Technical Strategies for Massive Model Visualization

Enrico Gobbetti
CRS4 *

Dave Kasik
Boeing †

Sung-eui Yoon
KAIST ‡

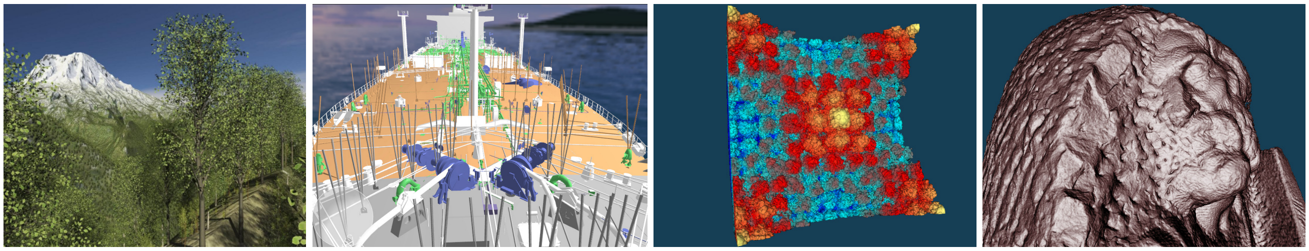

Figure 1: **Examples of massive models.** Landscape Model consisting of more than one billion polygons (courtesy Saarland University); Double Eagle Oil Tanker CAD model consisting of about 82 million triangles and taking more than 3GB data (courtesy University of North Carolina at Chapel Hill); Richtmyer-Meshkov isosurface consisting of about 472M triangles (courtesy CRS4 and Lawrence Livermore National Labs); St. Matthew 0.25mm laser scanning model consisting of about 372M triangles (courtesy CRS4 and Digital Michelangelo Project)

Abstract

Interactive visualization of massive models still remains a challenging problem. This is mainly due to a combination of ever increasing model complexity with the current hardware design trend that leads to a widening gap between slow data access speed and fast data processing speed. We argue that developing efficient data access and data management techniques is key in solving the problem of interactive visualization of massive models. Particularly, we discuss visibility culling, simplification, cache-coherent layouts, and data compression techniques as efficient data management techniques that enable interactive visualization of massive models.

1 Introduction

Over the last several decades, there have been significant advances in model acquisition, computer-aided design (CAD), and simulation technologies. These technologies have resulted in massive data sets of complex geometric models. These data include complex CAD environments, natural landscape models, scanned urban data, and various scientific simulation data. These massive data typically require giga byte size and even tera byte size. Some of these models are shown in Fig. 1.

Handling such massive models presents important challenges to software and system developers. This is particularly true for highly interactive 3D applications, such as visual simulations and virtual environments, with their inherent focus on interactive, low-latency, and real-time processing.

In the last decade, the graphics community has witnessed tremendous improvements in the performance and capabilities of computing and graphics hardware. The natural question becomes why such a performance boost has not transformed rendering performance problems into memories of the past. A single standard dual-core 3 GHz Opteron processor has roughly 20 GFlops, a Playstation 3's CELL processor has 180 GFlops, and recent GPUs, now fully programmable, provide around 340 GFlops. Hardware parallelism, e.g., in the form of multi-core CPUs or multi-pipe GPUs, results in the performance improvement, which tends to follow and even outpace Gordon Moore's exponential growth prediction. Such a rate increase seems to be continuing. For instance, Intel has already announced an 80 core processor capable of TeraFlop performance. Despite the continued increase in computing and graphics processing power, it is clear to the graphics community that one cannot just rely on hardware improvement to cope with arbitrarily large data sizes for the foreseeable future. This is not only because the increased computing power also allows users to produce more and more complex datasets, but also because memory bandwidth and data access speed grow at significantly slower rates than processing power and become the major bottlenecks when dealing with massive datasets. Typically, the access times of different levels of a memory hierarchy vary by orders of magnitude (e.g., 10^{-8} s for L1/L2 caches, 10^{-7} s for main memory, and 10^{-2} s for disks). Moreover, CPU performance has increased 60% per year for nearly two decades. On the other hand, the main memory and disk access time only decreased by 7–10% per year during the same period [Ruemmler and Wilkes 1994; Patterson et al. 1997]. This relative performance gap between CPU performance and access time to DRAM is shown in Fig. 2. As a result, a major computational bottleneck is usually in data access rather than computation, and we expect that this trend will continue in the near future.

As a result, massive datasets cannot be interactively rendered by brute force methods. The real challenge is in designing rendering systems that capture as much of this performance growth as possible. When dealing with massive models, achieving high performance requires methods for carefully managing bandwidth requirements, controlling working set size, and ensuring coherent access

*Visual Computing Group, CRS4, Sardegna Ricerche Ed. 1, 09010 Pula, Italy – `http://www.crs4.it/vic/` – `gobbetti@crs4.it`

†The Boeing Company, Seattle, WA – `http://www.boeing.com/` – `david.j.kasik@boeing.com`

‡Dept. of Computer Science Korea Advanced Institute of Science and Technology (KAIST) 373-1 Guseong-dong, Yuseong-gu, Daejeon, 305-701, South Korea – `http://jupiter.kaist.ac.kr/s̃ungeui/` – `sungeui@gmail.com`

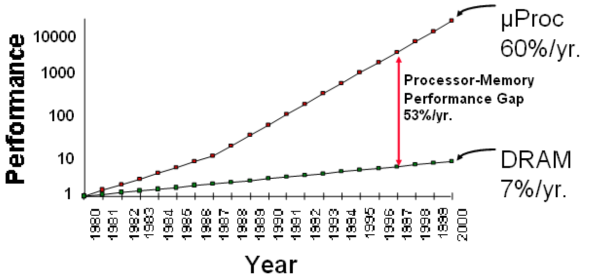

Figure 2: **Memory Wall.** The CPU performance has increased 60% per year for almost two decades. On the other hand, the access time for main memory consisting of DRAM only decreased by 7-10% per year during the same period. The graph shown is excerpted from a talk slide of Trishul Chilimbi.

patterns. Furthermore, given current multi-core CPUs and GPUs, solutions which can be formulated in a (data) parallel fashion will be able to continually take advantage of improved hardware performance.

In this paper, we provide a short overview of the technology behind massive model rendering, with a special emphasis on the following aspects:

Data reduction techniques. All systems dealing with massive models require the integration of techniques for reducing the working set by filtering out as efficiently as possible the data that is not contributing to a particular image. This goal is achieved by employing appropriate data structures and algorithms for visibility or detail culling, as well as by choosing alternate graphics primitive representations (Section 2).

Cache-coherent layout methods. Data access time increase dramatically if various cache does not have the requested data by the rendering method. In order to reduce the number of cache misses and, thus, lower the data access time, cache-coherent layouts have been proposed. This technique re-organizes the underlying data element to reduce the number of cache misses while accessing data (Section 3).

Data compression. Data access time can also be reduced by lowering data size requirements with compression techniques. Rendering methods require random access to the data, we can also design data compression and decompression methods that support random access on the compressed data structures without decompressing the whole data set (Section 4).

2 Reducing the rendering working set

Current graphics hardware cannot render massive models, consisting of hundreds of millions of polygons at interactive frame rates. This requires techniques to reduce the complexity of scenes on a frame by frame basis. The main techniques used to achieve this goal consists of: **(a)** rendering only the polygons that are determined visible (*Visibility Culling*); **(b)** using geometric approximations of the original model with lower polygon count (*Levels of Detail*); **(c)** using alternate representations in place of polygons (*Sample-based representations* and *Higher-order primitives*). By combining these techniques together in an adaptive system, it is possible to create a renderer whose runtime and memory footprint is proportional to the generated output complexity, not to the total model complexity.

2.1 Visibility culling

Determining which surface is visible for each pixel is the at core of rendering applications. Visible surface determination techniques are essentially methods for solving a sorting problem, i.e., determining which parts of the model are closer to the viewer. The many proposed solutions vary in the order in which the sort is performed and how the problem is subdivided to make it more tractable. At the broad level, practically only two classes of algorithms are used today when dealing with massive models: **(a)** rasterization with z-buffering, an object order approach that determines the visible surface by streaming through scene polygons, rasterizing them, and projecting them to screen while maintaining the depth of each pixel, and **(b)** ray tracing, an image space approach that determines visible surfaces by computing ray intersections for each pixel.

From the memory management point-of-view, rasterization offers in principle more code and data cache coherency, because switching primitives and rendering attributes occurs much less frequently and most operations work object-by-object basis on data residing in local memory. This explains the success of rasterization hardware for supporting real-time rendering of small scenes. The situation is more complex for massive models. In their most basic form, both rasterization and raytracing techniques are limited to linear time complexity in the number of scene primitives. In order to enable rendering in sub-linear time complexity, spatial index structures and visibility culling techniques must be applied to limit the number of polygons being sent to the graphics pipeline and/or checked for ray intersection. Even though the ray tracing and rasterization fields have independently developed their own approaches in the past, the underlying issues faced when dealing with massive models are somewhat similar, and state-of-the-art systems are converging towards applying similar solutions.

2.1.1 Object-space subdivision

Visibility culling methods are typically realized with the help of a so-called *spatial index*, a spatial data structure that organizes the geometric primitives in the 3D space. There are two major approaches, *spatial partitioning* and *bounding volume hierarchies* (BVHs). Bounding volume hierarchies conceptually organize geometric primitives in a bottom-up manner by hierarchically grouping bounding volumes of objects, while spatial partitioning schemes subdivide the scene in a top-down manner into a hierarchy of disjoint cells that contains the entire scene. Quite a number of spatial partitioning schemes have been proposed in the past. *Hierarchical grids*, *octrees*, *kd-trees* are the most popular methods, and *kd-trees* are usually considered the option of choice for massive models. More details can, e.g., be found in [Samet 2006]. Even though the concepts of classic spatial indexes are simple and well understood, constructing them for massive models requires specialized methods that have a low computational complexity and coherent access patterns to avoid I/O thrashing, while at the same time providing optimized space partitions for visibility queries.

In the case of kd-trees, a de-facto standard for obtaining optimized subdivision is to minimize the cost model for ray-object intersections called *Surface Area Heuristics* (SAH) [Goldsmith and Salmon 1987; MacDonald and Booth 1990; Havran 2000]. This heuristics assumes a uniform distribution of rays with no occlusion, and makes it simple to estimate the probability to traverse the different branches of the hierarchy by comparing the surface areas of the bounding boxes of the various nodes. An approximately optimal kd-tree can be computed by minimizing the expected ray tracing cost by performing a top-down greedy optimization. Such a technique, which recursively splits the model by choosing the minimum cost split plane at each step, is impractical for massive models. This is because there are too many possible splitting planes and finding the best split plane requires to sort triangles according to these planes. For these reasons, many authors have proposed simplified techniques for faster tree construction (e.g., [Popov et al. 2006; Hunt et al. 2006; Shevtsov et al. 2007]). These methods share a common set of concepts. First of all, they only consider axis aligned bounding boxes of objects instead of individual triangles for building the hierarchy. Second, they do not test all potential split planes, but only use K heuristically selected equidistantly spaced planes. Third, the SAH for each of these is computed in a single streaming pass. This approach greatly reduces the number of plane evaluations (K bin planes instead of $O(N)$ triangle bounds planes) but also avoids any sorting. Splitting can thus be done simply with two $O(N)$ passes. The full hierarchy can thus be constructed with $O(N \log N)$ operations. Moreover, and most importantly for massive models, all operations are performed in a streaming fashion, with good memory locality and minimal needs for in-core memory. The main drawback of these methods is the need to select up-front a small set of candidate planes. A more elaborate solution, which avoids binning and considers triangle splitting is presented in [Wald and Havran 18-20].

Bounding volume hierarchies are not generally used for large static environments but for (smaller) dynamic environments for which the hierarchy is either given up-front at modeling time, e.g., by associating bounding volumes to objects in a kinematic hierarchy, or recomputed dynamically as objects move. The research focus more on how to update a hierarchy after object motion. Reasonably fast $O(N \log N)$ algorithms for rebuilding BVHs are presented in [Wald et al. 2007; Lauterbach et al. 2006; Wald 2007], while $O(N)$ methods for refitting an already existing bounding volume hierarchy are presented in [Havran et al. 2006; Woop et al. 2006; Wächter and Keller 2006]. Hybrid methods combining refitting and construction techniques are also available [Lauterbach et al. 2006; Yoon et al. 2007].

2.1.2 From-point visibility culling

From-point algorithms are the basis of all interactive viewing applications, since they attempt to determine at each frame which parts of the scene are visible from the current viewpoint. Visible surface determination (or hidden-surface removal) is the most basic from-point algorithm and can be considered as the final stage of every rendering pipe-line. Before actual visible surface determinations takes place, visibility culling methods must be employed to filter out data at the coarse level. View-frustum and backface culling are simple but effective from-point operations that can be optimized using spatial data structures. A common choice is to combine a hierarchy of bounding spheres, axis-aligned bounding boxes, or kd-trees with cones of normals [Rusinkiewicz and Levoy 2000a; Cignoni et al. 2004; Gobbetti and Marton 2004], and to perform hierarchical view-frustum and backface culling during a top-down scene traversal. Processing stops whenever a model portion is proved out-of-view or backfacing. Such traversal schemes can also be adapted for implementing occlusion culling, which stops traversal in case of occlusion.

In the case of ray-tracing, early traversal termination in case of occlusion is simple to implement by a front-to-back traversal of the spatial index: since visibility is evaluated independently for each ray, once a hit-point has been found, it is certain that geometric primitives behind are not visible for that specific ray direction. An important optimization that is widely applied in state-of-the-art real-time ray tracing systems is to simultaneously trace bundles of rays called *packets* [Wald et al. 2001]. First, working on packets allows use of SIMD vector operations of modern CPUs to perform parallel traversal and intersection of multiple rays. Second, packets enable deferred shading, i.e., it is not necessary to switch between intersection and shading routines for every single ray, thus amortizing memory accesses, function calls, etc. Third, it is possible to avoid traversal steps and intersection calculations based on the bounds of ray packets and makes better use of both object and scanline coherence. This idea of accelerating raytracing by working on groups of rays is also exploited in frustum traversal methods [Reshetov et al. 2005].

When using rasterization, the decision about when traversal of the spatial index can be stopped can also be made in image space by exploiting the Z-buffer, and the most recent algorithms exploit graphics hardware for this purpose. For occlusion culling during front-to-back scene traversal, bounding volumes are simply tested for visibility against the current Z-buffer using the *occlusion query* functionality to determine whether to continue traversal. It should be noted that, although the query itself is processed quickly using the rasterization power of the GPU, its result is not available immediately due to the delay between issuing the query and its actual processing by the graphics pipeline. A naive application of occlusion queries can actually decrease the overall application performance due the associated CPU stalls and GPU starvation and introduce additional end-to-end latency in the application. For this reason, modern methods exploit spatial and temporal coherence to schedule the issuing of queries [Govindaraju et al. 2003; Bittner et al. 2004; Yoon et al. 2004; Heyer et al. 2005; Klosowski and Silva 2001]. The central idea of these methods is to issue multiple queries for independent scene parts and to avoid repeated visibility tests of interior nodes by exploiting the coherence of visibility classification. It is interesting to note that hierarchical front-to-back rasterization in combination with occlusion culling can be interpreted as a form of beam- or frustum tracing, which demonstrates the convergence of ray-tracing and rasterization research in the massive model rendering domain.

2.1.3 From-region visibility culling

From-point algorithms perform visibility culling at each frame by exploiting object space-subdivision. For static scenes, it is tempting to pre-compute visibility information for scene regions, to speed up up run-time visibility processing. This is the domain of *from-region* algorithms, which precompute a *potentially visible set* (PVS) for cells of a fixed subdivision of the scene partitioning the view-space. During rendering, only the primitives in the PVS of the cell where the observer is currently located, are rendered, potentially leading to large savings in rendering time. For complex scenes, however, these savings are more theoretical than practically achievable with current technology. While exact visibility from a single viewpoint can be calculated using visible surface determination methods and accelerated using space partitioning structures, computing the PVS for a region is much harder. Excellent algorithms for computing exact visibility from a region in space exist for general scenes do exist [Durand 1999; Duguet and Drettakis 2002; Nirenstein et al. 2002; Bittner 2002; Haumont et al. 2005; Mora et al. 2005]. However, their running time and memory costs make them unsuitable for massive models. For this reason, many authors have concentrated on “conservative” techniques, i.e., techniques that simplify computation by, hopefully slightly, over-estimating the PVS to include some objects that are actually not visible and to never exclude unoccluded objects. In reality, the problem turns out to also be very hard to solve, and there are practically no published provably conservative techniques for general environments. Rather, known techniques are restricted to particular types of scenes. Examples of constraints are the limitations to architectural building interiors [Airey et al. 1990; Teller and Sequin 1991], 2.5D visibility for terrains and urban scenes [Wonka et al. 2000; Bittner et al. 2001; Koltun et al. 2001], volumetric occluders [Schaufler et al. 2000] or large occluders close to the view cell [Durand et al. 2000; Andújar et al. 2000; Leyvand et al. 2003]. For general scenes, non conservative sampling based solutions, that compute from-region solutions combining results from from-point queries, have recently emerged as a practical approach, due to their robustness and ease of implementation. Nirenstein and Blake [Nirenstein and Blake 2004] proposed an approach which uses rasterization hardware for sampling visibility. An approach that focuses, instead on harnessing a fast ray-tracing kernel has been recently presented by Wonka et al. [Wonka et al. 2006].

Storing and transmitting the computed PVS is also an important problem. There is an obvious trade-off between the quality of the PVS estimation on one hand and memory consumption and precomputation time on the other hand. Smaller view cells improve the quality of PVS computation, simultaneously increasing the number of view cells that need to be precomputed. In addition to requiring large precomputation times, having a large number of view cells can result in extremely large storage requirements for storing all PVSs, as well as in large bandwidth requirements for communicating the PVSs to the rendering engine, which is an important drawback for massive scenes.

In general, from-point techniques are more robust and easy to integrate in a system, since they require less storage and less preprocessing time and resources. There are, however, situations in which a from-region algorithm is appropriate and can provide considerable advantages: this is the case, for instance, of a number of videogames, in which the scene is modeled only once and can be constructed to make region selection easy. A good from-region algorithm for general or massive models with reasonable preprocessing cost and good storage optimization remains an open issue.

2.2 Simplification and levels of detail

Relying on efficient visibility determination alone is not sufficient to ensure interactive performance for highly complex scenes with a lot of small scale details. In such cases, many visible modeling primitives may only project to a single pixel or sub-pixel. In order to bound the amount of data required for a given frame, a filtered representation of details must thus be available. Computing such a representation from highly detailed models, and efficiently extracting the required detail from this representation at rendering time is the goal of simplification and level of detail techniques.

2.2.1 Geometric simplification

Geometric simplification is a well studied subject, and a number of high quality automatic simplification techniques have been developed [Luebke 2001]. Optimal approximation of a surface, in terms of computing the minimal number of triangle primitives that would satisfy some approximation error metric, is known to be NP-Hard [Agarwal and Suri. 1994], and hence most research has focused on developing heuristic methods.

At the broadest level, simplification methods may be grouped into *global strategies* that are applied to the input mesh as a whole, and *local strategies* that iteratively simplify the mesh by the repeated application of some local operator. Local strategies are by far the most common simplification approaches, mainly because of their efficiency and robustness. The wide majority of the simplification methods iteratively simplifies an input mesh by sequences of vertex removals or edge contractions. In most current systems, simplification is performed in an iterative greedy fashion, which maintains a sorted list of candidate operations and applies the operation associated to the minimal simplification error at each step. Unfortunately, a direct implementation of this approach is not well-suited to work on massive meshes, since maintaining a priority queue of possible operations results in a memory consumption proportional to the size of the original mesh, a clearly untenable situation for extremely large models. Even if this obstacle could be overcome by using out-of-core data structures, the order of contraction operations could exhibit little locality in terms of memory accesses, with detrimental effects on algorithm performance.

The two main solutions that have been proposed for these problem are *streaming simplification* methods and *mesh partitioning* methods. The key insight behind streaming simplification [Wu and Kobbelt 2003; Isenburg et al. 2003] is to keep input and output data in streams that document, for example, when all triangles around a vertex or all points in a particular spatial region have arrived with ”finalization tags”. For simplification, an in-core buffer is filled and simplified and output is generated as soon as enough data is available.

Mesh partitioning methods, instead, are based on iterative simplification of mesh regions. Several authors [Hoppe 1998; Prince 2000] have proposed methods in which a mesh is segmented so that each piece fits in the main memory. While this solution is conceptually appealing, the segmenting and rejoining operations are expensive, and make this approach less attractive for very large meshes. A major drawback of these methods is that region boundaries remain unsimplified until the very last simplification step, with leading to quality and scalability problems. OEMM [Cignoni et al. 2003a] avoids the region boundary problem by exploiting a out-of-core octree-based data structure that maintains relationships between blocks and thus supports simplification of block boundaries. Another efficient technique for avoiding boundary locking has been proposed by [Cignoni et al. 2004; Cignoni et al. 2005]. In these approaches, the mesh is spatially partitioned using hierarchical volumetric subdivision schemes that create conforming volumetric meshes that support local refinement and coarsening operations.

Streaming simplification approaches lead in general to simpler and faster solutions because of their inherent I/O efficiency. On the other hand, mesh partitioning approaches are more general, can produce very high quality results, and have also the capability of producing continuous LOD representations.

2.2.2 Levels of detail

A *level-of-detail* (LOD) model is a compact description of multiple representations of a single shape and is the key element for providing the necessary degrees of freedom to achieve run-time adaptivity. LOD models can be classified as *discrete*, *progressive*, or *continuous*. Discrete models simply consist of ordered sequences of distinct representations of a shape and only support switching among representations. Progressive models consist of a coarse shape representation and of a sequence of modifications (e.g., edge splits) supporting incremental refinement. Continuous models improve over progressive models by fully supporting selective refinement, i.e., the extraction of representations with a LOD that can vary in different parts of the representation. Continuous representations can be changed on a virtually continuous scale.

A general framework for managing continuous LOD models is the multi-triangulation technique [De Floriani et al. 1998], which is based on the idea of encoding the partial order describing mutual dependencies between updates as a directed acyclic graph (DAG). In the DAG, nodes represent mesh updates (removals and/or insertions of triangles that change the representation of a mesh region), and arcs represent dependency relations among updates.

Most of the continuous LOD models can be expressed in this framework, and many variations have been proposed. Up until recently, however, the vast majority of view-dependent level-of-detail methods were all based on multi-resolution structures where LOD decisions are taken at the triangle/vertex primitive level. This kind of approach involves a constant CPU workload for each triangle and makes detail selection the bottleneck in the entire rendering process. This problem is exacerbated in rasterization approaches, because of the increasing CPU/GPU performance gap.

To overcome the detail selection bottleneck and to fully exploit the capabilities of current hardware, it is necessary to select and send batches of geometric primitives to be rendered using only a few CPU instructions. To this end, various GPU oriented multi-resolution structures have been recently proposed. The methods are based on the idea of moving the granularity of the representation from triangles to triangle patches [Cignoni et al. 2004; Yoon et al. 2004]. Thus, instead of working directly at the triangle level, the models are first partitioned into blocks containing many triangles, and, then, a multi-resolution structure is constructed among partitions. By carefully choosing appropriate subdivision structures for the partitioning and managing boundary constraints, hole-free adaptive models can be constructed. The benefit of these approaches is that the needed per-triangle workload to extract a multiresolution model is reduced by orders of magnitude. The small patches can be preprocessed and optimized off line for a more efficient rendering, and highly efficient retained mode graphics calls can be exploited for caching the current adaptive model in video memory. Recent work has shown that the vast performance increase in CPU/GPU communication results in greatly improved frame rates [Cignoni et al. 2004; Yoon et al. 2004; Cignoni et al. 2005]. Similar structures have been presented for 2D domains and have been used for terrain visualization [Cignoni et al. 2003b; Cignoni et al. 2003c], streaming [Bettio et al. 2007] and compression [Gobbetti et al. 2006]. The success of these coarse level approaches indicates the increasing importance of memory/bandwidth management issues in real-time rendering applications. Even though coarsening multiresolution granularity reduces the model flexibility and requires more triangles to achieve a given accuracy, the overall efficiency of the system is dramatically increased rather than reduced, since rendering time does not depend linearly on triangle count anymore. Instead, rendering time is strongly influenced by how the triangles are organized in memory and sent to the graphics card.

2.3 Alternate rendering primitives

So far, we have concentrated on methods centered around efficient procedures for simplifying triangle meshes, arranging details in a multiresolution structure, and efficiently extracting them at runtime to realize adaptive rendering. The complexity of the rendering operation can be also reduced by switching to representations other than triangle meshes. Representations other than polygons offer significant potential for massive models visualization.

On one hand, important model classes, such as CAD models, are well described in terms of higher order geometric primitives. One might thus consider directly rendering them instead of resorting to precomputed tessellations. The potential advantages of such an approach include a reduction of needed memory and the ability to generate smooth views at high magnification levels. On the other hand, in conventional polygon-based computer graphics, models have become so complex that for most views the projection of polygons may be smaller than one pixel in the final image. As a result, many researchers have been investigating alternate, mostly sample-based, scene representations. These representations use sets of points, voxels, or images to accelerate the rendering

2.3.1 Higher order primitives

Both raster-based and ray-tracing rendering approaches work directly and efficiently with low order primitives. High order primitives are generally tessellated into triangles or into intermediate forms in a preprocessing step. Direct rendering of the high order primitives is generally too slow to sustain interactive performance even for relatively small dataset sizes. There have been efforts to integrate high order primitives into the rendering pipeline since the early 1970's [Goldstein 1981]. This kind of work has progressed substantially, and recent approaches are using programming techniques on GPU hardware to increase performance. In particular, a number of authors have focused on devising efficient methods for raycasting quadrics, cubics, and quartics on the GPU [de Toledo and Levi 2004; Loop and Blinn 2006; Tarini et al. 2006; Sigg et al. 2006; de Toledo et al. 2007], and [Krishnamurthy et al. 2007] introduced a method for direct evaluation of NURBS surfaces on the GPU. Even when using specialized hardware, however, current systems do not match the performance of rendering from precomputed meshes. Since the performance of programmable graphics systems continues to grow, it is reasonable to expect that in the near future moderately complex models could be rendered in real-time. This is particularly important for interactive modeling applications, where manipulation of the original parametric data is important. It should also be noted that using higher-order primitives alone does not fully solve the scalability problem of massive model renderers, since at low magnification levels complex models still contain a large number primitives. The definition of a multiresolution representation above the primitive level is therefore required to support view-dependent rendering.

2.3.2 Sample-based representations

Sample-based representations occupy the opposite end of the spectrum from higher order representations. They exploit discrete sampling methods to represent complex models with sets of samples, typically points or voxels.

A point-based geometry representation can be considered a discrete sampling of a continuous surface, resulting in 3D positions $\mathbf{p}_i$, optionally with associated normal vectors $\mathbf{n}_i$ or auxiliary surface properties, e.g., colors or other material properties. One of the major benefits of such a modeling approach is its simplicity. There is no need to explicitly manage and maintain mesh connectivity during both preprocessing and rendering. On the other hand, the lack of connectivity makes it hard to reconstruct continuous (i.e., smooth and hole-free) images from such a discrete set of surface samples. Holes and gaps in-between the samples can be closed by image-space reconstruction techniques [Grossman and Dally 1998] or by object-space resampling.

The techniques from the latter category dynamically adjust the sampling rate so that the density of projected points meets the pixel resolution, which can be done both for rasterization and ray tracing approaches. Since this depends on the current viewing parameters, the resampling has to be done dynamically for each frame, and multi-resolution hierarchies or specialized procedural resamplers are exploited for this purpose. Examples are bounding sphere hierarchies [Rusinkiewicz and Levoy 2000b], dynamic sampling of procedural geometries [Stamminger and Drettakis 2001], the randomized Z-buffer [Wand et al. 2001], and the rendering of moving least squares (MLS) surfaces [Alexa et al. 2001].

As for polygonal multi-resolution rendering, amortization over a large number of primitives is essential to maximize rendering speed on current rasterization architectures. The highest performance is currently obtained by coarse-grained approaches. Coarse grained refinement for point clouds was introduced by the Layered Point Cloud multiresolution approach [Gobbetti and Marton 2004], a method that creates a coarse hierarchy over the samples of the datasets simply by reordering and clustering them into point clouds of approximately constant size arranged in a binary tree.

Even though these coarse grained techniques improve rendering speed over classic point render of over one order of magnitude, current point based techniques are competitive in terms of rendering performance with triangle mesh ones only if one uses simple unblended disks for point cloud rendering, which limits their ability to correctly treat texture and transparency and makes them more prone to produce aliasing artifacts.

Overall, peak performance of high quality techniques based on sophisticated point splatting is currently inferior to the performance of corresponding triangle rasterization and raytracing approaches, due to the additional overhead of sample blending. This situation might change in the near future, as novel architectures for hardware-accelerated rendering primitives are currently being introduced [Weyrich et al. 2007].

Sample based representations have been traditionally used to describe surfaces with oriented points. More recently, they have been used to model the appearance of small volumetric portions of the environment, which offers advantages in models with very complex geometry. In the Far Voxels approach [Gobbetti and Marton 2005], LODs are generated by discretizing spatial regions into cubical voxels. Each voxel contains a compact direction dependent approximation of the appearance of the associated volumetric subpart of the model when viewed from a distance. The approximation is constructed by a visibility aware algorithm that fits parametric shaders to samples obtained by casting rays against the full resolution dataset. The voxels are rendered using a point primitives interpreted by GPU shaders. The approach proved to be effective in cases where pure geometric simplification remains hard to apply. These cases appear in very complex models, where the visual appearance of an object depends on resolving the ordering and mutual occlusion of even very close-by surfaces, potentially with different shading properties. For such complex models, visibility preprocessing and model simplification are strictly coupled. A similar approach to model simplification is also applicable to ray tracing [Yoon et al. 2006; Dietrich et al. 2006].

2.3.3 Image-based approaches

In the geometry-based rendering approach, the visible component of the world is the union of two elements: the geometric description of the objects and the color and lighting conditions. A different approach is to consider the world as an collection of 2D images, one for each position, orientation and possibly time. The goal of *image-based rendering* (IBR) is to generate images by directly resampling such an image collection given the view parameters [McMillan and Bishop 1995], without the need of a full three-dimensional reconstruction. This approach has the theoretical advantage of decoupling rendering complexity from (geometric) scene complexity. In practice, however, a fully IBR approach is typically impractical, due to the sheer amount of data required for a full dense encoding of a complex model as a set of images. Full-scene image base rendering also loses the scene graph structure necessary for interactive selection. Restricted solutions have thus been proposed. The underlying idea behind all approaches is either to reduce the problem dimensionality by imposing constraints on viewer motion, or to compensate the aliasing effect by using additional geometric information.

Since no compensation of aliasing effects is possible without additional geometric information, either the sampling must be very dense [Gortler et al. 1996; Levoy and Hanrahan 1996], which is not practical for large scenes, or the possible viewer motion must be restricted, e.g., spherical or cylindrical panorama systems [Lippman 1980; Chen 1995].

When the viewer's motion is unrestricted, some geometric information must be employed in addition to images. In the last decade, a set of successful hybrid techniques have been proposed to accelerate the rendering of portions of a complex scene. The techniques replace complex geometry with textures in well-defined cases. In most cases, the basic idea is to use a geometry-based approach for near objects, and switch to a radically different image-based representation, called an *impostor*, for distant objects that have small, slowly changing on-screen projections. Successful examples include *portal textures* [Aliaga and Lastra 1997], *textured depth meshes* [Sillion et al. 1997; Wilson and Manocha 2003], *layered environment maps* [Jeschke et al. 2002; Jeschke and Wimmer 2002], *layered depth images* [Shade et al. 1998; Wimmer et al. 2001]. These techniques, introduced a decade ago, are enjoying a renewed interest, because of the evolution of graphics hardware, which is more and more programmable and oriented toward massively parallel rasterization.

A number of specialized hardware accelerated techniques, often based on GPU raycasting, have been introduced (e.g., *relief mapping* [Oliveira et al. 2000], and various forms of *view-dependent displacement mapping* [Wang et al. 2003; Wang et al. 2004; Baboud and Décoret 2006; Policarpo et al. 2005]). These methods have already demonstrated their applicability to massive model rendering systems [Wilson and Manocha 2003; Aliaga et al. 1999]. A very recent evolution of these methods is the BlockMap [Cignoni et al. 2007], A BlockMap compactly encodes in a single texture a set of textured vertical prisms with a bounded on-screen footprint and serves as replacement for a set of buildings in city rendering applications. One might argue that the BlockMap representation is more similar to LOD than to impostor approaches, as a BlockMap provides a view-independent, simplified representation of the original textured geometry, provides full support to visibility queries, and, when built into a hierarchy, offers multi-resolution adaptability. Similarly, encoding shape and appearance into a texture is also the goal of geometry images [Gu et al. 2002].

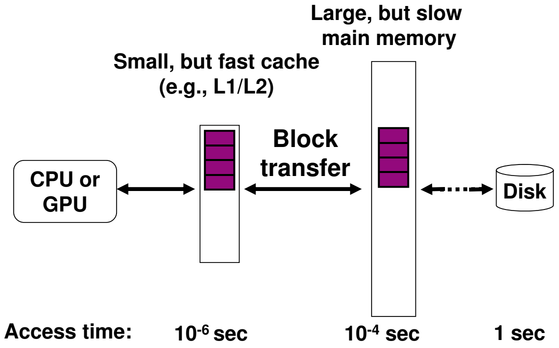

Figure 3: **Block-based Memory Hierarchy.** Lower memory levels are larger in size and slow in data access speed. Moreover, whenever there is a cache miss, data is moved in large block. Therefore, it is critical to store data accessed coherently closely in the one dimensional memory.

The evolution of the methods illustrates the convergence of rasterization and raytracing approaches, and the appeal of simple image based representation that enable the powerful GPU rasterization architecture, and more in general, streaming architectures, to process geometry in addition to images.

3 Cache-coherent layouts

Triangle meshes and various hierarchies are frequently used in many different applications, including rasterization and ray-tracing methods. Typically, we store data elements of these data structures sequentially in main memory and disk. A layout is simply an order of data elements. For example, a vertex layout of a triangle mesh is an order of vertices of the mesh. Some of well known layout methods in computer graphics include triangle strips [Deering 1995; Hoppe 1999] and space filling curves [Sagan 1994].

Data elements with a particular layout can be stored in a one dimensional array and accessed during runtime application. Triangle meshes and hierarchies usually have more than one dimensional structures. Therefore, when mapping data elements (e.g., vertices) of meshes and hierarchies into a one dimension array, two data elements (two vertices connected by an edge) located very closely in the mesh may be stored very far away in the one dimensional array. Intuitively speaking, it is likely that we may access those two vertices sequentially since they are located very close to one another in the mesh during accessing vertices of the mesh. However, it is also likely that we may get cache misses if the two vertices are stored far from one another in the one dimensional array.

Let us look at the phenomenon mentioned above in the context of caching architectures. Most modern computers use memory hierarchies, where each level of memory serves as a cache for the next level (see Fig. 3). Initially data is sequentially stored at the lowest and slowest level, typically disk, of the memory hierarchy. Memory hierarchies have two main characteristics. First, lower levels are larger in size, farther from CPUs/GPU, and, therefore, slower in access speed. Second, whenever a cache miss occurs, the accessed data move in a large block between memory levels. Since data move in a large block, it is critical to store data that are likely to be accessed sequentially close to one another in the one dimensional array.

3.1 Cache-oblivious mesh layouts

Layouts of triangle meshes can be classified into cache-aware and cache-oblivious layouts. Cache-aware layouts are constructed to minimize the number of cache misses when accessing the layout for a given block size. On the other hand, cache-oblivious layouts are designed for various block sizes. Ideally, cache-aware layouts can achieve better performance since they are more optimized for a specific block size. However, different machines can have different block sizes. Moreover, there are many different block sizes ina single machine since current architectures have many memory levels like L1/L2 caches, main memory, disk. Therefore, cache-oblivious layouts optimized for various block sizes can show high performance on various machines with different block sizes. Moreover, once a cache-oblivious layout of a mesh is computed, there is no need to re-compute it unless there are significant modifications to the mesh structure.

The computation of layouts of triangle meshes can be cast as an optimization problem. The main technical challenge is to compute an optimization metric that has strong correlation with the number of cache misses during accessing the layout by runtime applications. The optimization metric is defined as the measure of an expected number of cache misses during coherent random access on the input mesh. In the case of computing cache-oblivious layouts that are optimized for various block sizes, it is critical to consider block sizes that are likely to be encountered in current caching architecture. Please note that most block sizes have power-of-two bytes (e.g., 32B for L1, 64B for L2, 4KB for page block). For these block sizes that are geometrically increasing, the expected number of cache misses is reduced to a geometric mean of edge lengths of the mesh [Yoon and Lindstrom 2006]. A length of an edge consisting of two vertices is defined by an index difference of those two vertices in the layout.

The layout optimization method should efficiently handle massive models. Multi-level optimization method has been shown to work fast even for massive models that cannot fit into main memory [Yoon et al. 2005]. The main idea of multi-level optimization method is to recursively construct the layout from a coarse level to a fine level of a layout. The source of this method is available at `http://gamma.cs.unc.edu/COL/OpenCCL/`.

3.2 Cache-efficient layouts of bounding volume hierarchies

Bounding volume hierarchies (BVH) are object partitioning trees and have been used to accelerate the performance of visibility tests in ray tracing and rasterization. They are also employed to speed-up collision detection. In all these applications, BVHs are traversed from the root node and checked for collisions with rays, frusta or other objects. If there is collision, child nodes are recurively traversed until a miss occurs.

Cache-coherent layouts of BVHs strive to reduce the number of cache misses and improve the performance of BVH-based algorithms. One would like to use data layout optimization techniques to organize the nodes of a BVH in memory and reduce the number of cache misses at runtime. It is quite possible to apply the mesh optimization method described in Sec. 3.1 to the layout computation of nodes of BVHs. However, one can design a new layout method optimized further for BVHs. Please note that BVHs are usually accessed from the root node to leaf nodes. This access pattern on the BVHs is not fully accommodated in the mesh layout optimization method. In order to address this problem, a specialized layout technique for BVHs has been proposed in [Yoon and Manocha 2006]. By applying the computed layouts of BVHs to ray tracing and collision detection, a meaningful performance improvement (30%–180%) has been achieved compared to other well known BVH layouts, including the van Emde Boas layout [van Emde Boas 1977].

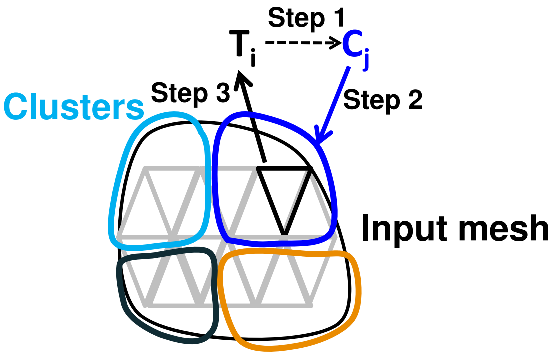

Figure 4: **Overall framework supporting random access on the compressed mesh:** If at runtime a triangle T_i is requested then, the runtime framework first identifies a cluster index, C_j, of a cluster containing the requested triangle. The second step is to load, decompress, and store the cluster in main memory. The third step is to return the requested triangle to the application.

4 Compression Techniques

Storing massive models with cache-coherent layouts can reduce the number of cache misses when accessing the models and, thus, improve the performance of various applications. Although these cache-coherent layouts of massive models can support random access to the stored models, like other popular mesh file representations (e.g., ply and obj formats), a raw cache-coherent layout format can take huge amounts of disk space, mainly due to the stored connectivity information. For example, the St. Matthew model consists of 372 million triangles and its cache-coherent layout format with connectivity information takes approximately 6GB. Mesh compression techniques can be used to drastically reduce the disk space requirement and, possibly, improve the performance of rendering applications by reducing the number of expensive disk reads.

Mesh compression techniques have been widely researched and good surveys are available [Alliez and Gotsman 2005; Gotsman et al. 2002]. In the context of rendering, triangle strips or rendering sequences [Deering 1995] have been used to compactly encode triangles of a triangular mesh. A rendering sequence of a mesh is a list of vertices encoding vertex connectivity and triangles of the mesh. Since modern GPUs maintain a small buffer to reuse recently accessed vertices, rendering sequences can improve the performance of rendering and reduce storage requirement. However, these rendering sequences are not directly applicable to other applications. For example, ray tracing requires random access to the compressed data representations since rendering sequences do not support such random access.

Unlike rasterization that sequentially traverses triangles of a mesh, ray tracing, visibility culling, collision detection, and isosurface extraction force a random access pattern to the underlying data structures. Since most of prior mesh compression techniques are designed for data archive or efficient data transmission, these techniques sequentially compress and decompress the underlying data representations. If we want to access a triangle in the middle of a series of compressed triangles with these sequential compression methods, we have to decompress the whole compressed mesh. Therefore, these sequential compression/decompression techniques are not directly applicable to the cases where random access is required to the compressed meshes. Recently, a few approaches have been introduded to provide random access to the compressed data representations.

4.1 Triangle meshes

Yoon and Lindstrom [2007] proposed random-accessible compressed triangle meshes. In order to support random access to the compressed meshes, they decompose a mesh into a set of clusters, each of which contains a few thousands vertices and triangles. Then, compression and decompression are performed at the granularity of clusters. At runtime, if an application requests a triangle, the proposed technique first identifies a cluster containing the requested triangle, decompresses the cluster without decompressing other clusters, and returns the decompressed data to the application. The overall framework is shown in Fig. 4. Moreover, the proposed technique preserves an original layout of the mesh and is able to maintain benefits of cache-coherent layouts of the mesh. This technique shows up to 20 to 1 data compression ratio and 6:1 runtime performance improvement compared to using un-compressed mesh representations. The source of this method is available at `http://jupiter.kaist.ac.kr/~sungeui/OpenRACM`. Choe et al. [Choe et al. 2004] proposed similar random-accessible compressed meshes, but does not preserve the original layout of the mesh.

4.2 Hierarchies

Ray tracing uses acceleration hierarchies such as bounding volume hierarchies to efficiently find a triangle intersecting with a ray. However, these acceleration hierarchies typically have a high data requirement. Lauterbach et al. [Lauterbach et al. 2007] introduced the Ray-Strip representation, which is a compact representation encoding the acceleration data structure and mesh information for ray tracing. A Ray-Strip consists of a list of vertices, which implicitly encodes a list of triangles like the triangle strip used in rasterization. A Ray-Strip implicitly encodes a balanced hierarchy that can be used for ray tracing. To do that, a list of vertices is decomposed into two equal-sized sets of vertices. Then, a bounding volume information is computed for each set. Since the structure of the hierarchy is reduced to a simple complete tree, a Ray-Strip representation can compactly encode mesh information and hierarchy information for ray tracing.

5 Conclusion

We have discussed visibility culling, simplification, cache-coherent layouts, and data compression methods as efficient data management techniques enabling interactive visualization of massive models. Since model complexity is highly likely to increase and the gap between the data access speed and computation speed continues to widen, further research is required to achieve interactive visualization of more massive models that will come in near future.

Acknowledgments. This work was partially supported by: the Italian Ministry of University and Research under grant CYBERSAR; a KAIST seed grant; the MIC, under grant "Development of Elemental Technology for Promoting Digital Textbook and u-Learning".

References

Agarwal, P. K., and Suri., S. 1994. Surface approximation and geometric partitions. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, 24?33.

Airey, J. M., Rohlf, J. H., and Brooks, Jr., F. P. 1990. Towards image realism with interactive update rates in complex virtual building environments. *Computer Graphics (1990 Symposium on Interactive 3D Graphics) 24*, 2 (Mar.), 41–50.

Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C. T. 2001. Point Set Surfaces. In *Proceedings of IEEE Visualization 2001*, 21–28.

Aliaga, D. G., and Lastra, A. A. 1997. Architectural Walkthroughs Using Portal Textures. In *Proceedings of IEEE Visualization 1997*, 355–362.

Aliaga, D. G., Cohen, J., Wilson, A., Baker, E., Zhang, H., Erikson, C., Hoff III, K. E., Hudson, T., Stürzlinger, W., Bastos, R., Whitton, M. C., Brooks Jr., F. P., and Manocha, D. 1999. MMR: An Interactive Massive Model Rendering System using Geometric and Image-Based Acceleration. In *SI3D '99: Proceedings of the 1999 Symposium on Interactive 3D Graphics*, 199–206.

Alliez, P., and Gotsman, C. 2005. *Recent advances in compression of 3D meshes*. Springer, 3–26.

Andújar, C., Saona-Vázquez, C., and Navazo, I. 2000. Lod visibility culling and occluder synthesis. *Computer-Aided Design 32*, 13 (Oct.), 773–783.

Baboud, L., and Décoret, X. 2006. Rendering geometry with relief textures. In *Graphics Interface*, Canadian Human-Computer Communications Society, C. Gutwin and S. Mann, Eds., 195–201.

Bettio, F., Gobbetti, E., Marton, F., and Pintore, G. 2007. High-quality networked terrain rendering from compressed bitstreams. In *Proc. ACM Web3D International Symposium*, New York, NY, USA, ACM Press, 37–44.

Bittner, J., Wonka, P., and Wimmer, M. 2001. Visibility preprocessing for urban scenes using line space subdivision. In *PG '01: Proceedings of the 9th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, USA, 276.

Bittner, J., Wimmer, M., Piringer, H., and Purgathofer, W. 2004. Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum 23*, 3, 615–624.

Bittner, J. 2002. *Hierarchical Techniques for Visibility Computations*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague.

Chen, S. E. 1995. Quicktime VR - an image-based approach to virtual environment navigation. In *SIGGRAPH 95 Conference Proceedings*, Addison Wesley, R. Cook, Ed., Annual Conference Series, ACM SIGGRAPH, 29–38. held in Los Angeles, California, 06-11 August 1995.

Choe, S., Kim, J., Lee, H., Lee, S., and Seidel, H.-P. 2004. Mesh compression with random accessibility. In *Israel-Korea Bi-National conf.*

Cignoni, P., Montani, C., Rocchini, C., and Scopigno, R. 2003. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics 9*, 525–337.

Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopigno, R. 2003. BDAM – batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum 22*, 3 (September), 505–514. Proc. Eurographics 2003.

Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopigno, R. 2003. Planet–sized batched dynamic adaptive meshes (p-bdam). In *Proceedings IEEE Visualization*, IEEE Computer Society Press, Conference held in Seattle, WA, USA, 147–155.

Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopigno, R. 2004. Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Transactions on Graphics 23*, 3 (Aug.), 796–803.

Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopigno, R. 2005. Batched multi triangulation. In *Proceedings IEEE Visualization*, IEEE Computer

Society Press, Conference held in Minneapolis, MI, USA, 207–214.

CIGNONI, P., DI BENEDETTO, M., GANOVELLI, F., GOBBETTI, E., MARTON, F., AND SCOPIGNO, R. 2007. Ray-Casted BlockMaps for Large Urban Models Visualization. In *Computer Graphics Forum (Proceedings of Eurographics)*. To appear.

DE FLORIANI, L., MAGILLO, P., AND PUPPO, E. 1998. Efficient Implementation of Multi-Triangulations. In *Proceedings of IEEE Visualization 1998*, 43–50.

DE TOLEDO, R., AND LEVI, B. 2004. Extending the graphic pipeline with new GPU-accelerated primitives. In *Proc. 24th gOcad Meeting*.

DE TOLEDO, R., LEVY, B., AND PAUL, J.-C. 2007. Iterative methods for visualization of implicit surfaces on gpu. In *ISVC, International Symposium on Visual Computing*, Springer, Lake Tahoe, Nevada/California, Lecture Notes in Computer Science.

DEERING, M. F. 1995. Geometry compression. In *ACM SIGGRAPH*, 13–20.

DIETRICH, A., SCHMITTLER, J., AND SLUSALLEK, P. 2006. World-space sample caching for efficient ray tracing of highly complex scenes. Tech. Rep. TR-2006-01, Computer Graphics Group, Saarland University.

DUGUET, F., AND DRETTAKIS, G. 2002. Robust epsilon visibility. In *Proceedings of ACM SIGGRAPH 2002*, ACM Press / ACM SIGGRAPH, J. Hughes, Ed.

DURAND, F., DRETTAKIS, G., THOLLOT, J., AND PUECH, C. 2000. Conservative visibility preprocessing using extended projections. In *SIGGRAPH 00 Conference Proceedings*, 239–248.

DURAND, F. 1999. *3D Visibility: Analytical study and Applications*. PhD thesis, Universite Joseph Fourier, Grenoble, France.

GOBBETTI, E., AND MARTON, F. 2004. Layered point clouds: a simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Computers & Graphics 28*, 6 (Dec.), 815–826.

GOBBETTI, E., AND MARTON, F. 2005. Far Voxels – a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. *ACM Transactions on Graphics 24*, 3, 878–885.

GOBBETTI, E., MARTON, F., CIGNONI, P., DI BENEDETTO, M., AND GANOVELLI, F. 2006. C-BDAM – compressed batched dynamic adaptive meshes for terrain rendering. *Computer Graphics Forum 25*, 3 (September), 333–342. Proc. Eurographics 2006.

GOLDSMITH, J., AND SALMON, J. 1987. Automatic creation of object hierarchies for ray tracing. *IEEE Comput. Graph. Appl. 7*, 5, 14–20.

GOLDSTEIN, R. 1981. Defining the bounding edges of a synthavision solid model. In *18th Conference on Design Automation*, 457–461.

GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. F. 1996. The lumigraph. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, 43–54.

GOTSMAN, C., GUMHOLD, S., AND KOBBELT, L. 2002. *Simplification and Compression of 3D Meshes*. Springer, 319–361.

GOVINDARAJU, N. K., SUD, A., YOON, S.-E., AND MANOCHA, D. 2003. Interactive visibility culling in complex environments using occlusion-switches. In *2003 ACM Symposium on Interactive 3D Graphics*, 103–112.

GROSSMAN, J., AND DALLY, W. J. 1998. Point Sample Rendering. In *Rendering Techniques 1998 (Proceedings of the Eurographics Workshop on Rendering)*, 181–192.

GU, X., GORTLER, S. J., AND HOPPE, H. 2002. Geometry Images. In *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 335–361.

HAUMONT, D., MAKINEN, O., AND NIRENSTEIN, S. 2005. A low dimensional framework for exact polygon-to-polygon occlusion queries. In *Rendering Techniques*, Eurographics Association, O. Deussen, A. Keller, K. Bala, P. Dutr?, D. W. Fellner, and S. N. Spencer, Eds., 211–222.

HAVRAN, V., HERZOG, R., AND SEIDEL, H.-P. 2006. On the fast construction of spatial data structures for ray tracing. In *Proceedings of IEEE Symposium on Interactive Ray Tracing 2006*, 71–80.

HAVRAN, V. 2000. *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague.

HEYER, M., PFÜTZER, S., AND BRÜDERLIN, B. 2005. Visualization Server for Very Large Virual Reality Scenes. In *4. Paderborner Workshop Augmented & Virtual Reality in der Produktentstehung*.

HOPPE, H. 1998. Smooth view-dependent level-of-detail control and its aplications to terrain rendering. In *IEEE Visualization '98 Conf.*, 35–42.

HOPPE, H. 1999. Optimization of mesh locality for transparent vertex caching. *ACM SIGGRAPH*, 269–276.

HUNT, W., MARK, W. R., AND STOLL, G. 2006. Fast kd-tree construction with an adaptive error-bounded heuristic. In *2006 IEEE Symposium on Interactive Ray Tracing*, IEEE.

ISENBURG, M., LINDSTROM, P., GUMHOLD, S., AND SNOEYINK, J. 2003. Large Mesh Simplification using Processing Sequences. In *Proceedings of IEEE Visualization 2003*, 465–472.

JESCHKE, S., AND WIMMER, M. 2002. Textured depth meshes for realtime rendering of arbitrary scenes. In *Proceedings of the 13th Eurographics Workshop on Rendering (RENDERING TECHNIQUES-02)*, Eurographics Association, Aire-la-Ville, Switzerland, S. Gibson and P. Debevec, Eds., 181–190.

JESCHKE, S., WIMMER, M., AND SCHUMANN, H. 2002. Layered environment-map impostors for arbitrary scenes. In *Graphics Interface*, 1–8.

KLOSOWSKI, J. T., AND SILVA, C. T. 2001. Efficient conservative visibility culling using the prioritized-layered projection algorithm. *IEEE Transactions on Visualization and Computer Graphics 7*, 4, 365–379.

KOLTUN, V., CHRYSANTHOU, Y., AND COHEN-OR, D. 2001. Hardware-accelerated from-region visibility using a dual ray space. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, Springer-Verlag, London, UK, 205–216.

KRISHNAMURTHY, A., KHARDEKAR, R., AND MCMAINS, S. 2007. Direct evaluation of nurbs curves and surfaces on the gpu. In *SPM '07: Proceedings of the 2007 ACM symposium on Solid and physical modeling*, ACM, New York, NY, USA, 329–334.

LAUTERBACH, C., YOON, S.-E., TUFT, D., AND MANOCHA, D. 2006. Rt-deform: Interactive ray tracing of dynamic scenes using bvhs. In *Proceedings of IEEE Symposium on Interactive Ray Tracing 2006*.

LAUTERBACH, C., YOON, S.-E., AND MANOCHA, D. 2007. Ray-strips: A compact mesh representation for interactive ray tracing.

In *IEEE/EG Symposium on Interactive Ray Tracing*, 19–26.

Levoy, M., and Hanrahan, P. 1996. Light field rendering. In *SIGGRAPH 96 Conference Proceedings*, 31–42.

Leyvand, T., Sorkine, O., and Cohen-Or, D. 2003. Ray space factorization for from-region visibility. *ACM Transactions on Graphics 22*, 3 (July), 595–604.

Lippman, A. 1980. Movie-maps: An application of the optical videodisc to computer graphics. *Computer Graphics (SIGGRAPH ?80 Proceedings) 14*, 3 (July), 32?42.

Loop, C., and Blinn, J. 2006. Real-time gpu rendering of piecewise algebraic surfaces. *ACM Transactions on Graphics 25*, 3 (July), 664–670.

Luebke, D. P. 2001. A Developer's Survey of Polygonal Simplification Algorithms. *IEEE Computer Graphics and Applications 21*, 3, 24–35.

MacDonald, J. D., and Booth, K. S. 1990. Heuristics for ray tracing using space subdivision. *Visual Computer*.

McMillan, L., and Bishop, G. 1995. Plenoptic Modeling: An Image-Based Rendering System. In *ACM Computer Graphics (Proceedings of ACM SIGGRAPH)*, 39–46.

Mora, F., Aveneau, L., and Mériaux, M. 2005. Coherent and exact polygon-to-polygon visibility. In *Proc. WSCG*, 87–94.

Nirenstein, S., and Blake, E. 2004. Hardware accelerated visibility preprocessing using adaptive sampling. In *Rendering Techniques 2004: 15th Eurographics Workshop on Rendering*, 207–216.

Nirenstein, S., Blake, E., and Gain, J. 2002. Exact from-region visibility culling. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 191–202.

Oliveira, M. M., Bishop, G., and McAllister, D. 2000. Relief Texture Mapping. In *ACM Computer Graphics (Proceedings of ACM SIGGRAPH)*, 359–368.

Patterson, D., Anderson, T., Cardwell, N., Fromm, R., Kimberly, Keaton, ChristoforosKazyrakis, Thomas, R., and Yellick, K. 1997. A case for intelligent ram. *IEEE Micro.*.

Policarpo, F., Oliveira, M. M., and Comba, J. L. D. 2005. Real-time relief mapping on arbitrary polygonal surfaces. *ACM Trans. Graph 24*, 3, 935.

Popov, S., Günther, J., Seidel, H.-P., and Slusallek, P. 2006. Experiences with streaming construction of SAH KD-trees. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, 89–94.

Prince, C. 2000. *Progressive Meshes for Large Models of Arbitrary Topology*. Master's thesis, Department of Computer Science and Engineering, University of Washington, Seattle.

Reshetov, A., Soupikov, A., and Hurley, J. 2005. Multi-Level Ray Tracing Algorithm. In *ACM Transaction of Graphics (Proceedings of ACM SIGGRAPH)*, 1176–1185.

Ruemmler, C., and Wilkes, J. 1994. An introduction to disk drive modeling. *IEEE Computer*.

Rusinkiewicz, S., and Levoy, M. 2000. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 343–352.

Rusinkiewicz, S., and Levoy, M. 2000. QSplat: A Multiresolution Point Rendering System for Large Meshes. In *Computer Graphics (Proceedings of ACM SIGGRAPH)*, 343–352.

Sagan, H. 1994. *Space-Filling Curves*. Springer-Verlag.

Samet, H., Ed. 2006. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann.

Schaufler, G., J.Dorsey, Decoret, X., and Sillion, F. 2000. Conservative volumetric visibility with occluder fusion. In *SIGGRAPH 00 Conference Proceedings*, 229–238.

Shade, J., Gortler, S., He, L., and Szeliski, R. 1998. Layered Depth Images. In *Computer Graphics (Proceedings of ACM SIGGRAPH)*, 231–242.

Shevtsov, Maxim, Soupikov, Alexei, Kapustin, and Alexander. 2007. Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes. *Computer Graphics Forum 26*, 3 (September), 395–404.

Sigg, C., Weyrich, T., Botsch, M., and Gross, M. 2006. Gpu-based ray-casting of quadratic surfaces. In *Symposium on Point - Based Graphics 2006*, 59–66.

Sillion, F., Drettakis, G., and Bodelet, B. 1997. Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery. In *Computer Graphics Forum (Proceedings of Eurographics)*, 207–218.

Stamminger, M., and Drettakis, G. 2001. Interactive Sampling and Rendering for Complex and Procedural Geometry. In *Proceedings of the Eurographics Workshop on Rendering Techniques*, 151–162.

Tarini, M., Cignoni, P., and Montani, C. 2006. Ambient occlusion and edge cueing for enhancing real time molecular visualization. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (Sept./Oct.), 1237–1244.

Teller, S., and Sequin, C. 1991. Visibility preprocessing for interative walkthroughs. *Computer Graphics (SIGGRAPH 91 Proceedings) 25*, 4 (July), 61–69.

van Emde Boas, P. 1977. Preserving order in a forest in less than logarithmic time and linear space. *Inf. Process. Lett.*.

Wächter, C., and Keller, A. 2006. Instant ray tracing: The bounding interval hierarchy. In *Proceedings of the Eurographics Symposium on Rendering*, 139–149.

Wald, I., and Havran, V. 18-20. On building fast kd-trees for ray tracing, and on doing that in o(n log n). In *Proceedings of IEEE Symposium on Interactive Ray Tracing 2006*, 61–69.

Wald, I., Slusallek, P., Benthin, C., and Wagner, M. 2001. Interactive rendering with coherent ray tracing. *Computer Graphics Forum 20*, 3, 153–164.

Wald, I., Boulos, S., and Shirley, P. 2007. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics 26*, 1, 6.1–6.10.

Wald, I. 2007. On fast construction of sah based bounding volume hierarchies. In *Proceedings of the 2007 Eurographics/IEEE Symposium on Interactive Ray Tracing*.

Wand, M., Fischer, M., Peter, I., auf der Heide, F. M., and Strasser, W. 2001. The Randomized z-Buffer Algorithm: Interactive Rendering of Highly Complex Scenes. In *Computer Graphics (Proceedings of ACM SIGGRAPH)*, 361–370.

Wang, L., Wang, X., Tong, X., Lin, S., Hu, S.-M., Guo, B., and Shum, H.-Y. 2003. View-Dependent Displacement Mapping. In *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 334–339.

Wang, X., Tong, X., Lin, S., Hu, S., Guo, B., and Shum, H.-Y. 2004. Generalized displacement maps. In *Proceedings of* *the 2004 Eurographics Symposium on Rendering*, Eurographics Association, D. Fellner and S. Spencer, Eds., 227–234.

WEYRICH, T., FLAIG, C., HEINZLE, S., MALL, S., AILA, T., ROHRER, K., FASNACHT, D., FELBER, N., OETIKER, S., KAESLIN, H., BOTSCH, M., AND GROSS, M. 2007. A Hardware Architecture for Surface Splatting. In *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 90.

WILSON, A., AND MANOCHA, D. 2003. Simplifying Complex Environments Using Incremental Textured Depth Meshes. In *ACM Transactions on Graphics (Proceedings of ACM SIGGRPAH)*, 678–688.

WIMMER, M., WONKA, P., AND SILLION, F., 2001. Point-based impostors for real-time visualization, May 29.

WONKA, P., WIMMER, M., AND SCHMALSTIEG, D. 2000. Visibility preprocessing with occluder fusion for urban walkthroughs. In *11th Eurographics Workshop on Rendering*, 71–82.

WONKA, P., WIMMER, M., ZHOU, K., MAIERHOFER, S., HESINA, G., AND RESHETOV, A. 2006. Guided visibility sampling. *ACM Transactions on Graphics 25*, 3 (July), 494–502.

WOOP, S., MARMITT, G., AND SLUSALLEK, P. 2006. B-KD Trees for Hardware Accelerated Ray Tracing of Dynamic Scenes. In *Proceedings of Graphics Hardware*.

WU, J., AND KOBBELT, L. 2003. A stream algorithm for the decimation of massive meshes. In *Proc. Graphics Interface*, 185–192.

YOON, S.-E., AND LINDSTROM, P. 2006. Mesh layouts for block-based caches. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization) 12*, 5.

YOON, S.-E., AND LINDSTROM, P. 2007. Random-accessible compressed triangle meshes. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization)*.

YOON, S.-E., AND MANOCHA, D. 2006. Cache-efficient layouts of bounding volume hierarchies. *Computer Graphics Forum (Eurographics) 25*, 507–516.

YOON, S.-E., SALOMON, B., GAYLE, R., AND MANOCHA, D. 2004. Quick-VDR: Interactive View-Dependent Rendering of Massive Models. In *Proceedings of IEEE Visualization 2004*, 131–138.

YOON, S.-E., LINDSTROM, P., PASCUCCI, V., AND MANOCHA, D. 2005. Cache-Oblivious Mesh Layouts. *Proc. of ACM SIGGRAPH*.

YOON, S.-E., LAUTERBACH, C., AND MANOCHA, D. 2006. R-LODs: Fast LOD-Based Ray Tracing of Massive Models. *The Visual Computer 22*, 9–11, 772–784.

YOON, S., CURTIS, S., AND MANOCHA, D. 2007. Ray tracing dynamic scenes using selective restructuring. *Proc. of Eurographics Symposium on Rendering*.