Eurographics Symposium on Rendering 2008
Steve Marschner and Michael Wimmer
(Guest Editors)

Volume 27 (2008), Number 4

ReduceM: Interactive and Memory Efficient Ray Tracing of Large Models

Christian Lauterbach[1] Sung-eui Yoon[2] Ming Tang[3] Dinesh Manocha[1]

[1]University of North Carolina at Chapel Hill
[2]Korea Advanced Institute of Science and Technology
[3]Zhejiang University

Abstract

We present a novel representation and algorithm, ReduceM, for memory efficient ray tracing of large scenes. ReduceM exploits the connectivity between triangles in a mesh and decomposes the model into triangle strips. We also describe a new stripification algorithm, Strip-RT, that can generate long strips with high spatial coherence. Our approach uses a two-level traversal algorithm for ray-primitive intersection. In practice, ReduceM can significantly reduce the storage overhead and ray trace massive models with hundreds of millions of triangles at interactive rates on desktop PCs with 4-8GB of main memory.

1. Introduction

Ray tracing has recently emerged as a viable alternative to rasterization for interactive applications. This is mainly due to the improvement in processing speed along with algorithmic developments that use optimized hierarchical representations and ray coherence techniques. Current ray tracers can render large models composed of a few millions of triangles at interactive rates (i.e. 10 frames per second) on current laptop or desktop systems. In this paper, we address the problem of interactive ray tracing of massive data sets on commodity desktop systems. Models with tens or hundreds of millions of triangles are commonly used in different applications including scientific visualization, terrain rendering, CAD/CAM, virtual environments, etc. The complexity of these models entails new challenges in terms of storing these data sets as well as ray tracing them in real time.

Current ray tracers use spatial partitioning or bounding volume hierarchies as acceleration structures. These hierarchies result in logarithmic behavior of ray tracing as a function of the number of primitives of models. At the same time, the use of hierarchies introduces many issues related to memory efficiency. The memory access pattern of a hierarchy traversal is non-local and can result in large working set sizes. If the size of the original model and the hierarchy exceeds the available main memory, the performance of ray tracing will degrade considerably due to slow disk I/O performance. There is considerable prior work on reducing the memory overhead of ray tracing algorithms. At a broad level these algorithms use a more compactrepresentation of the geometric primitives or the acceleration structure, perform memory reordering operations, use precomputed levels-of-detail, or use representations that implicitly represent the connectivity between the primitives (e.g. triangle strips). However, it still remains a challenge to ray trace large data sets (e.g. Boeing 777) on current desktop or laptop systems with only $4-8$ GB of main memory at interactive rates.

Main Results: We present a novel representation and a construction algorithm for memory efficient ray tracing of large scenes. Our formulation exploits the connectivity information between the mesh triangles and represents them as triangle strips. The two novel aspects of our work include:

1. ReduceM representation: We represent the model using a two level hierarchy. The lower levels correspond to strip hierarchies, and each strip hierarchy implicitly encodes a hierarchy generated on a triangle strip. The higher level hierarchy is a global hierarchical acceleration structure (e.g. a kd-tree), whose leaf nodes contain the strip hierarchies. This two level formulation is compact and requires less memory footprint than other hierarchies used for interactive ray tracing. Moreover, the ReduceM representation provides coherent access to the encoded vertices and thereby results in im-

proved run-time performance. Our overall representation is lossless and we describe an efficient algorithm for traversing the two-level hierarchies and performing intersection computations.

2. Strip-RT stripification algorithm: In order to achieve low memory overhead and high run-time performance for ray tracing, we present a new stripification method, Strip-RT, to generate triangle strips. Our method is based on two criteria: maximizing the length of the strip and increasing the spatial coherence of the acceleration hierarchy implicitly encoded in the triangle strip. We use a hierarchical method to compute the triangle strips based on the surface-area heuristic (SAH) metric. We show that Strip-RT results in improved performance as compared to prior stripification algorithms that were designed primarily for fast rasterization.

We have tested the performance of ReduceM on massive models with tens and hundreds of millions of triangles. These include CAD data sets, scanned models and iso-surfaces generated from scientific simulation. As compared to the currently fastest ray tracing algorithms based on kd-trees for massive models [RSH05, WDS04], ReduceM decreases the total storage overhead by up to 5 times and the runtime memory requirement for hierarchy and connectivity by over 10 times. This makes it possible to represent the complex mesh and the hierarchy of Boeing 777 model (with 362M triangles) using only 9GB of memory and ray trace it at 10 fps on a multi-core workstation. On the other hand, prior interactive algorithms using kd-trees needed more than 30GB of main memory for such models [SBB*06]. As compared to ray-strips [LYM07], the combination of of ReduceM and Strip-RT improves the frame rate by $50-80\%$ and at the same time reduces the overall storage complexity by $20-30\%$ (i.e. up to 3 GB for massive models). In practice, our approach can ray trace such massive models at interactive rates on desktop workstations.

2. Related work

The development of memory-efficient representations has a long history in graphics. In this section, we briefly survey related work in the context of ray tracing and mesh representations.

Ray tracing large models: Ray tracing of large data sets has been an active research area. There are many in-core algorithms that use large, shared memory systems to render these data sets [SBB*06, DSW07] while using standard ray tracing representations. Other methods to improve the performance of out-of-core ray tracing include reordering rays [PKGH97, DGP04, EMAM07] and latency hiding [WDS04]. However, these methods may not reduce the storage overhead. Many level-of-detail algorithms [CLF*03, WDS04, YLM06] have been proposed to reduce the working set size of ray tracing. However, they may result in visual artifacts. Other work has concentrated on reducing the size of the ray tracing hierarchy by either quantization [Mah05, CSE06], using a compact object hierarchy with lazy building [WK06], or efficient culling methods to allow a less detailed and thus smaller hierarchy [Res07]. These methods could be combined with our proposed methods to further reduce the memory requirement of ray tracing massive models.

Mesh Representations: There is considerable work on computing memory-efficient mesh representations for interactive rasterization. These include triangle strips [Dee95], rendering sequences [Hop99], and cache-oblivious mesh layouts [YLPM05]. In the ray tracing literature, there is work on improved representations for subdivision meshes [KDS98, MTF03, CLF*03, SMD*06]. Amanatides and Choi [AC97] present an edge-based ray-mesh intersection method for regular meshes using Plücker coordinates. For general triangular meshes, the most recent work uses triangle strips [LYM07].

3. Overview

In this section, we first describe the main issues in designing memory efficient representations for ray tracing. Then, we present our representation, ReduceM, and describe how the representation can be used to accelerate ray tracing on large models.

3.1. Memory Issues for Interactive Ray Tracing

Most interactive ray tracing approaches have considerable memory overhead. This is mainly due to the use of hierarchical acceleration structures for efficient ray-intersection tests. These hierarchies affect the performance in several ways: First, the memory overhead of hierarchical acceleration structures can be high in addition to that of scene primitives. For example, hierarchies (e.g., kd-trees) optimized for run-time performance according to the surface area heuristic (SAH) [GS87, MB90, Hav00], tend to have more nodes and may take almost as much memory as the triangle primitives. Second, hierarchy traversal for ray tracing can have a complex memory access pattern. Given the block-based data fetching memory hierarchy, the working set size of the ray tracer can thus be high. Finally, ray tracers typically require random access to the primitives since any triangle can intersect with the ray during traversal. Therefore, triangles are commonly stored in an indexed triangle list or similar representation, which can have large memory overhead in contrast to more efficient storage methods used in rasterization.

The combination of large working set size and non-local memory access pattern during the hierarchy traversal significantly affects the overall performance of ray tracing. When the model and the acceleration structure fit in the main memory, ray tracing has been shown to be very cache coherent for in-core operation and ray packets [Wal04]. However, when the model and its hierarchy do not fit into main memory, its performance is dominated by the slow disk I/O performance [WDS04, YLM06].

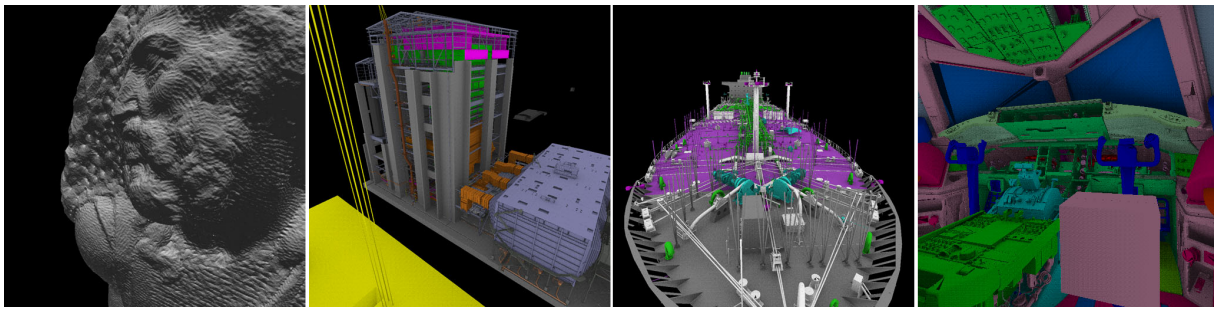

Figure 1: ***Benchmarks:*** *Our ReduceM approach allows interactive ray tracing visualization effects on workstation machines: St.Matthew (372M tris, primary visibility, 4.81fps), Power plant (12.7M tris, 16 ambient occlusion samples, 1 fps), Double Eagle tanker (82M tris, ambient occlusion 16 samples, 0.5 fps), Boeing 777 (364M tris, ambient occlusion 16 samples, 0.25 fps) (all results at* 1024×1024*)*

One approach to reduce the memory problem of ray tracing has been *reordering* the rays [PKGH97, DGP04, EMAM07]. In many ways, ray coherence techniques [Wal04, RSH05] can also be considered as reordering methods since their core operation is to group hierarchy traversal and primitive intersection operations such that each node or primitive element only needs to be loaded once for a group of rays. However, the downside of these methods is that they require sufficient ray coherence, which is only available for a restricted set of rays such as primary or shadow rays, but not for many secondary effects such as ambient occlusion or path tracing. In addition, coherence also commonly decreases with high geometric complexity, which limits its applicability to large models [Wal04].

3.2. ReduceM Representation

One of the reasons for the high performance of rasterization algorithms has been the almost linear memory access pattern and small working set size. However, such a pattern for rasterization cannot be directly replicated for ray tracing. On the other hand, a good alternative to accelerate ray tracing is to make the geometric representation and acceleration structure sufficiently compact by lowering the storage overhead and therefore be more memory efficient [WK06, LYM07]. Similar to the development of many efficient rasterization approaches, we exploit the connectivity between the triangles to reduce storage overhead and even use that connectivity to represent the hierarchy.

Our work is built on the Ray-Strip representation proposed by Lauterbach et al. [LYM07]. In practice, this approach can exploit the inherent ray coherence for traversal of the higher level structure and reduce the storage overhead. In this paper, we introduce ReduceM, a compact representation for large models that offers two main benefits over the Ray-Strip representation: First, Ray-Strips uses triangle strips generated by using a stripification algorithm [ESV96], which was primarily designed for rasterization. We identify constraints for triangle strips optimized for ray tracing and present a new stripification algorithm leading better run-time performance for ray tracing. Second, Ray-Strips encodes the strip as a sequence of vertex indices into a global vertex list and a simple balanced spatial kd-tree structure to allow traversing the strip. ReduceM is more compact and requires less run-time memory footprint. Also, it provides more coherent memory access to the encoded vertices.

We use a two level hierarchy similar to Ray-Strips. However, ReduceM is different from Ray-Strips in many aspects. An example of the ReduceM representation is shown in Fig.2. The ReduceM representation has three main components to store: 1) vertices (e.g., vertex coordinates), 2) the triangle connectivity information, and 3) the hierarchy encoded on the mesh. In order to preserve maximum locality of data access during traversal and intersection, we store all the vertex coordinates directly in the strip without using a global vertex list. Encoding the vertices directly instead of using indirect access through the global vertex list can duplicate the vertices that are shared between different strips, but reduces non-local memory access. As a result, it has a comparable or even better memory usage compared to indexed vertices. This benefit is mainly obtained by eliminating one indirection during ray tracing and preserving locality. Even though vertex coordinates are stored locally in the ReduceM representation, we store the actual triangle strip via vertex indices referring to vertex coordinates in the ReduceM representation. Each vertex index is encoded in one byte, which limits the maximum strip length. In practice useful strips longer than 255 vertices are hard to extract in many real-world models. On the other hand, vertices can often be referenced multiple times in one strip. For example, to create a valid triangle strip ordering from a sequence of triangles, it is often necessary to introduce edge swaps in a triangle strip in order to 'flip' the edge order in the strip by introducing the same edge twice (i.e. for the edge *AB* by having the sequence *ABA*) and hence creating a zero area triangle as well as an additional

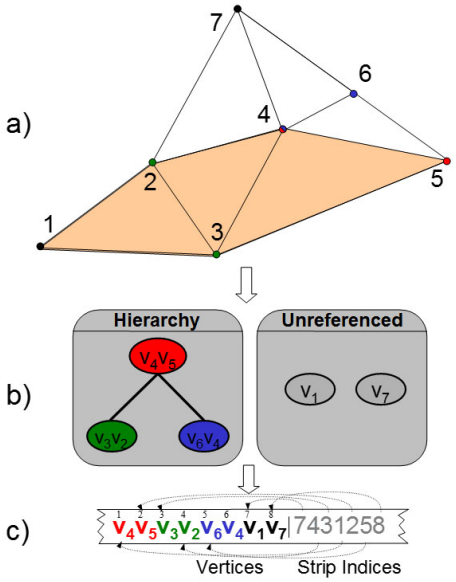

Figure 2: ***ReduceM representation:*** *This figure shows a triangle strip consisting of 7 vertices (see* ***a****). Our ReduceM representation consists of two main components: 1) a list of vertices and 2) a list of vertex indices referring the vertices. We implicitly encode a balanced s-kd-tree structure (see* ***b****), where each split along an axis partitions triangles into halves. By laying out the vertices referenced in the strip carefully, we can encode the bounds of the s-kd-tree (see* ***c****). The actual strip is then defined by indices after the vertices. Vertices not referenced in the hierarchy (here: 1 and 7) are stored afterwards.*

vertex reference. Therefore, encoding the triangle strip with vertex indices of small size will reduce the impact of such swaps in the ReduceM representation.

We use a two-level hierarchy because it allows a compact representation of geometric primitives at the bottom level and an arbitrary choice of acceleration structure at the top. Our two-level hierarchy consists of the strip hierarchies which we implicitly encode on *each* triangle strip. Then, we construct one global high-level hierarchy whose leaf nodes contain the strip hierarchies. We use a balanced spatial kd-tree (s-kd-tree) [OMSD87, WK06] for the strip hierarchy, which is a compact object hierarchy. Each node of the left-balanced spatial kd-tree is represented by the upper and lower bounds for side of the split respectively. Our observation is that the bounds for each split always coincide with one or more vertices on each side of the split. By using vertex indices for encoding the triangle strip, we are free to order the actual vertex coordinates in ReduceM. Therefore, we reorder coordinates of vertices such that every two consecutive vertices represent each split of the s-kd-tree. Based on this formulation, we implicitly define a left-balanced s-kd-tree from the underlying ReduceM representation (see Fig. 2). Each node of the hierarchy also needs to store the split axis for that node, using 2 bits. While it is possible to list this information separately, we store all the vertex coordinates relative to the strip's bounding box known from the traversal, and then encode that information into the sign bits of the vertices. In general, the upper and lower bounds of the strip hierarchy do not reference all the vertices in the strip. Therefore, we need to list all the unused vertices as well (e.g. vertex 1 and 7 in Fig. 2). There are cases where a vertex may need to be used twice in a strip hierarchy (such as vertex 4 in our example). Therefore, there is extra memory overhead from storing vertices twice in the representation. We minimize this overhead by searching for multiple split vertex candidates and always considering previously unused candidates first. We have found that for our test models this method effectively minimizes the overall overhead in our test models (see Table 1.)

Traversal and intersection: The traversal of strip hierarchies is virtually same as that of s-kd-trees [WK06]. We first traverse the high-level hierarchy, which can be any acceleration structure such as a kd-tree or bounding volume hierarchy with individual strips at the leaf nodes. We can also use ray packets to speed up the performance of rendering. Since inherent ray coherence is higher for the high-level structure, our algorithm can use larger packets for traversing the high-level hierarchy. We then switch to use smaller packets for the low-level hierarchy encoded by a triangle strip. Whenever a strip is reached, its hierarchy is traversed until a leaf is found. During the traversal of the s-kd-tree of the strip, we maintain the currently active interval inside the triangle strip. Therefore, intersection with all the triangles in a sub-tree can be performed at any time during the traversal, or at the leafs of the strip hierarchy (which is built down to two triangles per leaf.) The choice can be controlled by an external parameter; which one is faster can be implementation-specific depending on the quality of the hierarchy as well as the performance of the triangle intersection routine. Triangle intersection is performed by testing rays against all the edges that are directly defined by the triangle strip, which is more efficient than testing all triangles since each two triangles share an edge. In the ReduceM representation, the vertices of the strip also define the hierarchy bounds and split axis information. Thus, both intersection between ray and a triangle strip and traversal operations essentially access the same data. Therefore, memory access stays more local during this part of the traversal.

4. Strip-RT: Stripification for Ray Tracing

In this section we present our stripification algorithm, Strip-RT, to construct strips optimized for ray tracing. The goal is to lower the storage overhead and achieve high run-time performance during ray tracing using our representation. The ReduceM representation consists of a triangle strip and a low-level strip hierarchy implicitly encoded in the triangle strip. The high-level hierarchy is computed based on the low-level hierarchy. Therefore, the efficiency of ReduceM basi-

Model	Tris	Build time	Hierarchy overhead
Power plant	12.7M	5m	1.58%
Double Eagle	82M	33m	1.95%
Puget Sound	134M	36m	3.24%
Boeing 777	364M	1h50m	2.71%
St.Matthew	372M	1h36m	3.56%

Table 1: ***Construction statistics:*** *The build time column shows the overall time taken for construction including stripification and high-level hierarchy for our benchmark scenes. The last column shows the overall storage overhead in terms of duplicated vertices as a fraction of total vertices in order to implicitly store the strip hierarchy.*

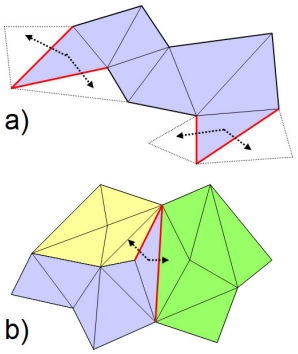

Figure 3: ***Strip ordering:*** *This figure illustrates the strip ordering in the stripification algorithm. a) Each given triangle sequence can have up to four open edges (shown in red) where combination with other sequences are possible. b) The blue triangle sequence has two possible other sequences it can be combined with on its two open edges.*

cally reduces to computation of triangle strips from an input mesh. In order to compute optimized triangle strips for ray tracing, we consider the following two properties:

- **Triangle strip length:** We can achieve a higher compression ratio as the length of the triangle strip increases. As a result, we can further reduce both the storage overhead and the memory overhead of ReduceM.
- **Spatial coherence of the balanced hierarchies:** The balanced s-kd-tree implicitly encoded on top of the triangle strips should have high spatial coherence in order to reduce the number of intersection tests with nodes of the s-kd-tree during ray tracing. It is widely known that we can achieve this goal by considering the surface-area heuristic (SAH) during hierarchy construction [GS87, MB90, Hav00].

Prior algorithms to compute triangle strips – such as Hoppe's rendering sequence [Hop99] – satisfy the first property related to the length of the generated triangle strips. These techniques work well for rasterization where the main goal is to achieve high GPU vertex cache utilization during rasterization. However, these approaches do not address the issue of computing tight fitting hierarchies on top of triangle strips and, therefore, do not address the problem of achieving high spatial coherence between triangle strips for ray tracing.

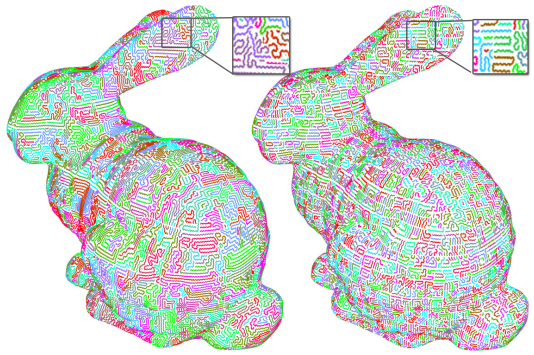

Figure 4: ***Stripification for ray tracing:*** *Triangle strips created for rasterization (such as on the left) are often relatively wide-spread and therefore have a high surface area compared to their size. Our stripification (on the right) creates strips that are both compact in terms of surface area and have sub-strips that are themselves compact. This creates higher traversal efficiency for ray tracing.*

4.1. Hierarchical Triangle Strip Computation

Most acceleration hierarchy construction methods for ray tracing use top-down methods while considering the SAH metric at each split. Since we implicitly encode our s-kd-tree from the constructed triangle strip, we design our triangle stripification method, Strip-RT, to construct triangle strips considering the SAH. Overall, the triangle strip computation algorithm is given an input mesh and produces one or more triangle strips from it using the following steps:

1. **Adjacency computation:** Usually the input mesh is not given as a graph with adjacency information but a list of unordered triangles. We find all shared vertices and edges between triangles and then record all the adjacency information in the mesh.
2. **Partitioning:** Given the mesh, the partitioning step recursively splits it into an initial object hierarchy optimized by the SAH metric.
3. **Ordering:** The ordering step takes the initial hierarchy and iteratively computes one or multiple triangle sequences optimized for ray tracing.
4. **Strip output:** Finally, using each sequence of triangles the strip output step produces a list of indices defining each strip in the ReduceM format.

Of these, 2 and 3 are the main steps in the algorithm and will thus be described in detail in the following text. Computing adjacency information in step 1 is relatively simple as long as the triangles are specified with vertex indices; otherwise, duplicate vertices have to be detected by analyzing the vertex coordinates. Computing a list of indices specifying a triangle strip from a sequence of triangles has also been discussed in depth in previous work (e.g. [ESV96]) and we opt for a similar implementation. Note that during the partitioning and ordering steps, our algorithm may not produce just one connected triangle strip from the input mesh. In gen-

eral,computing a single connected triangle strip of a mesh is equivalent to computing a Hamiltonian path, which is a NP-complete problem [Dil92], and even if such a strip exists, it might not be a good choice for ray tracing applications.

Partitioning: We recursively partition the mesh contained in a node of the hierarchy into two sub-meshes, starting with the input mesh and ending when only one triangle is left. During each partition, we find the best split of the triangles in the mesh by using the SAH metric and choosing the partition with the lowest cost. Thus, the partitioning is almost the same as building an object hierarchy such as a BVH on the input mesh (e.g. as described in [WBS06]). In addition to generating the hierarchy, we also store an inverse mapping for each triangle that records which hierarchy node it is referenced in (and since this is an object hierarchy, only one node can do so.)

Ordering: The purpose of the ordering step is to find actual triangle sequences in the input mesh that can later on be converted to triangle strips. In order to find good strips for ray tracing, the previously computed hierarchy is used to provide hints on determining good pairings of sequences. Our ordering algorithm iteratively combines existing triangle sequences into longer sequences. As a first step, we consider all triangles as a triangle strip of length one and place them into a list of active sequences. For each sequence, we record the list of triangles that defines the sequence, as well as the adjacency information: since each active sequence has two potentially 'open' edges at the end (see Fig. 3 for illustration), 4 pointers to the adjacent triangle at those edges (or to a null object if not a shared edge) are sufficient. In addition, we also maintain a reference to a node in the hierarchy built in the previous step, initialized to leaf nodes for the initial sequences. Finally, we maintain the list of triangles in the mesh such that each triangle also has a pointer to the sequence that it is currently contained in, which allows us to find the neighboring sequences based on the triangle adjacency information.

At each iteration in the algorithm, we look at each sequence in the list of active sequences and then evaluate all possible pairings with other active sequences using the adjacency information. All sequences that have no further possible combinations are placed in the output sequence list and removed from the active list. For many sequences, however, there will be multiple possibilities for combination and we want to find the one that will most likely produce strips that are best for our purposes. To do so, we rank all possible combinations with other strips according to two factors: first, since each sequence is associated with a node in the initial hierarchy, we test the path distance in the tree between the two nodes and prefer the combination that has the shortest one. Intuitively, this rewards combining strips that would have been split similarly in the optimized build and thus presumably are a good choice. However, we also use a second criterion by considering the *harmonic mean* between the number of triangles in each of the two sequences. The reason for this is that we later use a balanced hierarchy on the ReduceM representation and thus do not have a choice in where to put the split inside the strip. By choosing combinations of roughly equal relative size, it is very likely that the implicit hierarchy will have a split that corresponds to the two strips we are combining. For example, assume that we have two sequences with 130 and 10 triangles and another pair with 70 and 70 pairs. Although the arithmetic mean (i.e. 70) is the same, the pair with 130 and 10 connectivity pairs has a harmonic mean of 18.5 whereas the other is 70. Overall, when determining the best combination we weigh both the relative tree distance as well as the harmonic mean (relative to the total number of triangles in the sequences) equally and then pick the combination with the best results.

To merge triangle sequences, we concatenate the triangle index list of both sequences and then can easily find the adjacency information for the new strip. However, we also need to find a tree node associated with the new strip. To do so, we find the nodes associated with the two individual sequences and then assign the lowest common ancestor to the new strip. The algorithm terminates when at the end of an iteration there are no more active sequences left, i.e. no more possible combinations of sequences available. However, we have also found that it can be useful to introduce a user-specified maximum strip length parameter (please also see the discussion in section 6) that prevents sequences from growing above a certain size. The reason for doing so is that the algorithm may otherwise produce some very long strips while at the same time keeping others very short. We have found that it is more desirable to keep sequence lengths more uniform for performance.

4.2. Massive model stripification

In order to apply our algorithm to massive models and find smaller input meshes for the stripification algorithm, we first decompose an input model into small chunks, each of which fits into main memory. Then, we apply our algorithm to the chunks, each of which can also be processed in parallel on multiple processors to speed up the stripification process, with a merge operation at the end. Table 1 shows information on typical construction times for our benchmark models, using parallel processing on the benchmark system given in section 5.

5. Results and Analysis

We implemented the ReduceM algorithm in a full ray tracing system running on a Intel Xeon machine with 16 cores (X7350 at 2.93 GHz) and 16 GB of RAM running on 64-bit Microsoft Windows. We have used different models with various characteristics for our benchmarks. These data sets are selected to provide examples from different application areas including scientific visualization, terrain visualization, CAD and architectural environments, and scanned models. Parameters were set such that triangle strip length was lim-

Model	Tris	Vertices	ReduceM			kd-tree			BVH	
			Total	Vertices	Hierarchy+Tris	Total	Tree	Geometry	Total	Tree
Power plant	12.7M	11M	272	126	171	983	496	486	875	389
Double Eagle	82M	77M	1818	884	1047	7118	3373	3147	6241	2496
St.Matthew	128M	64M	2520	736	1783	12844	7961	4882	8789	3906
Puget Sound	134M	215	2834	768	1961	11164	6044	5120	9216	4095
St.Matthew	372M	186M	7290	2138	4985	n/a	n/a	13921	25595	11375
Boeing 777	360M	208M	8914	2389	8461	n/a	n/a	14219	25059	11137

Table 3: ***Memory footprint:*** *We show the effect on memory consumption of using ReduceM compared to kd-trees and BVHs for several complex models (all numbers in MB). The memory used by ReduceM can be split up in the uncompressed vertex data as well as the high-level and strip hierarchy representation including the connectivity. The BVH and kd-tree have the same storage for the triangle geometry, but different footprint for the hierarchy.*

Model	Tris	Memory total		Performance (fps)	
		Ray-Strips	ReduceM	Ray-Strips	ReduceM
Isosurface	10M	220 MB	201 MB	4.17	6.56
Power plant	12.7M	395 MB	272 MB	1.26	4.98
Lucy	30M	581 MB	568 MB	2.05	3.15

Model	Tris	Memory total		Performance (fps)	
		STRIPE	ReduceM	STRIPE	ReduceM
Power plant	12.7M	350 MB	272 MB	28.57	31.25
Double Eagle	82M	2672 MB	1818 MB	11.90	12.50
Puget Sound	134M	2736 MB	2834 MB	7.58	12.05
Boeing 777	364M	12128 MB	8914 MB	3.75	4.76
St.Matthew	372M	7531 MB	7290 MB	3.36	4.81

Table 2: ***Comparison to Ray-Strips:*** *The first table directly compares the results on the largest models as used in [LYM07] using* 2×2 *packets to an equivalent result with the ReduceM representation and stripification (*512^2*, single core only to allow clear comparison). The second table singles out the effect of stripification on performance and memory for the same ReduceM representation, performance numbers are for primary visibility at* 1024^2 *on the system described in section 5. Stripification is performed with STRIPE [ESV96] as used in Ray-Strips [LYM07] and then with our Strip-RT algorithm.*

ited to at most 60 triangles and strip hierarchy traversal was set to terminate when reaching 2 triangles.

We measure the improvements both in terms of overall memory footprint as well as the reduction in hierarchy and connectivity size alone. Table 3 summarizes the results for our benchmarks. We also compare the memory and rendering speed to a pure kd-tree [Wal04] and AABB bounding volume hierarchy implementation [LYTM06, WBS06], both of which are popular solutions in interactive ray tracing. The kd-tree implementation uses a standard 8 byte/node representation [WDS04], while each BVH node takes 28 bytes. All the hierarchies were built using the surface area heuristic with automatic termination criterion. Also, performance results were obtained using ray packet traversal and intersection. Note that performance results for these comparisons are not available for some of the larger models since the total size exceeded main memory size and performance results would therefore have been very low. The triangle geometry was stored in a standard intersection format [WDS04] using 40 bytes per triangle.

We also compare the memory requirement and runtime performance of our ReduceM method over Ray-Strips [LYM07] (see Table 2) and are able to achieve up to 3.9 times performance improvement and 45% memory reduction. In order to verify benefits of our Strip-RT method, we measure memory requirement and runtime performance of our ReduceM method when using triangle strips computed from Strip-RT and STRIPE [ESV96] that the Ray-Strip method used (see Table 2). Strip-RT achieves up to 58% runtime performance improvement over STRIPE.

6. Analysis and Comparison

Any mesh compression for ray tracing can reduce three different parts: vertex data, connectivity and hierarchy. The results from the ReduceM algorithm clearly show that we are successful in reducing the memory footprint of both connectivity and hierarchy, but even with the optimized stripification we still incur some rendering overhead compared to a fully optimized standard hierarchy. In particular in architectural and CAD scenes with strongly varying triangle sizes there still is a higher performance difference, while other models are very close. However, in the common situation where memory is in fact limited, being able to represent all ray tracing data in main memory is invaluable.

As discussed in section 4.1, it can be useful to limit the maximum triangle size during construction to avoid unevenly distributed strip sizes. In general, we find that due to the balanced split limitation even our optimized stripification algorithm cannot guarantee the same hierarchy quality as for example a common SAH hierarchy, and thus performance decreases slightly at some point as triangle strip length restrictions are lowered. Figure 5 visualizes this at the example of the Lucy model where the ReduceM representation was generated using different length limits ranging from 4 to 64 — note that this does not mean that all the strips are that size; in fact, the average strip length is usually much lower.

Comparison: Approaches to compress the hierarchy used for ray tracing were introduced in [Mah05, CSE06]. By reducing the number of bits used to encode the bounding boxes, the memory complexity can be reduced drastically. However, there are two main implications. First, decoding of

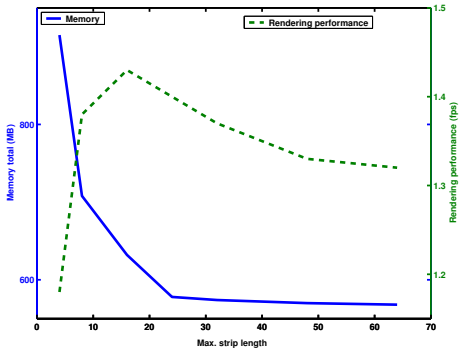

Figure 5: ***Effect of strip length parameter:*** *Results for RayStrips on the Lucy model when modifying the maximum strip length parameter in construction. Both performance as well as memory footprint improve with increasing strip length. However, at higher limits memory savings decrease as only some longer strips are found and performance decreases slightly.*

the coordinates has to occur at run-time and adds some overhead to traversal, although Mahovsky [Mah05] shows that the effect can be limited by using ray coherence approaches. Second, quantizing bounding boxes inherently slows down ray tracing since it enlarges the surface area of the boxes and thus increases the average number of traversal steps per ray. Reshetov has recently introduced a vertex culling method [Res07] that can reduce memory requirement of hierarchies. The vertex culling algorithm can efficiently intersect with relatively large kd-tree leaf nodes (i.e. with a much higher number of triangles than the usual 2-4 triangles per a node) and thus the overall hierarchy is smaller since fewer nodes are required. Geometry is still stored as usual. Since the paper was not focused on massive models, it is hard to directly compare both approaches due to lack of data. However, it is possible that vertex culling could be combined with the ReduceM approach as an alternative strip intersection method.

Limitations: Our method has certain limitations. First, ReduceM reduces memory requirement only for connectivity and hierarchy information, but not vertices. This puts a hard limit on the achievable overall compression since for massive models vertex data can make up a large part of the overall storage. Compression is also strongly dependent on available connectivity, i.e. if the input model is a fully disconnected set of triangles, no improvement is possible. For example, we have found connectivity to be a problem on the Boeing 777 model: vertex coordinates on the models were slightly perturbed beforedistribution for security reasons, which breaks some of the connectivity that would have otherwise been available. We assume that this is the main reason that the stripification algorithm finds only significantly shorter strips on this model. Finally, the stripification algorithm for ReduceM uses a greedy heuristic, which

Model	Tris	Vert.	ReduceM fps	kd-tree fps
Power plant	12.7M	11M	30.30	27.78
Double Eagle	82M	77M	4.61	5.95
Puget Sound	134M	67M	12.05	12.48
Boeing 777	364M	208M	4.76	n/a (OOM)
St.Matthew	372M	186M	4.81	n/a (OOM)

Table 4: ***Results: Rendering performance.*** *Performance results as average fps at* 1024^2 *pixels using* 2×2 *ray packets for primary visibility, compared to an optimized kd-tree implementation. Note that we do not report some numbers for the kd-treesince the dataset is out of core and thus fair comparison is impossible.*

does not guarantee to produce good results in all cases since it does not optimize globally. In addition, many CAD models usually have a limit on how long strips can potentially be for any algorithm, which limits the impact of ReduceM.

7. Conclusion

We have presented ReduceM, a novel representation for massive models that allows interactive ray tracing due to a compact memory footprint and localized data access. We also have proposed Strip-RT as an algorithm to compute triangle strips optimized for ray tracing. We have demonstrated the performance of our algorithm on several widely-used large models and showed the benefits compared to previous work. There are many avenues for future work. One major application for compact ray tracing representations is GPU-based ray tracing since memory is severely limited and being able to fit all data into GPU memory is paramount. We also believe that we can further improve the performance of the stripification algorithm by applying results from research in single-strip triangulation that change the mesh in order to provide longer strips. One of the remaining bottlenecks may increasingly become storage of vertices, especially when also using vertex normals or texture coordinates. It may be useful to extend ray tracing representations to also compactly store this data. Finally, we would like to investigate our approach for dynamic scenes.

Acknowledgments

The Lucy and St.Matthew models are courtesy of the 3D scanning repository at Stanford University. We would like to thank Dave Kasik at Boeing for providing the 777 model, and Peter Lindstrom at LLNL for the Puget sound models as well as simplified versions of the St.Matthew model. This work was supported in part by ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0400134 and 0118743, ONR Contract N00014-01-1-0496, DARPA/RDECOM Contract N61339-04-C-0043, Intel, a KAIST seed grant, and the IT R&D programs of MKE/IITA [2008-F-033-01] and “Development of Elemental Technology for Promoting Digital Textbook and u-Learning".

References

[AC97] AMANATIDES J., CHOI K.: Ray tracing triangular meshes. In *Proceedings of the Eighth Western Computer*

Graphics Symposium (1997), pp. 43–52. 2

[CLF*03] CHRISTENSEN P. H., LAUR D. M., FONG J., WOOTEN W. L., BATALI D.: Ray differentials and multiresolution geometry caching for distribution ray tracing in complex scenes. *j-CGF 22*, 3 (Sept. 2003), 543–552. 2

[CSE06] CLINE D., STEELE K., EGBERT P. K.: Lightweight bounding volumes for ray tracing. *Journal of Graphics Tools: JGT (to appear)* (2006). 2, 7

[Dee95] DEERING M. F.: Geometry compression. In *ACM SIGGRAPH* (1995), pp. 13–20. 2

[DGP04] DEMARLE D. E., GRIBBLE C. P., PARKER S. G.: Memory-savvy distributed interactive ray tracing. In *EGPGV* (2004), pp. 93–100. 2, 3

[Dil92] DILLENCOURT M.: Finding hamiltonian cycles in delaunay triangulartions is np-complete. *Canadian Conference on Computational Geometry* (1992). 6

[DSW07] DIETRICH A., STEPHENS A., WALD I.: Exploring a boeing 777: Ray tracing large-scale cad data. *IEEE Computer Graphics and Applications 27*, 6 (2007), 36–46. 2

[EMAM07] ERIK MANSSON J. M., AKENINE-MOLLER T.: Deep coherent ray tracing. *Symp. on Interactive Ray Tracing* (2007). 2, 3

[ESV96] EVANS F., SKIENA S. S., VARSHNEY A.: Optimizing triangle strips for fast rendering. In *IEEE Visualization* (1996), pp. 319–326. 3, 5, 7

[GS87] GOLDSMITH J., SALMON J.: Automatic creation of object hierarchies for ray tracing. *IEEE Comput. Graph. Appl. 7*, 5 (1987), 14–20. 2, 5

[Hav00] HAVRAN V.: *Heuristic Ray Shooting Algorithms*. PhD thesis, Czech Technical University in Prague, November 2000. 2, 5

[Hop99] HOPPE H.: Optimization of mesh locality for transparent vertex caching. *Proc. of ACM SIGGRAPH* (1999), 269–276. 2, 5

[KDS98] KOBBELT L. P., DAUBERT K., SEIDEL H.-P.: Ray tracing of subdivision surfaces. In *Proc. of the Eurographics Workshop* (1998), pp. 69–80. 2

[LYM07] LAUTERBACH C., YOON S., MANOCHA D.: Ray-strips: A compact mesh representation for interactive ray tracing. *IEEE Symposium on Interactive Ray Tracing* (2007). 2, 3, 7

[LYTM06] LAUTERBACH C., YOON S., TUFT D., MANOCHA D.: RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs. *IEEE Symposium on Interactive Ray Tracing* (2006). 7

[Mah05] MAHOVSKY J.: *Ray Tracing with Reduced-Precision Bounding Volume Hierarchies*. PhD thesis, University of Calgary, September 2005. 2, 7, 8

[MB90] MACDONALD J. D., BOOTH K. S.: Heuristics for ray tracing using space subdivision. *Visual Computer* (1990). 2, 5

[MTF03] MÜLLER K., TECHMANN T., FELLNER D. W.: Adaptive ray tracing of subdivision surfaces. *Comput. Graph. Forum 22*, 3 (2003), 553–562. 2

[OMSD87] OOI B., MCDONELL K., SACKS-DAVIS R.: Spatial kd-tree: An indexing mechanism for spatial databases. *Proc. of the IEEE COMPSAC Conf.* (1987). 4

[PKGH97] PHARR M., KOLB C., GERSHBEIN R., HANRAHAN P.: Rendering complex scenes with memory-coherent ray tracing. In *Proc. of ACM SIGGRAPH* (1997), pp. 101–108. 2, 3

[Res07] RESHETOV A.: Faster ray packets - triangle intersection through vertex culling. *Symp. on Interactive Ray Tracing* (2007). 2, 8

[RSH05] RESHETOV A., SOUPIKOV A., HURLEY J.: Multi-level ray tracing algorithm. *ACM Trans. Graph. 24*, 3 (2005), 1176–1185. 2, 3

[SBB*06] STEPHENS A., BOULOS S., BIGLER J., WALD I., PARKER S.: An application of scalable massive model interaction using shared memory systems. *Proc. of Eurographics Symp. on Parallel Graphics and Visualization* (2006), 19–26. 2

[SMD*06] STOLL G., MARK W. R., DJEU P., WANG R., ELHASSAN I.: *Razor: An Architecture for Dynamic Multiresolution Ray Tracing*. Tech. Rep. TR-06-21, Dept. of CS, Univ. of Texas, 2006. 2

[Wal04] WALD I.: *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics Group, Saarland University, 2004. 2, 3, 7

[WBS06] WALD I., BOULOS S., SHIRLEY P.: Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics* (2006). 6, 7

[WDS04] WALD I., DIETRICH A., SLUSALLEK P.: An Interactive Out-of-Core Rendering Framework for Visualizing Massively Complex Models. In *Proceedings of the Eurographics Symposium on Rendering* (2004). 2, 7

[WK06] WÄCHTER C., KELLER A.: Instant Ray Tracing: The Bounding Interval Hierarchy. In *Rendering Techniques 2006: Eurographics Symposium on Rendering*. (2006). 2, 3, 4

[YLM06] YOON S., LAUTERBACH C., MANOCHA D.: R-LODs: Interactive LOD-based Ray Tracing of Massive Models. *The Visual Computer (Pacific Graphics)* (2006). 2

[YLPM05] YOON S.-E., LINDSTROM P., PASCUCCI V., MANOCHA D.: Cache-Oblivious Mesh Layouts. *Proc. of ACM SIGGRAPH* (2005). 2