Eurographics Symposium on Parallel Graphics and Visualization (2007)
Jean M. Favre, Luis Paulo dos Santos, and Dirk Reiners (Editors)

Interactive Iso-Surface Ray Tracing of Massive Volumetric Data Sets

H. Friedrich[1], I. Wald[2], J. Günther[3], G. Marmitt[1], and P. Slusallek[1]

[1]Computer Graphics Group, Saarland University, Germany
[2]SCI Institute, University of Utah, USA
[3]MPI Informatik, Saarbrücken, Germany

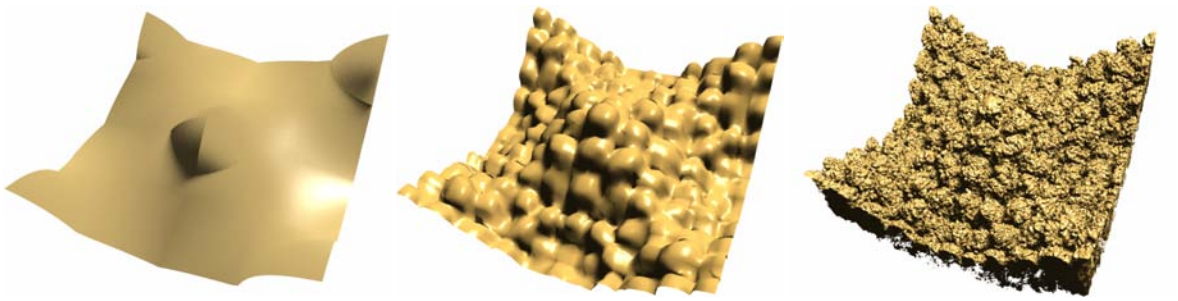

Figure 1: *Three images of the Lawrence-Livermore National Laboratory (LLNL) simulation of a Richtmyer-Meshkov instability (time step 270; iso-value 16) at different level-of-detail (LOD) levels. While data is loaded asynchronously in the background it is possible to fully interact with the scene. A kD-tree based LOD structure is used to bridge loading times while allowing interactive ray tracing of up to four frames per second on a custom PC.*

Abstract

The visualization of iso-surfaces from gridded volume data is an important tool in many scientific applications. Today, it is possible to ray trace high-quality iso-surfaces at interactive frame rates even on commodity PCs. However, current algorithms fail if the data set exceeds a certain size either because they are not designed for out-of-core data sets or the loading times are too high because there is too much overhead involved in the out-of-core (OOC) techniques. We propose a kD-tree based OOC data structure that allows to ray trace iso-surfaces of large volumetric data sets of many giga bytes at interactive frame rates on a single PC. A LOD technique is used to bridge loading times of data that is fetched asynchronously in the background. Using this framework we are able to ray trace iso-surfaces between 2 and 4 fps on a single dual-core Opteron PC at 640×480 *resolution and an in-core memory footprint that is only a fraction of the entire data size.*

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Graphics Systems]: Stand-alone systems; I.3.6 [Methodology and Techniques]: Graphics data structures and data types; I.3.7 [Three-Dimensional Graphics and Realism]: Ray tracing;

1. Introduction and Previous Work

A very important tool for exploring special properties of a volumetric scalar field is the rendering of iso-surfaces. Iso-surfaces show the distribution of a desired iso-value c within the volume data set. They consist of all points that satisfy the trivariate function $f(x,y,z) = c$. Interactive post-processing applications should allow the user to select an arbitrary number of iso-values, alter them on-the-fly, navigate free in the scene (zoom, rotate, pan), and allow to change freely shading and illumination parameters.

Primary acquisition sources for volumetric data sets are (among others) Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) scanners [HJ04] as well as computational simulations like field and fluid simulations

[MCC*99]. The volumetric data sets we are considering in this work consist of single scalar values organized on rectilinear grids.

1.1. Iso-Surface Rendering

In order to visualize an iso-surface it either has to be extracted entirely [LC87, Gib98], e.g. a triangle mesh is built as an approximation to the iso-surface, or the iso-surface is directly rendered without an intermediate geometric representation either by projection [Wes90, MSHC99, CRZP04, NM05], graphics hardware (including special-purpose hardware) [EKE01, RGW*03, RPSC99] or ray tracing techniques [PSL*98a, DPH*03, WFM*05].

Today, the visualization of iso-surfaces is typically the domain of rasterization hardware that renders efficiently extracted triangle meshes. The latest generation of graphics hardware is able to render several million shaded triangles per second and highly optimized systems like InterViews3D [HB04, HPB05] exploiting hardware accelerated *occlusion culling* and *LOD* can efficiently render massive models like a complete Boeing 777 with more than 350 million triangles at realtime frame rates. However, current graphics boards have still some limitations i.e. the limited memory available and the constrained programming model. As a result, complex scenarios including points, surfaces, and volume data plus even advanced shading effects like shadows or reflections are very difficult to render. Additionally, current algorithms are usually limited to rendering a single object of a single type. All of these limitations can be tackled by one or an other way but many individual techniques are necessary leading to a complex system design.

Algorithms that project the voxels onto the image plane, i.e. splatting, are also not appropriate for a flexible out-of-core iso-surface rendering system. Splatting algorithms generate in general *blurry* images due to the usage of blending kernels in image space [HJ04, SM00]. Recent approaches (e.g. [CRZP04]) that can produce high-quality images are not able to deliver interactive frame rates even for smaller data sets due to high computational costs. Additionally, it is at least non-trivial to render scenes with different primitives, e.g. polygonal- *and* iso-surfaces, simultaneously. The same holds also for *object-order* based ray casting approaches like [MJC02, NMHW02] that project cells, or *macro-cells*, onto the image-plane and perform ray marching through the (macro-)cell for all the covered pixels.

Software based ray tracing on the other hand is extremely flexible, has access to the global scene description, and can handle different rendering primitives in a simple plug 'n' play fashion. Additionally, surface-shader effects, once implemented, operate across all primitives because they are (mostly) independent of ray tracing and intersection computations.

1.2. Massive Model Visualization

DeMarle et al. presented in [DPH*03] an interactive out-of-core iso-surface ray tracing engine. They exploit a PC cluster to ray trace iso-surfaces from volume data sets that are too large to fit into the main memory of a single PC. The core rendering engine is basically a port of Parker et al's *-Ray engine [PSL*98b, PPL*99, PMS*99]. The *-Ray ray tracer is considered as one of the first interactive ray tracing systems and runs on a parallel shared memory SGI Onyx 2000 with typically 128 – 256 processors. For gridded volumes, the ray tracer uses a hierarchical min/max grid acceleration structure and a three level bricking approach with a very low memory overhead for the acceleration structure. For a single time-step of the LLNL data set only 8.5 MB of memory was needed for the multi-level grid data structure. Their application structure follows a typical client/server approach. A master system divides the image into rectangular tiles and distributes the tasks to the render nodes on a per tile basis. If a node is a multi-processor system the render task is broken down into sub tasks e.g. scan-lines or image tiles. A `DataServer` on each node is used as an abstraction layer for data management. The `DataServer` is shared between all processors using *semaphores* and *shared memory*. If a ray touches a brick, it must request the brick from the `DataServer`, which loads the data from the network if the data is not present locally. However, in order to achieve interactive rendering performance a cluster setup with 32 render nodes had to be used.

Wald and Dietrich et al. showed in [DWS05, WDS04] how to efficiently ray trace large-scale polygonal models, i.e. a complete Boeing 777, on a scalable shared-memory architecture as well as on commodity PCs. In order to render such massive models with several hundred million triangles on commodity PCs with limited memory, an out-of-core *on-demand* loading scheme was exploited. During each traversal step of their kD-tree acceleration structure a request is sent to a *memory-management unit* (MMU) to check whether the next required kD-tree node is already in main-memory. If not, the request is placed in a priority-queue. A separate loader thread fetches the requested data asynchronously from the hard drive and marks the data as present in the MMU. Two different render modes were proposed. The first exploits so called *proxies* which are pre-rendered approximations of the model, at several LOD levels, that are used until all data is loaded. A proxy is a six-sided box where for each face a color is computed during pre-processing. Wherever, a kD-tree node is not in memory the appropriate color of the proxy is chosen. These proxies are similar to the ones used by Gobbetti et al. in their Far Voxels approach [GM05] that uses these approximations for an efficient visualization of massive models using standard graphics hardware. The disadvantage of these proxy methods is that the geometry as well as the shading and illumination parameters cannot be altered because these information is *captured* in the proxies during pre-processing. Additionally, the

pre-processing of such data is very time consuming. The pre-processing time for a single iso-surface of the LLNL data set takes more than six hours on a 32 CPU cluster and requires approximately 16GB on the hard disc [GM05]. The second proposed method does not rely on any pre-processed LOD structures and just colors pixels red when the next node to be traversed is not in memory. This avoids the pre-processing of the proxies, while on the other hand, no meaningful visualization is possible until most of the visible data has been loaded. Wald exploits the second method for large iso-surface ray tracing [WFM*05].

QSplat [RL00] is a multi-resolution point rendering algorithm that exploits the hierarchical bounding spheres (HBS) data structure for visibility culling, level-of-detail control, and rendering. For rendering a pixel, the HBS are traversed until the size of a bounding sphere is smaller than a predefined threshold or a leaf is reached. Then the data associated with this node are splat onto the screen whereby the size of the splat depends on a defined diameter. For shading a pre-processed quantized *shading normal* is used. An extension [RL01] allows to render large data sets in a streaming fashion over a network. To do so, a bit-mask is used to determine which parts of the HBS are present on the client. If a node, that is not loaded yet, is requested, this request is placed in a queue prioritized by the depth of the node in the hierarchy. A separate loader-thread handles the communication with a data server and is responsible for the actual data loading and bit-mask management. This is similar to Wald et al's. MMU technique [WDS04] for handling out-of-core data sets. Recently, Knoll et al. [KWPH06] showed how to exploit octrees for interactive iso-surface rendering. To do so, the volume data is losslessly compressed using a branch-on-need octree [WV92]. This octree is then also used as acceleration structure for on-the-fly iso-surface ray tracing. In a later extension Knoll et al. [KHW07] added a LOD scheme and packet traversal to increase the ray tracing performance of the octree. Their latest results show that for time slice 270 of the LLNL data set interactive frame rates between 1.1 and 4.7 fps can be achieved on an Intel CoreDuo machine. Additionally, the octree-compressed data set has only a size of approximately 2.5GB.

1.3. Method Overview

In our approach we combine some of the previous mentioned techniques to get a highly flexible and fast rendering system that only requires a single commodity PC to allow for interactive iso-surface ray tracing of large data sets with short loading times. The main goal is to start immediately with the exploration of the data set without the need to wait for all, maybe multiple GB, of data. To do so, we first build a LOD hierarchy of the volume. These LOD data is solely used for rendering while not all data from the finest LOD level is present. Then, for each LOD level we create an implicit min/max kD-tree (Section 2) and merge them together such that we obtain a single kD-tree that is valid for all LOD levels. This kD-tree and the LOD data are then decomposed into *treelets* and saved together in a *page-based* data structure (Section 3.1). During rendering, a separate thread loads all relevant treelets from the hard disc that are required for rendering the desired iso-surfaces in a breadth-first-search (bfs) order (Section 3.2). Whenever a new LOD level is completely loaded, the render threads are notified such that they can use a finer LOD for the next frame.

2. Implicit Min/Max KD-Trees

Kd-trees have proven to be an efficient acceleration structure for ray tracing polygonal scenes [Hav01, Wal04, RSH05] as well as volumetric data [SF90, WFM*05].

In [WFM*05] Wald et al. utilize implicit min/max kD-trees for iso-surface ray tracing. In this context the term *implicit* has two meanings. At first due to the usage of the min/max values entire sub-trees can be culled during ray traversal when they cannot contain the iso-surface. Therefore the min/max kD-tree implicitly represents different kD-trees for ll different iso-values. Second, for rectilinear grid structures the kD-tree nodes in the tree are not stored but are implicitly given by the grid structure.

Using this implicit structure, all information that is actually required for kD-tree traversal can be stored in small tables for each level of the kD-tree i.e. the splitting dimension and position. There is also no need for a leaf indicator since all leaf nodes are stored in the last level of the (complete) kD-tree. All that remains to be stored are the min/max values in the kD-tree nodes. A further significant memory saving can be accomplished if the min/max values at the leaf nodes are not stored, and instead are computed on the fly from the cell's voxels. This is tolerable since these computations have to be performed only at leaf nodes. Although other data tructures like, e.g. , octrees [KWPH06] require less memory we decided to use kD-trees anyway as they are simpler to implement while offering at the same time at least comparable ray tracing performance and still require only a reasonable amount of memory.

3. The Out-of-Core Data Structure

3.1. Building the Out-of-Core Data Structure

The complete pre-processing chain is depicted in Figure 2. In the first step we build a LOD hierarchy of the volume data set. Since we do not know in advance which iso-values are interesting for the user, we use a simple 3D Gauss-filter kernel to scale down the data set and do not try to preserve the topology of iso-surfaces.

Then for each LOD level (including the original volume data) a min/max kD-tree is built. These kD-trees are then merged into a single one such that we have a min/max kD-tree that is valid for rendering all LOD levels. To do so, just

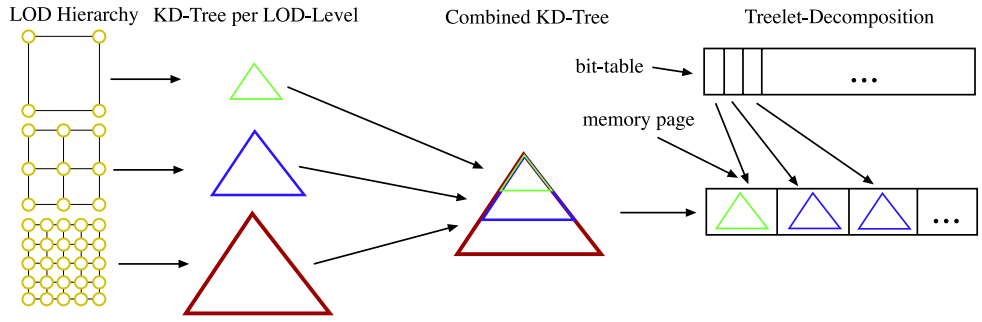

Figure 2: *The pre-process pipeline: At first we create a LOD hierarchy from the volume data. Then, for each LOD data set a min/max kD-tree is built. Afterwards we join all LOD kD-trees into a single one. This kD-tree is finally decomposed into sub-trees of a certain hight and together with the corresponding LOD data of the sub-tree stored in a treelet array on the hard-disc. While rendering, a bit-table keeps track of what treelets are already fetched into main-memory.*

the min/max intervals of the nodes have to be adjusted by joining the corresponding min/max intervals. This merging is necessary because the LOD filter shifts the range of values and thus the kD-tree of the original data may not be valid for all LOD volumes.

Once we have the merged min/max kD-tree, we decompose this tree and the LOD volume data into treelets (see Figure 3). A treelet consists of a sub-tree of the min/max kD-tree with a fixed height N (e.g. 63 nodes for $N = 6$), a corresponding block of LOD voxels (similar to [BPTZ99]), or voxels from the original data set at the last treelet level, an ID that identifies the treelet, and the number of voxels in each dimension (see Figure 3). All sub-trees have the same height except maybe the top-most sub-tree. Note that the number of LOD levels correlates with the height of the sub-trees. If we have e.g. four levels of treelets, then we have also four LOD levels.

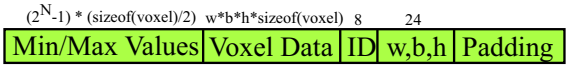

Figure 3: *The structure and memory requirements of a treelet. Min/max values, voxel data, an ID, and the dimensions of the voxel data are grouped together. If the size of a treelet is small enough multiple treelet structures can be placed in one page on the hard disc. Width w, breadth b, and height h of the voxel block include an outer layer of voxels for calculating central differences for shading.*

These treelets are saved in a page-based data structure that allows fast loading from the hard disc. Page-based means that treelets will be stored with the size of a *memory-page* on the hard disc. The size of the page is given by the operating system and is in our case 4k bytes. This page based treelet approach is similar to the blocklets approach by Bajaj et al. [BPTZ99] and Zhang et al. [ZN03]. If the treelets do not have the size of a page, the data will be padded to page size, or the next multiple of page size if the treelet is larger than one page. If the size of a treelet is small enough multiple treelets can be placed in one page. All treelets of a particular LOD level are stored consecutively in breadth-first-search order.

In order to allow a proper shading, not only the voxel data of the corresponding voxel region are placed in a treelet but also an outer layer of voxels such that the central difference can be calculated for estimating gradients. To decrease the memory requirement further the min/max values are quantized to half of the original bit resolution. The additional traversal operations that are caused by a reduced efficiency of the sub-tree culling decreases the overall rendering performance typically in the range of five to ten percent.

3.2. Traversal and Treelet Loading

The basic kD-tree traversal scheme is the same as in Wald et al's. [WFM*05] approach (see [WFM*05, Wal04] for a thorough discussion).

3.2.1. Traversal

However, every traversal step and every access to voxel data at the leaf nodes requires a MMU request (see Section 1.2) in Wald's approach which slows down the overall rendering performance and loading time until all relevant data is fetched from the hard disc. With our new approach only one MMU request per treelet is required. Figure 4 sketches the traversal algorithm.

Due to the fact that all needed data for treelet traversal, intersection tests etc. are stored in one memory page cache-aliasing can be reduced. Another advantage is that we know the height of the sub-tree in a treelet. This fact can be used to remove a conditional (leaf-node check) in the innermost traversal loop and thus reduces the traversal costs.

```
Treelet = Load First Treelet
Start Loader Thread
while(1)
{
  Traverse kD-Tree Sub-Tree in Treelet

  if(CurrentTreeletLevel < LoadedLevels)
  {
    Treelet = Calculate Next TreeletAddr
    Continue
  }

  hit = Intersect Treelet
  if (hit) Shade Pixel
  Return
}
```

Figure 4: *Traversal pseudo-code: The first treelet is loaded from the hard disc and one loader thread is started before rendering. Afterwards, for each ray the traversal starts with the first treelet and traverses as much LOD levels as have been already loaded. The loader thread updates the maximum traversal depth for the render threads whenever the necessary treelets of a new LOD level are loaded. After traversal the usual ray iso-surface intersection and shading takes place with the current treelet data.*

3.2.2. Treelet Loading

For loading the required data from the hard disc we exploit a similar memory-management-unit (MMU) technique as Dietrich and Wald et al. [DWS05, WDS04, WFM*05].

The MMU has to perform two tasks: The first task is to load all treelets that are required for rendering a desired iso-surface without *blocking reads* and second to notify the render threads whenever all necessary treelets of a new LOD level are loaded.

To do so, the MMU creates a loader thread that manually fetches the data required for the current iso-value so that the loading process does not stall the render threads. The loader thread sweeps over the treelets, which are stored in breadth-first-search manner, and checks the min/max intervals of the leaf nodes of the kD-tree sub-trees. If the iso-value is within the min/max interval both children will be loaded and afterwards marked as present in a bit-table. For every memory-page this bit-table has an entry that is covered by the `mmap`ed (memory mapped) file with treelets. Before the sweep process starts the first treelet is loaded separately to assure that it is available. The bit list is necessary because we have to know in LOD level $+1$ which treelets have been loaded in the previous level and thus do not touch unneeded data on the hard disc.

4. Results

In order to evaluate the efficiency of our out-of-core data structure we measured performance numbers for two data sets: Time-step 270 of the LLNL Richtmyer-Meshkov instability with a voxel resolution of $2048^2 \times 1920$ as well as a synthetic data set Attractor (see Figure 5).

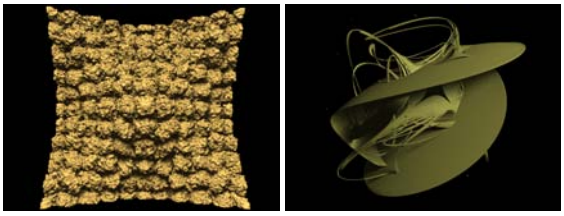

Figure 5: *Example images of our test scenes: LLNL and Attractor rendered with phong-shading, progressive soft-shadows and a single point-light source between 1.7 and 1.9 fps at* 640×480 *image resolution using two CPU cores.*

The Attractor volume consists of three different 3D-attractors and a voxel resolution of 2048^3. Both data sets use 8-bit data values. The test system is a dual dual-core Opteron 275 (2.2GHz) PC with 8GB main memory. Image resolution is always 640×480 pixel and for shading a simple diffuse surface-shader is used (see Table 2).

4.1. Treelet-Size Influence

As described in Section 3.1 we decompose the LOD data and min/max kD-tree into a treelet hierarchy. The number of treelet levels influences the required disc space, in-core RAM for a particular iso-surface, loading time, and rendering performance. Table 1 shows some results when we increase the height of the treelets, and thus reduce the number of levels in the treelet hierarchy. The pre-processing time in Table 1 includes loading/writing the data from/to hard disc, LOD creation, min/max kD-tree building, and merging. In our experiments we found that a treelet height of nine yields the best overall performance numbers. With larger treelets, some numbers i.e. the required in-core RAM increase again.

Treelet Height	Pre-process Time (h)	Disc Space (GB)	Working Set (GB)	Loading Time (Min)	FPS
3	17	292	16.0	58	1.0
6	3	66	10.0	30	1.8
9	1	33	6.1	5	2.5

Table 1: *Performance numbers for the LLNL data set with increasing treelet heights. If we increase the treelet height, the pre-processing time, the required in-core memory for rendering a particular iso-surface, and the loading time of the relevant data decrease significantly. At the same time the rendering performance almost doubles.*

4.2. Overall Rendering Performance

The overall rendering performance is presented in Table 2 (see Figures 5, 6). As the fps numbers show we always achieve interactive rendering performance. These numbers indicate the fps for the finest level in our hierarchy. The rendering of coarser LOD levels is faster and drops down with every finer LOD level. Furthermore, the required in-core memory is reasonable, especially if we consider that we use quantized min/max values in the treelets. This means that we not only load the treelets for a single iso-surface but all treelets for a quantized "bucket".

Scene	Iso	Loading Time (Min)	Working Set (GB)	FPS 2 Cores	FPS 4 Cores
LLNL (zoom)	16	5	6.1	2.0	3.9
LLNL (overview)	16	5	6.1	2.6	5.1
Attr (zoom)	25	4	2.1	1.5	2.9
Attr (overview)	25	4	2.1	3.6	6.9

Table 2: *Overall rendering performance for our two test scenes (Figures 5, 6) measured with a simple diffuse shading using two and four CPU cores. As the results show, independent from the viewpoint at least interactive frame rates can be achieved. The loading time is the time until all relevant data from the finest level are loaded.*

4.3. Comparison

In comparison to the approach of DeMarle et al. [DPH*03] we achieve almost the same rendering performance using only a single PC rather then a 32 PC cluster setup (although we obviously use a newer and correspondingly more powerful multi-core CPU that the CPUs used in DeMarle's cluster). This reduces the hardware requirements significantly.

In addition, we achieve very fast loading times due to the linear memory-page loading from hard disc. Wald et al's [WFM*05] out-of-core method requires for the LLNL data set approximately 18 minutes until all relevant data are loaded (see Section 1.2). However, only the visible data are loaded and if the camera position changes loading starts again. This is in contrast to our method were we load all relevant data for an iso-surface independent of visibility. Furthermore, our in-core footprint is smaller: 6.1GB in our system compared to 8.0GB in Wald's approach (for all views).

An additional plus is the faster kD-tree traversal. Due to our simple out-of-core scheme we are able to achieve approximately twice the rendering performance of Wald's approach.

Compared to the approach of Knoll et al. [KWPH06, KHW07] we have a higher rendering performance but on the other side require more memory due to the usage of a kD-tree rather than octree. Nevertheless, both approaches are orthogonal to each other. Both approaches could be combined into a single system either by adding their compression scheme and LOD usage to our system or by extending their approach with the out-of-core approach.

5. Conclusions and Future Work

We have presented a fast out-of-core iso-surface rendering system that is able to render giga-byte data sets at interactive frame rates on commodity PCs. As the results show we outperform the current state-of-the-art iso-surface ray tracing systems in terms of higher rendering performance and shorter loading times. Additionally, the out-of-core scheme allows for immediate starting the exploration of the data set and the in-core memory requirements are smaller than the original data set size. Finally the hardware requirements are still reasonable although we need considerably more hard-disc memory compared to [KWPH06].

To further improve the usability of our system, we will extend our approach in various ways. Currently we only use a single-ray traversal and analyze different traversal strategies for kD-trees to support larger ray bundles similar to [WBS06, LYTM06] for BVH's. Our focus in that direction is to support secondary rays as cheap as possible i.e. if rays have no common origin. A first attempt with the algorithm of Reshetov et al. [RSH05] was not successful since rendering performance degenerated too fast with an increasing number of secondary rays due to their ray-frustum approach.

In order to reduce the in-core and hard-disc memory footprint of our approach we want to exploit lossless multi-resolution compression schemes e.g. wavelets, and additionally avoid the storage of min/max values in each kD-tree level. Furthermore, we want to incorporate the time-domain for such massive models.

References

[BPTZ99] BAJAJ C. L., PASCUCCI V., THOMPSON D., ZHANG X. Y.: Parallel accelerated isocontouring for out-of-core visualization. In *PVGS '99: Proceedings of the 1999 IEEE symposium on Parallel visualization and graphics* (New York, NY, USA, 1999), ACM Press, pp. 97–104. 4

[CRZP04] CHEN W., REN L., ZWICKER M., PFISTER H.: Hardware-Accelerated Adaptive EWA Volume Splatting. In *VIS '04: Proceedings of the conference on Visualization '04* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 67–74. 2

[DPH*03] DEMARLE D. E., PARKER S., HARTNER M., GRIBBLE C., HANSEN C.: Distributed Interactive Ray Tracing for Large Volume Visualization. In *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG)* (2003), pp. 87–94. 2, 6

[DWS05] DIETRICH A., WALD I., SLUSALLEK P.: Large-Scale CAD Model Visualization on a Scalable

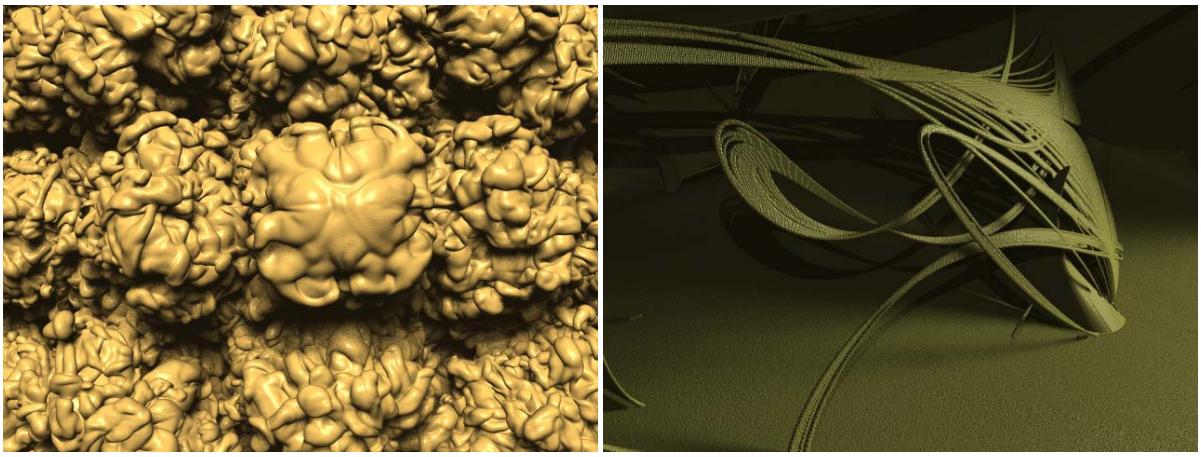

Figure 6: *Converged high quality renderings of our test scenes. Left: A closeup on the LLNL surface. Right: A zoom in the Attractor model showing some fine details. Both models are rendered with 1.0 to 1.6 fps at* 640×480 *image resolution, phong-shading and progressive soft-shadows using two CPU cores.*

Shared-Memory Architecture. In *Proceedings of 10th International Fall Workshop - Vision, Modeling, and Visualization (VMV) 2005* (Erlangen, Germany, November 2005), Greiner G., Hornegger J., Niemann H., Stamminger M., (Eds.), Akademische Verlagsgesellschaft Aka, pp. 303–310. 2, 4

[EKE01] ENGEL K., KRAUS M., ERTL T.: High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (2001), pp. 9–16. 2

[Gib98] GIBSON S. F. F.: Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. In *MICCAI '98: Proceedings of the First International Conference on Medical Image Computing and Computer-Assisted Intervention* (London, UK, 1998), Springer-Verlag, pp. 888–898. 2

[GM05] GOBBETTI E., MARTON F.: Far Voxels: A Multiresolution Framework for Interactive Rendering of Huge Complex 3D Models on Commodity Graphics Platforms. In *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)* (2005), vol. 24, pp. 878–885. 2

[Hav01] HAVRAN V.: *Heuristic Ray Shooting Algorithms*. PhD thesis, Faculty of Electrical Engineering, Czech Technical University in Prague, 2001. 3

[HB04] HEYER M., BRÜDERLIN B.: Hardwareunterstütztes Occlusion Culling zur Visualisierung sehr grosser Modelle. In *3. Paderborner Workshop Augmented Reality / Virtual Reality in der Produktentstehung* (Paderborn, June 2004), HNI, pp. 39–49. 2

[HJ04] HANSEN C., JOHNSON C. R.: *The Visualization Handbook*. Elsevier, 2004. 1, 2

[HPB05] HEYER M., PFÜTZER S., BRÜDERLIN B.: Visualization Server for Very Large Virual Reality Scenes. In *4. Paderborner Workshop Augmented Reality / Virtual Reality in der Produktentstehung* (Paderborn, June 2005), HNI. 2

[KHW07] KNOLL A., HANSEN C., WALD I.: *Coherent Multiresolution Isosurface Ray Tracing*. Tech. Rep. UUSCI-2007-001, University of Utah, School of Computing, 2007. 3, 6

[KWPH06] KNOLL A., WALD I., PARKER S. G., HANSEN C. D.: Interactive Isosurface Ray Tracing of Large Octree Volumes. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing* (2006). 3, 6

[LC87] LORENSEN W. E., CLINE H. E.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (Proceedings of ACM SIGGRAPH) 21*, 4 (1987), 163–169. 2

[LYTM06] LAUTERBACH C., YOON S.-E., TUFT D., MANOCHA D.: RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs. *Technical Report, University of North Carolina at Chapel Hill* (2006). 6

[MCC*99] MIRIN A. A., COHEN R. H., CURTIS B. C., DANNEVIK W. P., DIMITS A. M., DUCHAINEAU M. A., ELIASON D. E., SCHIKORE D. R., ANDERSON S. E., PORTER D. H., WOODWARD P. R., SHIEH L. J., WHITE S. W.: Very high resolution simulation of compressible turbulence on the ibm-sp system. In *Supercomputing '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing* (New York, NY, USA, 1999), ACM Press, p. 70. 1

[MJC02] MORA B., JESSEL J. P., CAUBET R.: A new object-order ray-casting algorithm. In *VIS '02: Proceed-*

ings of the conference on Visualization '02 (Washington, DC, USA, 2002), IEEE Computer Society, pp. 203–210. 2

[MSHC99] MUELLER K., SHAREEF N., HUANG J., CRAWFIS R.: High-quality splatting on rectilinear grids with efficient culling of occluded voxels. *IEEE Transactions on Visualization and Computer Graphics 5*, 2 (1999), 116–134. 2

[NM05] NEOPHYTOU N., MUELLER K.: GPU Accelerated Image Aligned Splatting. In *Volume Graphics 2005* (Washington, DC, USA, 2005), pp. 197–205. 2

[NMHW02] NEUBAUER A., MROZ L., HAUSER H., WEGENKITTL R.: Cell-based first-hit ray casting. In *Proceedings of the Symposium on Data Visualisation 2002* (2002), pp. 77–ff. 2

[PMS*99] PARKER S., MARTIN W., SLOAN P.-P., SHIRLEY P., SMITS B., HANSEN C.: Interactive Ray Tracing. In *Proceedings of Interactive 3D Graphics* (1999), pp. 119–126. 2

[PPL*99] PARKER S., PARKER M., LIVNAT Y., SLOAN P.-P., HANSEN C., SHIRLEY P.: Interactive Ray Tracing for Volume Visualization. *IEEE Transactions on Computer Graphics and Visualization 5*, 3 (1999), 238–250. 2

[PSL*98a] PARKER S., SHIRLEY P., LIVNAT Y., HANSEN C., SLOAN P.-P.: Interactive Ray Tracing for Isosurface Rendering. In *IEEE Visualization '98* (1998), Ebert D., Hagen H., Rushmeier H., (Eds.), IEEE, pp. 233–238. 2

[PSL*98b] PARKER S., SHIRLEY P., LIVNAT Y., HANSEN C., SLOAN P.-P.: Interactive Ray Tracing for Isosurface Rendering. In *IEEE Visualization* (October 1998), pp. 233–238. 2

[RGW*03] ROETTGER S., GUTHE S., WEISKOPF D., ERTL T., STRASSER W.: Smart Hardware-Accelerated Volume Rendering. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 231–238. 2

[RL00] RUSINKIEWICZ S., LEVOY M.: QSplat: A multiresolution point rendering system for large meshes. In *Siggraph 2000, Computer Graphics Proceedings* (2000), Akeley K., (Ed.), ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 343–352. 3

[RL01] RUSINKIEWICZ S., LEVOY M.: Streaming qsplat: a viewer for networked visualization of large, dense models. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics* (New York, NY, USA, 2001), ACM Press, pp. 63–68. 3

[RPSC99] RAY H., PFISTER H., SILVER D., COOK T. A.: Ray casting architectures for volume visualization. *IEEE Transactions on Visualization and Computer Graphics 5*, 3 (1999), 210–223. 2

[RSH05] RESHETOV A., SOUPIKOV A., HURLEY J.: Multi-Level Ray Tracing Algorithm. *ACM Transaction of Graphics 24*, 3 (2005), 1176–1185. (Proceedings of ACM SIGGRAPH). 3, 6

[SF90] SUBRAMANIAN K. R., FUSSEL D. S.: *A Search Structure based on K-d Trees for Efficient Ray Tracing*. Tech. Rep. PhD Dissertation, Tx 78712-1188, The University of Texas at Austin, Dec. 1990. 3

[SM00] SCHUMANN H., MUELLER W.: *Visualisierung - Grundlagen und allgemeine Methoden*. Springer, 2000. 2

[Wal04] WALD I.: *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics Group, Saarland University, 2004. 3, 4

[WBS06] WALD I., BOULOS S., SHIRLEY P.: Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *Technical Report, SCI Institute, University of Utah, No UUSCI-2005-014 (conditionally accepted at ACM Transactions on Graphics)* (2006). 6

[WDS04] WALD I., DIETRICH A., SLUSALLEK P.: An Interactive Out-of-Core Rendering Framework for Visualizing Massively Complex Models. In *Rendering Techniques 2004, Proceedings of the Eurographics Symposium on Rendering* (2004), pp. 81–92. 2, 3, 4

[Wes90] WESTOVER L.: Footprint evaluation for volume rendering. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1990), ACM Press, pp. 367–376. 2

[WFM*05] WALD I., FRIEDRICH H., MARMITT G., SLUSALLEK P., SEIDEL H.-P.: Faster Isosurface Ray Tracing using Implicit KD-Trees. *IEEE Transactions on Visualization and Computer Graphics 11*, 5 (2005), 562–573. 2, 3, 4, 6

[WV92] WILHELMS J., VAN GELDER A.: Octrees for faster isosurface generation. *ACM Transactions on Graphics 11*, 3 (July 1992), 201–227. 3

[ZN03] ZHANG H., NEWMAN T. S.: Efficient parallel out-of-core isosurface extraction. In *PVG '03: Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (Washington, DC, USA, 2003), IEEE Computer Society, p. 3. 4