual ray differentials can always be used per ray. Because the bottom-level acceleration structure is a complete quad-tree and the hit points are calculated as the intersection with a bounding box, the upper levels always represent the coarser levels of detail. The rest of the shading information (color from texture, uvs, normals from vertex differences) can then be filtered as needed and rays can be terminated at individual levels of detail. Note however that this will result in a slightly more complicated memory access pattern.

Of course LOD is an approximation that implies a compromise with quality, but the speedup is significant. For some extremely massive scenes it is the use of LOD which makes the difference whether a scene can be rendered or not. The forest in Figure 1 experienced a 50× speedup when using LOD compared to rendering it with a medium detailed, fixed LOD for all patches.

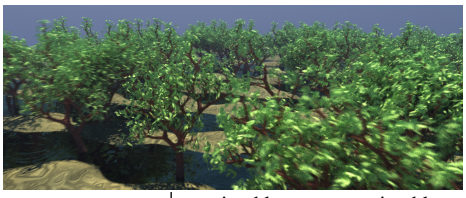
The popping problem is ameliorated by the fact that LOD is chosen to be sub-pixel accurate for lens connection rays. This way, no popping of directly visible geometry can take place. Unfortunately, for secondary effects as self-shadowing of a patch, popping can still become visible in form of a noticeable difference in shading. This can be alleviated by adding a-priori knowledge of rays to be spawned at the surface to the LOD decision. Other ways to create soft shadows are to complement the coarse levels with directional opacity information as in [LBBS08] or use frequency domain filtering [HSRG07].

To avoid self-intersection, the origin of the new ray has to be pushed outside the hit point voxel. This is done by an offset in normal direction of $\varepsilon = W/2^m$ and W the largest box width of the bounding box of the patch during bottom-level traversal. This LOD-dependent offset is applied during traversal, to ensure different LOD between the waves of rays are considered correctly.

3.4. Motion Blur

Motion approximated by linear splines is standard in production (see e.g. [CFLB06, Sec.6.3]). Given the instants $t_0 < t_1 < \cdots < t_n$ defining the time intervals $[t_i, t_{i+1})$, tracing a ray at time $t \in [t_i, t_{i+1})$ is accomplished by instancing two micro-polygon buffers, one at time t_i and one at time t_{i+1}. The actual bounding boxes and micro-polygons used during ray traversal then are determined by linear interpolation. We use this method for the bottom-level hierarchy.

Concerning the top-level hierarchy, the same principles can



	motion blur	no motion blur
dice [s]	211.06	104.10
bottom-level [s]	86.90	56.50
top-level [s]	27.70	24.76
shade [s]	25.20	25.36
sort [s]	34.84	28.82
#diced patches	1497633	1313136
total time [s]	179.0	133.0

Table 1: *Timings for the forest scene with exaggerated motion blur on a core2 quad. Motion blur results in a slowdown of a factor of two for the dicing stage, and micro-polygon intersection is slightly slower. As more patches have to be diced in the presence of motion blur, also the sorting time is increased. All times are total times, except dice and bottom-level times, which are accumulated over all four cores.*

be applied. However, due to the cost to construct the hierarchy, we chose to use only one hierarchy based on bounding boxes conservatively covering the whole time interval $[t_0, t_n)$.

See Table 1 for a comparison of render times with and without motion blur.

Figure 5: *A tree rendered using our architecture. If it was dumped to a triangle mesh, it would consist of around 8.4 billion triangles (micro-polygons from 12k displaced Bézier patches). Due to the lazy procedural geometry generation and the level of detail system, our system does not even create all these, and is able to render this scene with global illumination in a few minutes.*

3.5. Tracing Rays in Groups and by Generation

Physically-based rendering requires a lot of rays to be traced. This number is typically too large to fit the required ray buffer into main memory. Also, at the beginning, not all rays are known. Some effects (such as soft shadows, ambient occlusion, reflections and so on) require several passes to be rendered, i.e. another generation, or wave, of rays to be shot.

There are several choices, which balance depth complexity, re-dicing, and memory requirements:

- Re-inject rays as needed after an early termination event. This is done by replacing the terminated ray by a newly spawned one, instead of removing it from the buffer. This will always utilize the ray buffer well and use the (`rayid`, `leafid`) buffer for new rays rather than to tackle depth complexity (see Section 3.1)
- Group rays by generation. This fixes the memory requirements for this wave of rays, but suffers from re-dicing for each pass.
- Tile the screen. This can exploit some locality for first generation lens connection rays, but as rays quickly become divergent, re-dicing is as bad as in the previous variant.

Our current implementation uses the second approach. In general it is most efficient to trace as many rays as possible (i.e. fit into main memory) at a time.

For a simple path tracer, it is sufficient to update a single (spectral or RGB) path contribution value in the ray at each bounce. In the presence of complex reflection shaders with splitting into S sub-paths, each new ray needs to be assigned the correct weight $1/S$, but great care has to be taken not to exceed the buffer limits. A similar approach would be taken to implement ambient occlusion.

Bi-directional path tracing can be done by first tracing a wave of S paths from the sensor and T paths from the lights at the same time (resulting in $S+T$ rays at a time). After that, $S \cdot T$ connection rays have to be spawned with the respective weights, for example calculated using multiple importance sampling [VG95]. To bring the number of connection rays down to $S+T$ as well, Russian roulette based on these weights can be used.

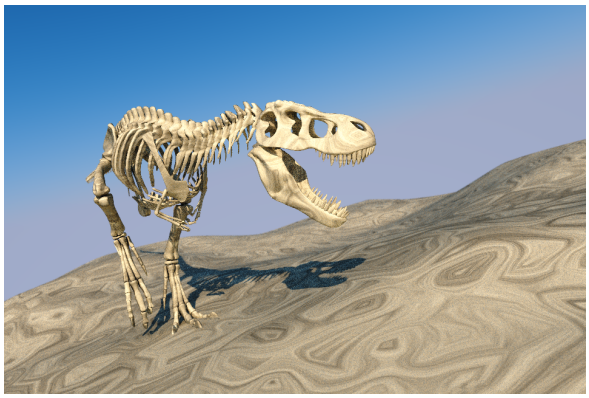


Figure 7: *A dinosaur (taken out of the natural history museum from the lighting challenge site and converted to Bézier patches using vertex normals), with 56k patches. On the left, around* 11 *million micro-polygons out of* 59 *billion possible have been created and rendered using path tracing in* 18 *seconds on a core2 quad. On the right, a noisy displacement has been applied, resulting in about* 34 *million created micro-polygons and an increased render time of* 37 *seconds. Both images are rendered at* $960 \times 640 \times 16$ *samples. Some of the time is spent in the (intentionally) expensive procedural displacement texture.*

Figure 6: *Stress test: this forest consists of two million patches, which would need a triangle mesh equivalent of more than 1050 billion triangles fully tessellated. Fully path traced with motion blur at* $1920 \times 768 \times 16$ *samples in 2:24 minutes on a four core machine.*

dicing	1000 trees	100 trees	memory needed
cache 10	1897385	1161468	4374 MB
cache 100	943669	772491	43732 MB
cache 1000	825808	606371	437320 MB
reordering	482405	354534	6600 MB

Table 2: *This table shows how often patches have to be diced using a cache with 10, 100 and 1000 patches and our reordering method. Numbers are acquired for the two forest scenes with motion blur at* $1920 \times 768 \times 64$ *rays, LOD* $m = 9$*. The caching method requires an exorbitant amount of memory and still does not reach our numbers by far.*

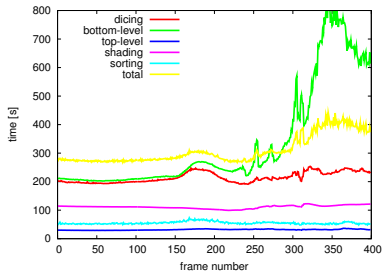


Figure 8: *Statistics for the video where Figure 1 are still frames from. These numbers have been generated on a core2 quad, using 32 samples per pixel in* 1920×768 *resolution, path traced up to three bounces of light interaction (plus evaluation of direct light at each bounce). Again all times are total times, except dice and bottom-level times, which are accumulated over all four cores. Near the end, the camera closes up to a branch, so one patch has to be tessellated very finely.*

4. Results

We implemented a Monte Carlo global illumination renderer on top of our ray tracing architecture. We chose a simple path tracer with next event estimation, i.e. paths are traced from the eye and are taken through a maximum of three bounces, additionally sampling the direct light contribution at each interaction point. At each point, Russian roulette is used to decide whether to sample the hemisphere or the light sources. This way, a maximum number of *width* $\times$ *height* $\times$ *samples per pixel* $\times$ *4* rays is traced per frame. While uncommon in movie production, this is a good demonstration of the generality of our method.

To achieve equivalent detail in a regular mesh-based renderer, the micro-polygons would have to be dumped to a triangle soup which would exceed the capacities of these

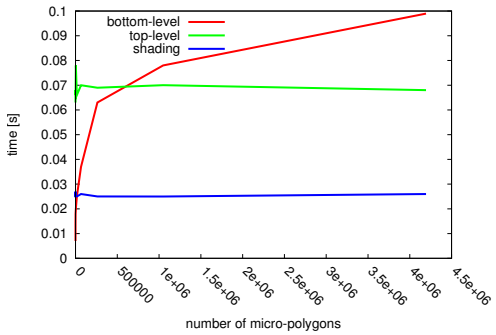


Figure 9: *Rendering time is determined by the programmable parts of the system, i.e. shading and surface patch tessellation (tessellation time is linear and takes over 100 seconds for $4 \cdot 10^6$ micro-polygons and is therefore omitted in the plot). Timings are obtained for the displaced teapot example (Figure 3).*

eye		bounce 1		bounce 2		bounce 3	
R	*N*	*R*	*N*	*R*	*N*	*R*	*N*
23592960	8	21589447	8	15585868	12	10052438	18
19009592	9	16419230	11	10886646	17	6707465	28
5840016	32	4549352	41	1529033	100	298166	100
1115814	100	406945	100	14093	100	82	100
1062	100	17	100	-	-	-	-
23592960	8	20674988	9	11985885	15	7311443	25
11479199	16	11145131	16	4617303	40	1775062	100
2110951	89	1241408	100	59364	100	204	100
7509	100	333	100	-	-	-	-

Table 3: *Tackling depth complexity: when rays are terminated early due to sorted BVH traversal, the memory can be used to store more patch intersections N for the remaining rays R. This is an example for the forest in Figure 6 (top table) and Figure 1 (bottom table), for the four waves of path tracing with next event estimation. N is capped at 100 to avoid excessive fragmentation of memory for simple cases.*

renderering systems. We therefore do not show comparisons with these. We tested the system on a variety of scenes ranging from trivial (Figure 3 left, equivalent to 260k triangles) over simple (Figure 3 right, equivalent to 16 million triangles), moderately complex (Figures 1,5 and 7) to massive scenes with detail equivalent to a mesh with over 1050 billion triangles (Figure 6), and scenes using reflection shaders accessing very large measured BRDF data sets (Figure 10). The trees are procedurally generated using L-systems with procedural displacement textures for the patches. The rest of the scenes is modeled in Bézier patches.

As all available rays are intersected with the scene at once before shading is started, complex reflection shaders benefit from this deferred shading architecture. If a second sorting step is inserted after all ray intersections have been found, even one single data load operation per bounce can be guaranteed. This is necessary, if not all used BRDF data sets fit

scene	maximum	accessed
100 trees (Figure 1)	$108 \cdot 10^9$	$170 \cdot 10^6$
1000 trees (Figure 6)	$1,058 \cdot 10^9$	$317 \cdot 10^6$
dinosaur (Figure 7, left)	$59 \cdot 10^9$	$11 \cdot 10^6$
displaced dinosaur (Figure 7, right)	$59 \cdot 10^9$	$34 \cdot 10^6$

Table 4: *Impact of LOD and early ray termination due to occlusion: ratio of micro-polygons actually created to a constant LOD of $m = 9$.*

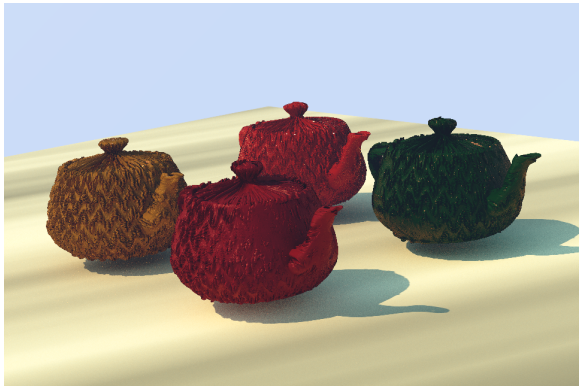


Figure 10: *Four teapots rendered with measured BRDF data. One BRDF data set alone is over 300 MB large, and each teapot consists of over 16 million triangles. Thanks to reordering of shading computations, it can be guaranteed that each BRDF data set is loaded only once per bounce.*

into main memory at the time. In the case of our test scene (Figure 10), this was not necessary, but the time spent to sort the ray buffer can almost be neglected compared to the time spent in shading (less than 10 seconds compared to 263 seconds for shading $960 \times 640 \times 64$ rays).

We implemented a caching based system as a comparison, similar to [PH96]. Our results in Table 2 indicate that for highly complex scenes caches need to be very large. With our reordering method, they are not even necessary, and the implementation becomes thus a lot simpler than [PKGH97].

For a single pass path tracing at 1920x768x64 rays, our method needs 6120+480 MB, (68 bytes per ray plus one diced patch, note that this includes acceleration structure, textures and normals) as compared to 43732 MB needed for a geometry cache with only 100 cachelines for LOD $m = 9$ (i.e. 512×512 micro-quads with uvs and normals at two time instances) preferably for each thread. This cache would not even with 1000 lines match our dicing numbers, which dominate render time.

For an example how depth complexity is handled by the limited `(rayid, leafid)` buffer (see Section 3.1), see Table 3. It can be seen that even for scenes with very large overdraw, only few iterations are required. As some rays ter-

minate, the buffer is quickly available for the remaining rays to store many more intersection candidates in the next iteration.

To get an impression of the impact of LOD, see Figure 9. The rendering times are dominated by dicing, so the graph shows only timings for top- and bottom-level traversal and shading. As expected, dicing and bottom-level tree construction seems to be linear, and tracing bottom-level rays about logarithmic. Top-level traversal does not change in this chart, as the top-level hierarchy is not affected by the LOD changes. Obviously a lot of time can be saved by reducing the number of micro-polygons per patch.

Our system does this very effectively by avoiding dicing for occluded patches altogether on the one hand, and choosing LOD on the other. Table 4 illustrates this. The comparison here is done between actually created micro-polygons and the number of polygons in a triangle mesh with equivalent detail the maximum LOD $m = 9$. Note that this number is not overly large, this LOD is also chosen for some patches by the algorithm and becomes especially necessary for heavily displaced patches. The figures show that our system can robustly handle a vast amount of geometry, which surpasses the complexity demonstated by the Razor system [SMD*06]. Also, we do not need to keep any diced micro-polygons which avoids the problem of flushing on demand geometry. In contrast, the Razor system explicitly stores all data, and only discards everything when a memory overflow trigger is hit.

5. Conclusion

We presented a ray tracing method which is able to efficiently handle large amounts of data resulting from free form surface patches, details added by micro-polygon tessellation and data intensive reflection shaders. Expensive (in terms of computation or memory access) geometry and reflection shaders are handled well, due to reordering of computations which results in great data locality. Parallelization is simple as all rays traverse one phase before the next one is started. We introduced only one additional sorting step on the ray buffer, which has negligible impact on rendering time.

There are no restrictions imposed on ray tracing. However, there are some limitations when using the method in a rendering system. First, the shading language needs a mechanism to dispatch a ray and correctly account for its contribution by the time it finishes (see Section 3.5). For physically-based rendering, this is not a problem, since a BRDF can be handled transparently.

Second, the presented LOD assumes good importance sampling. That is, it assumes that if rays are diverging, the contributing radiance is low frequency. It will thus result in blocky shadows if sampling a small direct light source over the hemisphere instead of the geometry. If this is recognized by the rendering algorithm, it can be used as a feature to speed up low-frequency indirect illumination.

In future work, the method can be complemented by specialized rendering algorithms which better exploit its strengths by reflecting the two-level nature, such as local high-frequency ambient occlusion inside a diced patch together with a far-field approximation for global illumination, similar to [KK04, AFO05].

Acknowledgements

The first author wishes to thank Holger Dammertz for countless productive discussions, Leonhard Grünschloß for proof reading, mental images GmbH for funding, and Matthias Hullin for the measured BRDF data sets.

References

[AFO05] ARIKAN O., FORSYTH D., O'BRIEN J.: Fast and detailed approximate global illumination by irradiance decomposition. *ACM Transactions on Graphics (Proc. SIGGRAPH 2005)* (2005), 1108–1114.

[BBS*09] BUDGE B., BERNARDIN T., STUART J., SENGUPTA S., JOY K., OWENS J.: Out-of-core data management for path tracing on hybrid resources. In *Computer Graphics Forum (Proc. of Eurographics 2009)* (2009), pp. 385–396.

[CCC87] COOK R., CARPENTER L., CATMULL E.: The REYES image rendering architecture. *Computer Graphics (Proc. SIGGRAPH '87)* (1987), 95–102.

[CFLB06] CHRISTENSEN P., FONG J., LAUR D., BATALI D.: Ray tracing for the movie 'Cars'. In *Proc. 2006 IEEE Symposium on Interactive Ray Tracing* (2006), pp. 73–78.

[CHCH06] CARR N., HOBEROCK J., CRANE K., HART J.: Fast GPU ray tracing of dynamic meshes using geometry images. In *GI '06: Proceedings of the 2006 conference on Graphics interface* (2006), pp. 203–209.

[CHPR07] COOK R., HALSTEAD J., PLANCK M., RYU D.: Stochastic simplification of aggregate detail. *ACM Transactions on Graphics 26*, 3 (2007), 79.

[DHK08] DAMMERTZ H., HANIKA J., KELLER A.: Shallow bounding volume hierarchies for fast SIMD ray tracing of incoherent rays. In *Computer Graphics Forum (Proc. 19th Eurographics Symposium on Rendering)* (2008), pp. 1225–1234.

[DK06] DAMMERTZ H., KELLER A.: Improving ray tracing precision by world space intersection computation. In *Proc. 2006 IEEE Symposium on Interactive Ray Tracing* (2006), pp. 25–32.

[Gri09] GRITZ L.: Production perspectives on high performance graphics. Keynote Talk at HPG09, 2009.

[HL90] HANRAHAN P., LAWSON J.: A language for shading and lighting calculations. *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (1990), 289–298.

[HMF07] HUNT W., MARK W. R., FUSSELL D.: Fast and lazy build of acceleration structures from scene hierarchies. In *Proc. 2007 IEEE/EG Symposium on Interactive Ray Tracing* (2007), pp. 47–54.

[HSRG07] HAN C., SUN B., RAMAMOORTHI R., GRINSPUN E.: Frequency domain normal map filtering. *ACM Transactions on Graphics (Proc. SIGGRAPH 2007)* (2007), 28.

[Ige99] IGEHY H.: Tracing ray differentials. *ACM Transactions on Graphics (Proc. SIGGRAPH 1999)* (1999), 179–186.

[KK04] KOLLIG T., KELLER A.: Illumination in the presence of weak singularities. In *Proc. Monte Carlo and Quasi-Monte Carlo Methods 2002*. Springer, 2004, pp. 245–257.

[KS02] KATO T., SAITO J.: Kilauea parallel global illumination renderer. In *Fourth Eurographics Workshop on Parallel Graphics and Visualization* (2002), pp. 7–13.

[LBBS08] LACEWELL D., BURLEY B., BOULOS S., SHIRLEY P.: Raytracing prefiltered occlusion for aggregate geometry. In *IEEE Symposium on Interactive Raytracing 2008* (2008), pp. 19–26.

[LMW90] LAMPARTER B., MÜLLER H., WINCKLER J.: *The Ray-z-Buffer—An Approach for Ray Tracing Arbitrarily Large Scenes*. Tech. rep., Albert-Ludwigs University at Freiburg, 1990.

[LV00] LOKOVIC T., VEACH E.: Deep shadow maps. *ACM Transactions on Graphics (Proc. SIGGRAPH 2000)* (2000), 385–392.

[LYM07] LAUTERBACH C., YOON S.-E., MANOCHA D.: Ray-strips: A compact mesh representation for interactive ray tracing. In *Proc. 2007 IEEE/EG Symposium on Interactive Ray Tracing* (2007), pp. 19–26.

[LYTM08] LAUTERBACH C., YOON S.-E., TANG M., MANOCHA D.: Reducem: Interactive and memory efficient ray tracing of large models. *Computer Graphics Forum 27*, 4 (2008), 1313–1321.

[NFL07] NAVRÁTIL P., FUSSELL D., LIN C.: Dynamic ray scheduling to improve ray coherence and bandwidth utilization. In *Proc. 2007 IEEE/EG Symposium on Interactive Ray Tracing* (2007), pp. 95–104.

[PH96] PHARR M., HANRAHAN P.: Geometry caching for ray-tracing displacement maps. In *Eurographics Rendering Workshop 1996* (1996), pp. 31–40.

[PKGH97] PHARR M., KOLB C., GERSHBEIN R., HANRAHAN P.: Rendering complex scenes with memory-coherent ray tracing. *ACM Transactions on Graphics (Proc. SIGGRAPH 1997)* (1997), 101–108.

[PO08] PATNEY A., OWENS J.: Real-time Reyes-style adaptive surface subdivision. *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers* (2008), 143:1–143:8.

[SB87] SNYDER J., BARR A.: Ray tracing complex models containing surface tessellations. *Computer Graphics (Proc. SIGGRAPH '87)* (1987), 119–128.

[SBB*06] STEPHENS A., BOULOS S., BIGLER J., WALD I., PARKER S. G.: An application of scalable massive model interaction using shared memory systems. In *Proceedings of the 2006 Eurographics Symposium on Parallel Graphics and Visualization* (2006), pp. 19–26.

[SMD*06] STOLL G., MARK W., DJEU P., WANG R., ELHASSAN I.: *Razor: An architecture for dynamic multiresolution ray tracing*. Technical report 06-21, Department of Computer Science, University of Texas at Austin, 2006.

[SSS00] SMITS B., SHIRLEY P., STARK M.: Direct ray tracing of displacement mapped triangles. In *Proc. Eurographics Workshop on Rendering Techniques 2000* (2000), pp. 307–318.

[Tat05] TATARCHUK N.: Practical dynamic parallax occlusion mapping. *ACM SIGGRAPH 2005 Sketches* (2005), 106.

[VG95] VEACH E., GUIBAS L.: Optimally compining sampling techniques for Monte Carlo rendering. *ACM Transactions on Graphics (Proc. SIGGRAPH 1995)* (1995), 419–428.

[Wäc08] WÄCHTER C.: *Quasi-Monte Carlo Light Transport Simulation by Efficient Ray Tracing*. PhD thesis, Universität Ulm, 2008.

[Wal07] WALD I.: On fast construction of SAH based bounding volume hierarchies. In *Proc. 2007 IEEE/EG Symposium on Interactive Ray Tracing* (2007), pp. 33–40.

[WBWS01] WALD I., BENTHIN C., WAGNER M., SLUSALLEK P.: Interactive rendering with coherent ray tracing. In *Computer Graphics Forum (Proc. Eurographics 2001)* (2001), pp. 153–164.

[WH06] WALD I., HAVRAN V.: On building fast *k*d-trees for ray tracing, and on doing that in O(N log N). In *Proc. 2006 IEEE Symposium on Interactive Ray Tracing* (2006), pp. 18–20.

[YM06] YOON S.-E., MANOCHA D.: R-LODs: fast LOD-based ray tracing of massive models. *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches* (2006), 67.

Liste der bisher erschienenen Ulmer Informatik-Berichte

Einige davon sind per FTP von `ftp.informatik.uni-ulm.de` erhältlich
Die mit * markierten Berichte sind vergriffen

List of technical reports published by the University of Ulm

Some of them are available by FTP from `ftp.informatik.uni-ulm.de`
Reports marked with * are out of print

91-01 *Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe*
Instance Complexity

91-02* *K. Gladitz, H. Fassbender, H. Vogler*
Compiler-Based Implementation of Syntax-Directed Functional Programming

91-03* *Alfons Geser*
Relative Termination

91-04* *J. Köbler, U. Schöning, J. Toran*
Graph Isomorphism is low for PP

91-05 *Johannes Köbler, Thomas Thierauf*
Complexity Restricted Advice Functions

91-06* *Uwe Schöning*
Recent Highlights in Structural Complexity Theory

91-07* *F. Green, J. Köbler, J. Toran*
The Power of Middle Bit

91-08* *V.Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf*
Reductions for Sets of Low Information Content

92-01* *Vikraman Arvind, Johannes Köbler, Martin Mundhenk*
On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets

92-02* *Thomas Noll, Heiko Vogler*
Top-down Parsing with Simulataneous Evaluation of Noncircular Attribute Grammars

92-03 *Fakultät für Informatik*
17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen

92-04* *V. Arvind, J. Köbler, M. Mundhenk*
Lowness and the Complexity of Sparse and Tally Descriptions

92-05* *Johannes Köbler*
Locating P/poly Optimally in the Extended Low Hierarchy

92-06* *Armin Kühnemann, Heiko Vogler*
Synthesized and inherited functions -a new computational model for syntax-directed semantics

92-07* *Heinz Fassbender, Heiko Vogler*
A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing

92-08* *Uwe Schöning*
On Random Reductions from Sparse Sets to Tally Sets

92-09* *Hermann von Hasseln, Laura Martignon*
Consistency in Stochastic Network

92-10 *Michael Schmitt*
A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function

92-11 *Johannes Köbler, Seinosuke Toda*
On the Power of Generalized MOD-Classes

92-12 *V. Arvind, J. Köbler, M. Mundhenk*
Reliable Reductions, High Sets and Low Sets

92-13 *Alfons Geser*
On a monotonic semantic path ordering

92-14* *Joost Engelfriet, Heiko Vogler*
The Translation Power of Top-Down Tree-To-Graph Transducers

93-01 *Alfred Lupper, Konrad Froitzheim*
AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager

93-02 *M.H. Scholl, C. Laasch, C. Rich, H.-J. Schek, M. Tresch*
The COCOON Object Model

93-03 *Thomas Thierauf, Seinosuke Toda, Osamu Watanabe*
On Sets Bounded Truth-Table Reducible to P-selective Sets

93-04 *Jin-Yi Cai, Frederic Green, Thomas Thierauf*
On the Correlation of Symmetric Functions

93-05 *K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam*
A Conceptual Approach to an Open Hospital Information System

93-06 *Klaus Gaßner*
Rechnerunterstützung für die konzeptuelle Modellierung

93-07 *Ullrich Keßler, Peter Dadam*
Towards Customizable, Flexible Storage Structures for Complex Objects

94-01 *Michael Schmitt*
On the Complexity of Consistency Problems for Neurons with Binary Weights

94-02 *Armin Kühnemann, Heiko Vogler*
A Pumping Lemma for Output Languages of Attributed Tree Transducers

94-03 *Harry Buhrman, Jim Kadin, Thomas Thierauf*
On Functions Computable with Nonadaptive Queries to NP

94-04 *Heinz Faßbender, Heiko Vogler, Andrea Wedel*
Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers

94-05 *V. Arvind, J. Köbler, R. Schuler*
On Helping and Interactive Proof Systems

94-06 *Christian Kalus, Peter Dadam*
Incorporating record subtyping into a relational data model

94-07 *Markus Tresch, Marc H. Scholl*
A Classification of Multi-Database Languages

94-08 *Friedrich von Henke, Harald Rueß*
Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

94-09 *F.W. von Henke, A. Dold, H. Rueß, D. Schwier, M. Strecker*
Construction and Deduction Methods for the Formal Development of Software

94-10 *Axel Dold*
Formalisierung schematischer Algorithmen

94-11 *Johannes Köbler, Osamu Watanabe*
New Collapse Consequences of NP Having Small Circuits

94-12 *Rainer Schuler*
On Average Polynomial Time

94-13 *Rainer Schuler, Osamu Watanabe*
Towards Average-Case Complexity Analysis of NP Optimization Problems

94-14 *Wolfram Schulte, Ton Vullinghs*
Linking Reactive Software to the X-Window System

94-15 *Alfred Lupper*
Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen

94-16 *Robert Regn*
Verteilte Unix-Betriebssysteme

94-17 *Helmuth Partsch*
Again on Recognition and Parsing of Context-Free Grammars:
Two Exercises in Transformational Programming

94-18 *Helmuth Partsch*
Transformational Development of Data-Parallel Algorithms: an Example

95-01 *Oleg Verbitsky*
On the Largest Common Subgraph Problem

95-02 *Uwe Schöning*
Complexity of Presburger Arithmetic with Fixed Quantifier Dimension

95-03 *Harry Buhrman,Thomas Thierauf*
The Complexity of Generating and Checking Proofs of Membership

95-04 *Rainer Schuler, Tomoyuki Yamakami*
Structural Average Case Complexity

95-05 *Klaus Achatz, Wolfram Schulte*
Architecture Indepentent Massive Parallelization of Divide-And-Conquer Algorithms

95-06 *Christoph Karg, Rainer Schuler*
Structure in Average Case Complexity

95-07 *P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe*
ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen

95-08 *Jürgen Kehrer, Peter Schulthess*
Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik

95-09 *Hans-Jörg Burtschick, Wolfgang Lindner*
On Sets Turing Reducible to P-Selective Sets

95-10 *Boris Hartmann*
Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper

95-12 *Klaus Achatz, Wolfram Schulte*
Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists

95-13 *Andrea Mößle, Heiko Vogler*
Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes

95-14 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
A Generic Specification for Verifying Peephole Optimizations

96-01 *Ercüment Canver, Jan-Tecker Gayen, Adam Moik*
Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE

96-02 *Bernhard Nebel*
Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class

96-03 *Ton Vullinghs, Wolfram Schulte, Thilo Schwinn*
An Introduction to TkGofer

96-04 *Thomas Beuter, Peter Dadam*
Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering

96-05 *Gerhard Schellhorn, Wolfgang Ahrendt*
Verification of a Prolog Compiler - First Steps with KIV

96-06 *Manindra Agrawal, Thomas Thierauf*
Satisfiability Problems

96-07 *Vikraman Arvind, Jacobo Torán*
A nonadaptive NC Checker for Permutation Group Intersection

96-08 *David Cyrluk, Oliver Möller, Harald Rueß*
An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction

96-09 *Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte*
Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT–Ansätzen

96-10 *Falk Bartels, Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Formalizing Fixed-Point Theory in PVS

96-11 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Mechanized Semantics of Simple Imperative Programming Constructs

96-12 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Generic Compilation Schemes for Simple Programming Constructs

96-13 *Klaus Achatz, Helmuth Partsch*
From Descriptive Specifications to Operational ones: A Powerful Transformation Rule, its Applications and Variants

97-01 *Jochen Messner*
Pattern Matching in Trace Monoids

97-02 *Wolfgang Lindner, Rainer Schuler*
A Small Span Theorem within P

97-03 *Thomas Bauer, Peter Dadam*
A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration

97-04 *Christian Heinlein, Peter Dadam*
Interaction Expressions - A Powerful Formalism for Describing Inter-Workflow Dependencies

97-05 *Vikraman Arvind, Johannes Köbler*
On Pseudorandomness and Resource-Bounded Measure

97-06 *Gerhard Partsch*
Punkt-zu-Punkt- und Mehrpunkt-basierende LAN-Integrationsstrategien für den digitalen Mobilfunkstandard DECT

97-07 *Manfred Reichert, Peter Dadam*
$ADEPT_{flex}$ - Supporting Dynamic Changes of Workflows Without Loosing Control

97-08 *Hans Braxmeier, Dietmar Ernst, Andrea Mößle, Heiko Vogler*
The Project NoName - A functional programming language with its development environment

97-09 *Christian Heinlein*
Grundlagen von Interaktionsausdrücken

97-10 *Christian Heinlein*
Graphische Repräsentation von Interaktionsausdrücken

97-11 *Christian Heinlein*
Sprachtheoretische Semantik von Interaktionsausdrücken

97-12 *Gerhard Schellhorn, Wolfgang Reif*
Proving Properties of Finite Enumerations: A Problem Set for Automated Theorem Provers

97-13 *Dietmar Ernst, Frank Houdek, Wolfram Schulte, Thilo Schwinn*
Experimenteller Vergleich statischer und dynamischer Softwareprüfung für eingebettete Systeme

97-14 *Wolfgang Reif, Gerhard Schellhorn*
Theorem Proving in Large Theories

97-15 *Thomas Wennekers*
Asymptotik rekurrenter neuronaler Netze mit zufälligen Kopplungen

97-16 *Peter Dadam, Klaus Kuhn, Manfred Reichert*
Clinical Workflows - The Killer Application for Process-oriented Information Systems?

97-17 *Mohammad Ali Livani, Jörg Kaiser*
EDF Consensus on CAN Bus Access in Dynamic Real-Time Applications

97-18 *Johannes Köbler,Rainer Schuler*
Using Efficient Average-Case Algorithms to Collapse Worst-Case Complexity Classes

98-01 *Daniela Damm, Lutz Claes, Friedrich W. von Henke, Alexander Seitz, Adelinde Uhrmacher, Steffen Wolf*
Ein fallbasiertes System für die Interpretation von Literatur zur Knochenheilung

98-02 *Thomas Bauer, Peter Dadam*
Architekturen für skalierbare Workflow-Management-Systeme - Klassifikation und Analyse

98-03 *Marko Luther, Martin Strecker*
A guided tour through *Typelab*

98-04 *Heiko Neumann, Luiz Pessoa*
Visual Filling-in and Surface Property Reconstruction

98-05 *Ercüment Canver*
Formal Verification of a Coordinated Atomic Action Based Design

98-06 *Andreas Küchler*
On the Correspondence between Neural Folding Architectures and Tree Automata

98-07 *Heiko Neumann, Thorsten Hansen, Luiz Pessoa*
Interaction of ON and OFF Pathways for Visual Contrast Measurement

98-08 *Thomas Wennekers*
Synfire Graphs: From Spike Patterns to Automata of Spiking Neurons

98-09 *Thomas Bauer, Peter Dadam*
Variable Migration von Workflows in *ADEPT*

98-10 *Heiko Neumann, Wolfgang Sepp*
Recurrent V1 – V2 Interaction in Early Visual Boundary Processing

98-11 *Frank Houdek, Dietmar Ernst, Thilo Schwinn*
Prüfen von C–Code und Statmate/Matlab–Spezifikationen: Ein Experiment

98-12 *Gerhard Schellhorn*
Proving Properties of Directed Graphs: A Problem Set for Automated Theorem Provers

98-13 *Gerhard Schellhorn, Wolfgang Reif*
Theorems from Compiler Verification: A Problem Set for Automated Theorem Provers

98-14 *Mohammad Ali Livani*
SHARE: A Transparent Mechanism for Reliable Broadcast Delivery in CAN

98-15 *Mohammad Ali Livani, Jörg Kaiser*
Predictable Atomic Multicast in the Controller Area Network (CAN)

99-01 *Susanne Boll, Wolfgang Klas, Utz Westermann*
A Comparison of Multimedia Document Models Concerning Advanced Requirements

99-02 *Thomas Bauer, Peter Dadam*
Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation

99-03 *Uwe Schöning*
On the Complexity of Constraint Satisfaction

99-04 *Ercument Canver*
Model-Checking zur Analyse von Message Sequence Charts über Statecharts

99-05 *Johannes Köbler, Wolfgang Lindner, Rainer Schuler*
Derandomizing RP if Boolean Circuits are not Learnable

99-06 *Utz Westermann, Wolfgang Klas*
Architecture of a DataBlade Module for the Integrated Management of Multimedia Assets

99-07 *Peter Dadam, Manfred Reichert*
Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications. Paderborn, Germany, October 6, 1999, GI–Workshop Proceedings, Informatik ’99

99-08 *Vikraman Arvind, Johannes Köbler*
Graph Isomorphism is Low for ZPP^{NP} and other Lowness results

99-09 *Thomas Bauer, Peter Dadam*
Efficient Distributed Workflow Management Based on Variable Server Assignments

2000-02 *Thomas Bauer, Peter Dadam*
Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT

2000-03 *Gregory Baratoff, Christian Toepfer, Heiko Neumann*
Combined space-variant maps for optical flow based navigation

2000-04 *Wolfgang Gehring*
Ein Rahmenwerk zur Einführung von Leistungspunktsystemen

2000-05 *Susanne Boll, Christian Heinlein, Wolfgang Klas, Jochen Wandel*
Intelligent Prefetching and Buffering for Interactive Streaming of MPEG Videos

2000-06 *Wolfgang Reif, Gerhard Schellhorn, Andreas Thums*
Fehlersuche in Formalen Spezifikationen

2000-07 *Gerhard Schellhorn, Wolfgang Reif (eds.)*
FM-Tools 2000: The 4th Workshop on Tools for System Design and Verification

2000-08 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Effiziente Durchführung von Prozessmigrationen in verteilten Workflow-Management-Systemen

2000-09 *Thomas Bauer, Peter Dadam*
Vermeidung von Überlastsituationen durch Replikation von Workflow-Servern in ADEPT

2000-10 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Adaptives und verteiltes Workflow-Management

2000-11 *Christian Heinlein*
Workflow and Process Synchronization with Interaction Expressions and Graphs

2001-01 *Hubert Hug, Rainer Schuler*
DNA-based parallel computation of simple arithmetic

2001-02 *Friedhelm Schwenker, Hans A. Kestler, Günther Palm*
3-D Visual Object Classification with Hierarchical Radial Basis Function Networks

2001-03 *Hans A. Kestler, Friedhelm Schwenker, Günther Palm*
RBF network classification of ECGs as a potential marker for sudden cardiac death

2001-04 *Christian Dietrich, Friedhelm Schwenker, Klaus Riede, Günther Palm*
Classification of Bioacoustic Time Series Utilizing Pulse Detection, Time and Frequency Features and Data Fusion

2002-01 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata

2002-02 *Walter Guttmann*
Deriving an Applicative Heapsort Algorithm

2002-03 *Axel Dold, Friedrich W. von Henke, Vincent Vialard, Wolfgang Goerigk*
A Mechanically Verified Compiling Specification for a Realistic Compiler

2003-01 *Manfred Reichert, Stefanie Rinderle, Peter Dadam*
A Formal Framework for Workflow Type and Instance Changes Under Correctness Checks

2003-02 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Supporting Workflow Schema Evolution By Efficient Compliance Checks

2003-03 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values

2003-04 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
On Dealing With Semantically Conflicting Business Process Changes.

2003-05 *Christian Heinlein*
Dynamic Class Methods in Java

2003-06 *Christian Heinlein*
Vertical, Horizontal, and Behavioural Extensibility of Software Systems

2003-07 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values
(Corrected Version)

2003-08 *Changling Liu, Jörg Kaiser*
Survey of Mobile Ad Hoc Network Routing Protocols)

2004-01 *Thom Frühwirth, Marc Meister (eds.)*
First Workshop on Constraint Handling Rules

2004-02 *Christian Heinlein*
Concept and Implementation of C+++, an Extension of C++ to Support User-Defined Operator Symbols and Control Structures

2004-03 *Susanne Biundo, Thom Frühwirth, Günther Palm(eds.)*
Poster Proceedings of the 27th Annual German Conference on Artificial Intelligence

2005-01 *Armin Wolf, Thom Frühwirth, Marc Meister (eds.)*
19th Workshop on (Constraint) Logic Programming

2005-02 *Wolfgang Lindner (Hg.), Universität Ulm , Christopher Wolf (Hg.) KU Leuven*
2. Krypto-Tag – Workshop über Kryptographie, Universität Ulm

2005-03 *Walter Guttmann, Markus Maucher*
Constrained Ordering

2006-01 *Stefan Sarstedt*
Model-Driven Development with ACTIVECHARTS, Tutorial

2006-02 *Alexander Raschke, Ramin Tavakoli Kolagari*
Ein experimenteller Vergleich zwischen einer plan-getriebenen und einer leichtgewichtigen Entwicklungsmethode zur Spezifikation von eingebetteten Systemen

2006-03 *Jens Kohlmeyer, Alexander Raschke, Ramin Tavakoli Kolagari*
Eine qualitative Untersuchung zur Produktlinien-Integration über Organisationsgrenzen hinweg

2006-04 *Thorsten Liebig*
Reasoning with OWL - System Support and Insights –

2008-01 *H.A. Kestler, J. Messner, A. Müller, R. Schuler*
On the complexity of intersecting multiple circles for graphical display

2008-02 *Manfred Reichert, Peter Dadam, Martin Jurisch,l Ulrich Kreher, Kevin Göser, Markus Lauer*
Architectural Design of Flexible Process Management Technology

2008-03 *Frank Raiser*
Semi-Automatic Generation of CHR Solvers from Global Constraint Automata

2008-04 *Ramin Tavakoli Kolagari, Alexander Raschke, Matthias Schneiderhan, Ian Alexander*
Entscheidungsdokumentation bei der Entwicklung innovativer Systeme für produktlinien-basierte Entwicklungsprozesse

2008-05 *Markus Kalb, Claudia Dittrich, Peter Dadam*
Support of Relationships Among Moving Objects on Networks

2008-06 *Matthias Frank, Frank Kargl, Burkhard Stiller (Hg.)*
WMAN 2008 – KuVS Fachgespräch über Mobile Ad-hoc Netzwerke

2008-07 *M. Maucher, U. Schöning, H.A. Kestler*
An empirical assessment of local and population based search methods with different degrees of pseudorandomness

2008-08 *Henning Wunderlich*
Covers have structure

2008-09 *Karl-Heinz Niggl, Henning Wunderlich*
Implicit characterization of FPTIME and NC revisited

2008-10 *Henning Wunderlich*
On span-P^{cc} and related classes in structural communication complexity

2008-11 *M. Maucher, U. Schöning, H.A. Kestler*
On the different notions of pseudorandomness

2008-12 *Henning Wunderlich*
On Toda's Theorem in structural communication complexity

2008-13 *Manfred Reichert, Peter Dadam*
Realizing Adaptive Process-aware Information Systems with ADEPT2

2009-01 *Peter Dadam, Manfred Reichert*
The ADEPT Project: A Decade of Research and Development for Robust and Fexible Process Support
Challenges and Achievements

2009-02 *Peter Dadam, Manfred Reichert, Stefanie Rinderle-Ma, Kevin Göser, Ulrich Kreher, Martin Jurisch*
Von ADEPT zur AristaFlow® BPM Suite – Eine Vision wird Realität "Correctness by Construction" und flexible, robuste Ausführung von Unternehmensprozessen

2009-03 *Alena Hallerbach, Thomas Bauer, Manfred Reichert*

Correct Configuration of Process Variants in Provop

2009-04 *Martin Bader*
On Reversal and Transposition Medians

2009-05 *Barbara Weber, Andreas Lanz, Manfred Reichert*
Time Patterns for Process-aware Information Systems: A Pattern-based Analysis

2009-06 *Stefanie Rinderle-Ma, Manfred Reichert*
Adjustment Strategies for Non-Compliant Process Instances

2009-07 *H.A. Kestler, B. Lausen, H. Binder H.-P. Klenk. F. Leisch, M. Schmid*
Statistical Computing 2009 – Abstracts der 41. Arbeitstagung

2009-08 *Ulrich Kreher, Manfred Reichert, Stefanie Rinderle-Ma, Peter Dadam*
Effiziente Repräsentation von Vorlagen- und Instanzdaten in Prozess-Management-Systemen

2009-09 *Dammertz, Holger, Alexander Keller, Hendrik P.A. Lensch*
Progressive Point-Light-Based Global Illumination

2009-10 *Dao Zhou, Christoph Müssel, Ludwig Lausser, Martin Hopfensitz, Michael Kühl, Hans A. Kestler*
Boolean networks for modeling and analysis of gene regulation

2009-11 *J. Hanika, H.P.A. Lensch, A. Keller*
Two-Level Ray Tracing with Recordering for Highly Complex Scenes

2009-12 *Stephan Buchwald, Thomas Bauer, Manfred Reichert*
Durchgängige Modellierung von Geschäftsprozessen durch Einführung eines Abbildungsmodells: Ansätze, Konzepte, Notationen

2010-01 *Hariolf Beth, Frank Raiser, Thom Frühwirth*
A Complete and Terminating Execution Model for Constraint Handling Rules

2010-02 *Ulrich Kreher, Manfred Reichert*
Speichereffiziente Repräsentation instanzspezifischer Änderungen in Prozess-Management-Systemen

2010-03 *Patrick Frey*
Case Study: Engine Control Application

2010-04 *Matthias Lohrmann und Manfred Reichert*
Basic Considerations on Business Process Quality

2010-05 *HA Kestler, H Binder, B Lausen, H-P Klenk, M Schmid, F Leisch (eds):*
Statistical Computing 2010 - Abstracts der 42. Arbeitstagung

2010-06 *Vera Künzle, Barbara Weber, Manfred Reichert*
Object-aware Business Processes: Properties, Requirements, Existing Approaches

Ulmer Informatik-Berichte
ISSN 0939-5091

Herausgeber:
Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
89069 Ulm