A Compressed Representation for Ray Tracing Parametric Surfaces

KAI SELGRAD, ALEXANDER LIER and MAGDALENA MARTINEK
Computer Graphics Group, University of Erlangen-Nuremberg
and
CHRISTOPH BUCHENAU and MICHAEL GUTHE
Visual Computing Group, University of Bayreuth
and
FRANZISKA KRANZ, HENRY SCHÄFER and MARC STAMMINGER
Computer Graphics Group, University of Erlangen-Nuremberg

Parametric surfaces are an essential modeling tool in computer aided design and movie production. Even though their use is well established in industry, generating ray-traced images adds significant cost in time and memory consumption. Ray tracing such surfaces is usually accomplished by subdividing the surfaces on-the-fly, or by conversion to a polygonal representation. However, on-the-fly subdivision is computationally very expensive, whereas polygonal meshes require large amounts of memory. This is a particular problem for parametric surfaces with displacement, where very fine tessellation is required to faithfully represent the shape. Hence, memory restrictions are the major challenge in production rendering. In this paper, we present a novel solution to this problem. We propose a compression scheme for a-priori Bounding Volume Hierarchies (BVHs) on parametric patches, that reduces the data required for the hierarchy by a factor of up to 48. We further propose an approximate evaluation method that does not require leaf geometry, yielding an overall reduction of memory consumption by a factor of 60 over regular BVHs on indexed face sets and by a factor of 16 over established state-of-the-art compression schemes. Alternatively, our compression can simply be applied to a standard BVH while keeping the leaf geometry, resulting in a compression rate of up to 2:1 over current methods. Although decompression generates additional costs during traversal, we can manage very complex scenes even on the memory restrictive GPU at competitive render times.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Raytracing*

General Terms: Performance, Algorithms

DOI 10.1145/10.1145/2953877
http://doi.acm.org/10.1145/10.1145/2953877

Additional Key Words and Phrases: Production Rendering, Subdivision Surfaces, Displacement Mapping, Ray Tracing

1. INTRODUCTION

Parametric surfaces are *the* standard modeling primitive in high-end graphics applications. CAD-systems typically use NURBS-surfaces to model smooth objects, whereas feature film production tools are mostly based on subdivision surfaces with detailed displacement maps. Nevertheless, generating high-quality ray-traced images of such objects is still challenging, and with the movie industry migrating to path tracing [Christensen et al. 2006] ever more relevant. Though it is possible to directly intersect these surfaces using iterative approaches, such methods are prone to numerical problems and even if these problems are solved, they cannot manage displacement maps.

Hence, for production rendering the surfaces are subdivided very finely and the resulting triangles or quads are stored in an acceleration structure (such as a BVH or a kd-tree) to improve ray traversal performance. This is suitable as long as the generated hierarchy, as well as the geometry itself, fit into main memory. However, the resulting meshes are usually very large and the memory limit is quickly reached even for scenes of moderate size. The limit can be pushed by adaptive and on-the-fly subdivision or by swapping data on demand. However, the resulting performance loss and implementation overhead is considerable. Furthermore, the memory restrictions demand a careful, and often manual, choice of the tessellation levels as even slight over-tessellation results in strongly increased memory consumption.

We propose a novel approach for compressing BVHs that allows us to push the memory limit by up to a factor of 60 over a regular representation as tight indexed face set, and up to a factor of 16 over state-of-the-art representations. Our compression scheme is tailored to storing BVHs of *parametric* patches, possibly with displacement maps. The hierarchy is implicitly generated by building a full quadtree in the parameter domain of each surface. Thereby, we avoid indexing overhead for accessing child nodes in the BVH and enable obtaining each node's interval in parameter space solely from its index in the hierarchy. This reduces the amount of required data per BVH node to its bounding box, only.

Our compression scheme (Section 3) is based on a two level BVH, where the upper level, containing much fewer nodes compared to the lower levels, is a standard BVH. All parametric patches

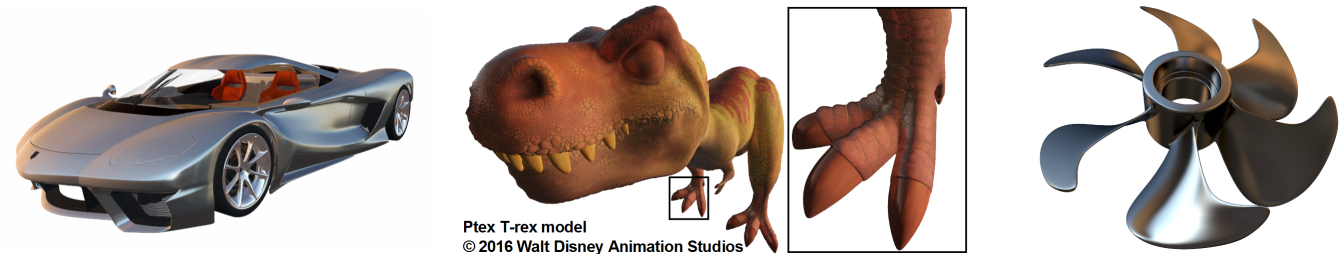


Fig. 1. Our novel compression method reduces the memory requirements for ray tracing parametric surfaces by 60 over regular BVHs on indexed face sets and by 16 over state-of-the-art compressed structures while maintaining reasonable rendering performance. It thereby allows to render much larger scenes in-core or on the GPU than previous approaches. We successfully applied our method to highly detailed production-grade subdivision models with sharp creases (left) and displacement (center) as well as to NURBS models (right). All models shown were ray traced at subdivision level 8 on the GPU.

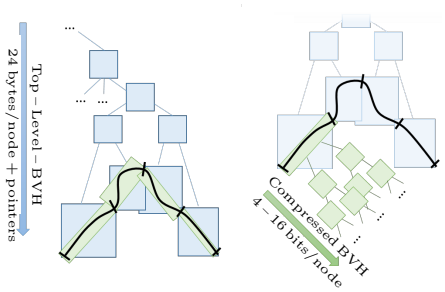


Fig. 2. To construct our hierarchy, we first adaptively subdivide all patches until they fulfill a flatness criterion, and then sort them into a global hierarchy with full precision. For all these flat patches we compute an aligned local coordinate frame and switch to our lightweight quantized and compressed hierarchy.

are inserted into the top level BVH and subdivided therein. As soon as a certain flatness criterion is reached, we switch to our highly compressed BVH. We align the root of our compressed BVH with the flat patch, and store the bounds of each node relative to its parent. As a result, bound values are clustered and can be well predicted, which allows us to use strong quantization as well as compression, exploiting the properties of the parent/child box relations. Figure 2 depicts this two-level hierarchy.

In this paper, we propose and examine different quantization and compression strategies reducing the memory footprint to 8 or 16 bit per node and, with even more aggressive compression, down to 4 bit per node. Compared to a representation with 6 floats (i.e. 24 bytes per node), this corresponds to a compression factor of 12 up to 48 for the bounding geometry.

Such high compression rates allow us to represent surfaces very finely, so that we can approximate the parameter values of the hit-point directly on the BVH-data. That is, only the BVH-bounds are used to intersect a ray with the scene and surface geometry is discarded. Compared to other approaches that store surface geometry in the leaves, this improves the compression rate even further.

Our approach integrates nicely into a production environment, where ray tracing and shading are performed separately (see Eisenacher et al. [2013] and Laine et al. [2013]). A query to our compressed hierarchy returns a patch index and parameter space coordinates per hit-point and from this data, hit position, normals etc. are then determined during shading (as described by Eisenacher et al. [2013]). Details on this integration are given in Section 4.

Naturally, decompression comes at a cost, but our results (see Section 5) show that our method often performs better than competing compression approaches. Due to the fine subdivision, the method generates very good results, even if the intersection points are approximate. To visualize the precision of our method, Figure 3 shows a close-up of the T-rex model and a two-level zoom into a visualization of the compressed hierarchy. Note that the real hierarchy is three levels deeper than the visualized boxes. However, if approximate hit-points cannot be tolerated in certain cases our compressed hierarchy can still be applied while keeping the leaf geometry. Memory savings are reduced in this scenario, but still significant and furthermore accompanied by performance gains.

The specific contributions of our work are as follows:

— A novel method to quantize the bounds of subdivided patches based on the orientation of locally flat and aligned sub-patches, which achieves a peak compression rate of the bounding geometry of 12 over full-precision bounding boxes.

— Different approaches for further compression of the quantized bounds, increasing the compression rate of the bounding geometry from 12 for quantized boxes to 24 up to 48.

— An approximate evaluation scheme that integrates nicely with how production renderers work and that allows to trace without geometric primitives beside compressed bounding volumes, which adds to the previous compression rates by not having to keep leaf primitives.

— An exact solution that still provides memory savings of up to a factor of 2. Overall compression is reduced when compared to the approximate solution as leaf geometry cannot be dropped. This scheme still provides memory and performance gains, and is compatible to state-of-the-art traversal of parametric patches.

2. RELATED WORK

Parametric surfaces are an essential tool for modeling smooth continuous shapes in production rendering and CAD systems. Especially Non-Uniform Rational B-splines (NURBS) and subdivision surfaces are widely used in practice. Catmull-Clark subdivision [Catmull and Clark 1978] is a generalization of bi-cubic B-spline subdivision to irregular control meshes that freed designers from topology constraints. By repeating a set of simple subdivision rules, the input mesh converges to a smooth limit surface. Since the approach was extended to support boundaries [Hoppe et al. 1994] and creases [DeRose et al. 1998], subdivision surfaces with displacement textures became the standard representation in movie production, whereas NURBS are the standard in CAD applications.

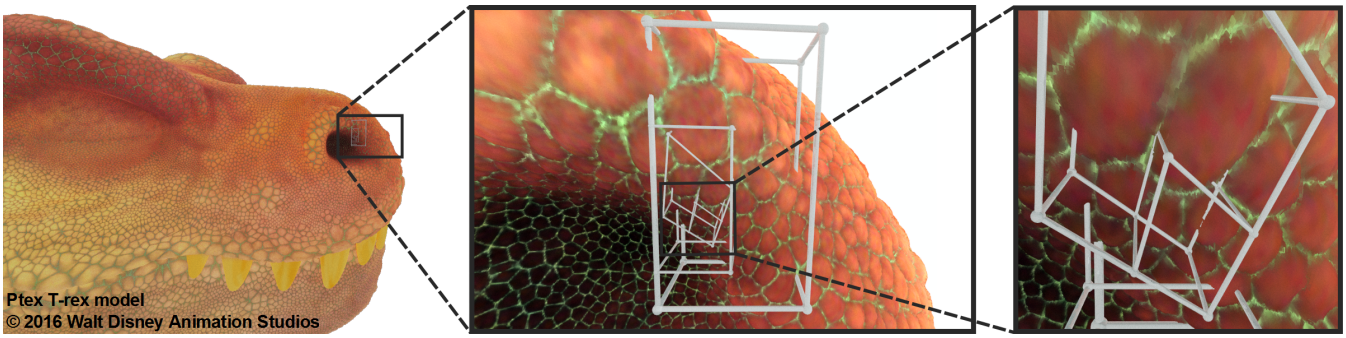


Fig. 3. A close-up on the T-rex's nose illustrates how fine our hierarchy is built, and how well it adapts to the local geometry. For better presentation we only display bounding boxs down to subdivision level 3, whereas the hierarchy used for rendering is subdivided to level 6. Also note the high accuracy at the silhouette in the center panel.

Direct Ray Tracing of Parametric Surfaces. Early approaches for computing ray intersections with parametric surfaces are based on numeric techniques such as algebraic surfaces [Kajiya 1982] or Newton iteration with interval analysis [Toth 1985].

These computationally expensive methods were improved by providing good starting positions for evaluating the surfaces using Bézier-clipping [Nishita et al. 1990] and bounding volume hierarchies. A combination of an axis aligned bounding box hierarchy with Newton iteration for intersecting Bézier-patches was proposed by Geimer and Abert [2005] and later extended to NURBS surfaces [Abert et al. 2006]. Convex hulls as bounding volumes are combined with Newton iteration in one of the first GPU ray tracing algorithms for NURBS surfaces [Pabst et al. 2006]. Tejima et al. [2015] apply feature adaptive subdivision [Nießner et al. 2012] followed by Bézier-clipping for direct ray tracing on the CPU. Alternative approaches subdivide the surfaces on-the-fly to generate sub-patches that approach the actual surfaces [Rockwood et al. 1989; Benthin et al. 2007]. We compare our results to a recent implementation using a shared lazy build cache [Benthin et al. 2015].

All these direct ray tracing approaches deliver high quality solutions and require little memory. However, the performance of these approaches is significantly below that of polygonal representations, as long as the resulting polygons fit into memory [Benthin et al. 2007; Segovia and Ernst 2010]. Furthermore, all these direct approaches cannot handle displacement maps, and are thus not applicable to most scenes used in movie production. In addition, efficient Bézier-clipping requires tight ray differentials, which are difficult to obtain for rays from light sources.

Polygonal Representations. An alternative to directly computing the intersection with parametric surfaces is the rendering of a polygonal approximation of the surfaces. A very successful example of this approach in production rendering is the Reyes rendering pipeline [Cook et al. 1987]. Parametric surfaces are subdivided and finally split into micro-polygons that are then shaded to produce the final image. This works particularly well for subdivision surfaces with displacement mapping, which are difficult to handle with direct evaluation methods.The introduction of the hardware tessellation unit enabled efficiently mapping the Reyes split stage and the evaluation of the limit surface to the graphics hardware [Nießner et al. 2012; Schäfer et al. 2014]. While these approaches work well for rasterization, rendering with global illumination requires accessing all scene geometry and there is a strong demand in industry to combine the Reyes pipeline with ray tracing [Christensen et al. 2006; Eisenacher et al. 2013]. Therefore, all parametric surfaces in a scene are typically subdivided and converted into dense polygonal meshes that represent the original surface, possibly with displacement. Although the resulting meshes can be efficiently rendered using acceleration structures, the memory requirements for storing micro-polygons and bounding nodes are quickly expanding beyond the amount of available main memory. This necessitates out-of-core techniques [Pharr et al. 1997] as well as caching [Wald et al. 2007; Benthin et al. 2015], LOD [Christensen et al. 2003; Yoon et al. 2006] and compression schemes to handle the amount of data.

BVH Compression. Closely related to our work is the hierarchical quantization schemes on point clouds on bounding spheres [Rusinkiewicz and Levoy 2000] and kd-trees [Hubo et al. 2006], where child nodes are quantized locally in the coordinate system of the parent nodes.

As shown by Yoon et al. [2006] as well as by Christensen et al. [2006] ray tracing performance drops by at least two orders of magnitude due to swapping when the scene does not fit into memory any more. Yoon et al. [2006] propose to reduce bandwidth by stopping BVH traversal as soon as the ray diameter is reached.

Mahovsky [2005] proposed a reduced precision integer BVH with a compression rate of 3:1 to 4:1. He used an 8 or 4 bit linear quantization of the coordinates relative to the parent bounding box. Segovia and Ernst [2010] use quantization for both vertex positions and bounding volumes resulting in a total compression rate of 4:1 to 8:1.

Our concept of using a two-level hierarchy has also been previously proposed by Lauterbach et al. [Lauterbach et al. 2008] where a kd-tree is built on top of a triangle strip hierarchy. In total they achieve a compression rate of 5:1 to 6:1 on arbitrary triangle meshes. Embree [Wald et al. 2014] provides an implementation of this scheme that is tailored to subdivision surfaces with a compression rate of up to 16:1. We compare our approach to this method in Section 5. Kim et al. [2010] cluster BVH nodes and use arithmetic compression for each cluster. While they achieve a moderate compression rate of 10:1, they always need to decompress the whole cluster even if only a single node is traversed by the ray. The HC-CMesh algorithm [Kim et al. 2010] encodes bounding boxes using six extremal vertices and achieves a compression rate of 8:1 (4 bytes per node). The minimal BVH [Bauszat et al. 2010] also uses a two-level hierarchy, where the lower nodes are compressed to two bits storing if the two boundary planes in split direction are reduced by a constant factor and always splitting a node in its largest extent. While the compression rate is similar to our approach, the constant

factor reduction leads to over-sized bounding volumes that significantly degrade performance.

All of these approaches use quantization to reduce memory consumption, and apply further ideas to compress bounding values, indices, or mix BVH and triangle data to achieve better efficiency. However, most of these approaches have a dramatic impact on performance, and/or can only achieve moderate compression rates. The reason is probably that all these approaches compress the entire BVH of a scene, which is much harder than concentrating to flat, aligned patches as we do. Furthermore in a single BVH for an entire scene, one cannot rely on a full hierarchy (as we do), which makes it necessary to take care of indices.

Approximate Methods. Novák and Dachsbacher [2012] replace BVH sub-trees by rasterized height fields. These RBVHs are similar to our approach in the sense that they switch to an alternative surface representation as soon as geometry-clusters get sufficiently flat. The main difference in memory consumption is that our hierarchy *implicitly* stores parameter values, whereas the height fields of RBVHs are a resampled surface representation. Any surface attributes (such as texture coordinates, but also normals etc.) need to be stored with this representation, see Section 5 for a comparison to our method.

Another related approach are GigaVoxels [Crassin 2011]. Using this method, scenes are represented as voxel grids and any original geometry is discarded. The voxel grid is stored hierarchically, which allows efficient cone tracing with impressive performance. Our approach is comparable in that it also relies on tiny, but approximate, volumes at leaf level with our approximate version. However, our hierarchy is not bound to a global grid, but spans along the geometry, very much like object oriented BVHs versus octrees. GigaVoxels integrate level-of-detail and caching easily, whereas our approach is tailored towards higher quality and to be more robust under animation.

3. PATCH-ALIGNED BVH COMPRESSION

In the following we describe the motivation behind our BVH representation and how its structural properties are exploited to achieve compression rates beyond previous methods. Our algorithm builds this structure from a set of input patches which, in this paper, are Catmull-Clark subdivision surfaces with displacement textures and NURBS-patches. These are the most important patch types in practice, however, our algorithm is by no means limited to these.

3.1 Two-Level Hierarchy

Our construction starts by generating a standard, full-precision BVH and inserting all parametric patches as leaves. It then applies top-down subdivision of the input primitives on a patch-by-patch basis, thereby extending the full-precision BVH.

The key insight we use is that, as soon as a patch is even moderately flat (see Section 3.2) and well aligned (see Section 3.3), further subdivision yields surfaces with predictable bounding boxes. We exploit this by using a local coordinate frame that is aligned by a projection to the patch such that the frame's origin maps to one corner of the patch, and the basis vectors span the patch's extent along (u, v), i.e. we evaluate the limit positions at the corners and use the two edge vectors in positive u and v direction. In this frame, and under the assumption of flatness, subdividing the patch at, e.g., $u = \frac{1}{2}$ yields x-bounds for the resulting child nodes close to 0, $\frac{1}{2}$ and 1 (v and y analogous), as shown in Figure 4.

Consequently, when taking advantage of this fact, it is possible to store the bounds using only very few bits, without losing too much precision. This observation is the basis of our compression scheme as detailed in the remainder of this section.

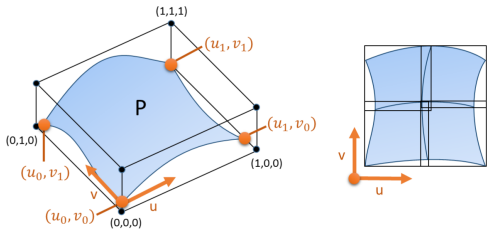


Fig. 4. For reasonably flat patches subdivision in (u, v) yields predictably distributed values in patch-local (x, y) (see Figure 5).

During the subdivision of a possibly displaced patch we test each generated sub-patch for flatness, i.e. our subdivision is adaptive. As soon as a sub-patch is considered sufficiently flat (see Section 3.2), we switch to our compressed BVH (CBVH) representation, i.e. we enter a leaf node in the top-level tree. This leaf stores the local system of the CBVH that represents the further subdivided and compressed patch. Within each CBVH, bounding box coordinates are stored relative to each node's parent's bounds while keeping the orientation of the local root, i.e. all nodes in a single CBVH are in the same coordinate frame (see Figure 2). During this process we employ full-precision arithmetic to subdivide the sub-patches and map the resulting bounding box relative to the bounds of the parent node. This fraction is then quantized and stored in our hierarchy. Thus, the subdivision for a given sub-patch is first stored in n levels of full precision, followed by c levels of our compressed representation, denoted S_{nc} in the following (e.g. S_{32} for three levels of full precision followed by two levels of compressed data).

We encode the CBVH such that it exploits the predictable configuration of child-bounds in two ways: by using strong quantization to store the bounds (see Section 3.4), and by jointly compressing the bounds of the four child-nodes (see Section 3.5).

Note that as soon as we switch to our CBVH, we no longer use fully adaptive subdivision, but fully subdivide until a certain level. The maximum subdivision level can be chosen independently for each CBVH such that all child boxes reach a given accuracy threshold, i.e. adaptive per CBVH, e.g. until the resulting patches cover an area below a predefined threshold. Therefore our whole structure is fully adaptive in the upper full-precision levels of our BVH, while the CBVH can adapt the maximum subdivision level. This is a reasonable compromise, as it allows us to use implicit addressing in the CBVH, which frees us from storing indexing information such as pointers.

3.2 Flatness of Sub-Patches

We determine the flatness of a patch by evaluating its cone of normals [Shirmun and Abi-Ezzi 1993]. For displaced surfaces this entails sampling the height field at the desired target resolution. A patch is then considered flat if the opening angle of this cone, θ, is below some threshold: $\theta < \theta_{\text{flat}}$.

This flatness threshold should be kept small to ensure high fidelity of the local frame. On the other hand, if the threshold is chosen too conservatively, memory savings can drop as compression is applied only at the very bottom of a patch's BVH. However, even conservative values of θ_{flat} result in considerable reduction because of the exponential growth in the number of leaf nodes. Section 5 provides an analysis of how this choice affects image quality, memory consumption and render times.

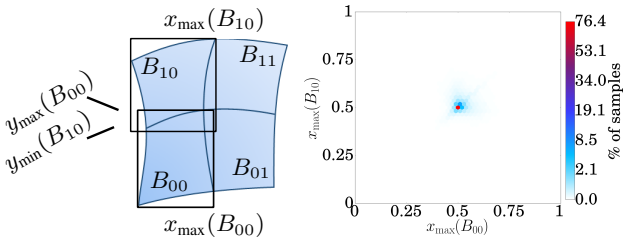
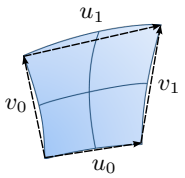


Fig. 5. The distribution of the $x_{\max}$ values of the two left child boxes (configuration shown left) in relation to bounds of the parent node is shown in the histogram (right). Note that almost all values are very close to the predicted value of $\frac{1}{2}$. The histogram is built on the entire T-rex model shown in Figure 1, subdivided to level 6, including displacement. The illustration on the left corresponds to the B_{10} sub-patch shown in Figure 4.

3.3 Alignment of Sub-Patches

To ensure predictable boundaries during subdivision, even for sheared or trapezoidal sub-patches, the local frames stored at the root of the oriented CBVHs are not restricted to rotations, but can be arbitrary projective mappings. To this end we first compute a simple local non-orthogonal, but normal, frame F by averaging the sub-patch's control vectors, as shown in the inset image. Based on this frame we compute a projection P that maps the u and v components to the unit square. We then traverse the patch's child nodes and accumulate their bounding volumes in the projected space, potentially yielding (u, v)-bounds outside the unit square. These bounds can be mapped back to the unit square with a simple scale and bias transform, S. The resulting local, projective transformation that the sub-patch's CBVH is expressed in is thus $P' = SPF$.



Using P' we then recompute the bounding boxes of this CBVH and also accumulate tight bounding volumes in the global frame to avoid introducing overestimation at the frame transition. The upper, full precision part of our BVH then uses these volumes in the global frame when computing bounding volumes for its nodes.

The remainder of this section describes, starting from a local and properly aligned frame, how we reduce the memory footprint of our BVH by quantizing and compressing inner nodes and not storing primitives in the leaf nodes.

3.4 BVH Quantization

The first part of our proposed compression scheme applies quantization to individual bounding box components in a patch's local coordinate frame. Subdivision of a (sub-) patch at $\left(\frac{1}{2}, \frac{1}{2}\right)$ yields four new bounding volumes in the local frame: $B_{00}, B_{01}, B_{10}, B_{11}$. As exemplified in Figure 5, the distribution of their individual bounds is heavily non-uniform (note the non-linear scale) and we assume

$$\begin{array}{ll} (a)\ E[x_{\min}(B_{ij})] = \frac{j}{2} & (c)\ E[x_{\max}(B_{ij})] = \frac{j+1}{2} \\ (b)\ E[y_{\min}(B_{ij})] = \frac{i}{2} & (d)\ E[y_{\max}(B_{ij})] = \frac{i+1}{2} \end{array} \quad (1)$$

where $i, j \in \{0, 1\}$.

Given these expected values we then compute the deviation of each bound, e.g. $x^{\Delta}_{\max}(B_{ij}) = x_{\max}(B_{ij}) - E[x_{\max}(B_{ij})]$, and apply a quantization function, Q, i.e. $x^{Q}_{\max}(B_{ij}) = Q(x^{\Delta}_{\max}(B_{ij}))$. Note that this function must produce conservative bounds to avoid underestimation which would result in false negatives (i.e. holes) during tracing. Therefore our quantization function selects the appropriate bounds of each quantization interval (bin) for minimal (Q^-), and maximal (Q^+) extents, i.e. rounds down or up.

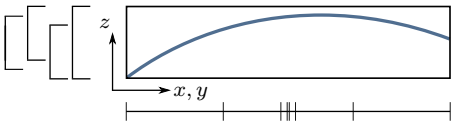


Fig. 6. Q332 and C332 Quantization-pattern for the x and y coordinates, with C332's two-bit representation of the z interval. The three-bits in x and y encode $(-1.0, -0.2, -0.01, -0.001, 0.001, 0.01, 0.2, 1)$ while the two-bit values used for z represent the intervals $([0.0, 1.0], [0.0, 0.7], [0.3, 1.0], [0.2, 0.8])$.

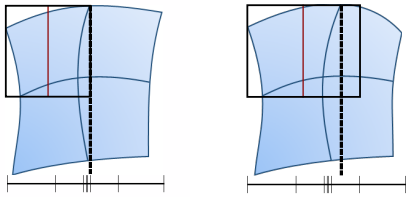


Fig. 7. Overestimation results in splits which are not captured well using the quantization. The left image shows that splits close to the expected value are captured faithfully, while splits which cross quantization boundaries produce larger boxes. Note how the center of the overestimated bounds (right) also no longer captures the sub-patch's center.

In the following we show how the expected value and standard deviation of $x_1 = x_{\max}(B_{00})$ and $x_2 = x_{\max}(B_{10})$ behave under quantization. For the dataset shown in Figure 5 we computed $E[x_1] = 0.5038$, and $\sigma_{x_1} = 0.016$. We exploit the low variance by distributing our quantization bounds closely around zero, similar to a logarithmic distribution. Because we only use a few bits our logarithmic quantization function (Q_L) is implemented by a short lookup-table (see Figure 6). Figure 8 shows the distribution of $Q_L(x_1)$ and $Q_L(x_2)$, for increasing subdivision levels. Note the histogram's non-linear scale and how the strong focus close to the original expected value is clearly present with our logarithmic quantization. For further details on different quantization patterns see our supplemental material.

Our non-uniform quantization captures changes around the expected value with high precision, but larger deviations map to conservative, low resolution bounds. This results in overestimated boxes, and thereby skews the prediction as the center of such boxes no longer coincides with the expected value. Figure 7 gives an illustration of this effect. This is also reflected in the mean and standard deviation of the samples. Referring to the example above we computed $E[Q_L(x_1)] = 0.5031$ and $\sigma_{Q_L(x_1)} = 0.028$. It can be seen that the variance increased, but it is still very low and the error introduced is marginal. See Section 5 for a more thorough analysis.

We typically use a quantization to three or two bits, resulting in eight or four possible values. The standard quantization pattern used for all renderings shown in this paper (and the previous analysis) is based on a three-bit non-uniform quantization for the x and y components and two bits for the (already flat) z components, abbreviated Q332. Its x and y bounds are illustrated in Figure 6. This representation consumes 8 bits of memory per box-vertex and thus 2 bytes per local CBVH bounding box. Since we apply full subdivision for each CBVH indexing is implicit and we arrive at 2 bytes per node, which is in contrast to 24 bytes for implicitly addressed single precision nodes, i.e. a compression factor of 12.

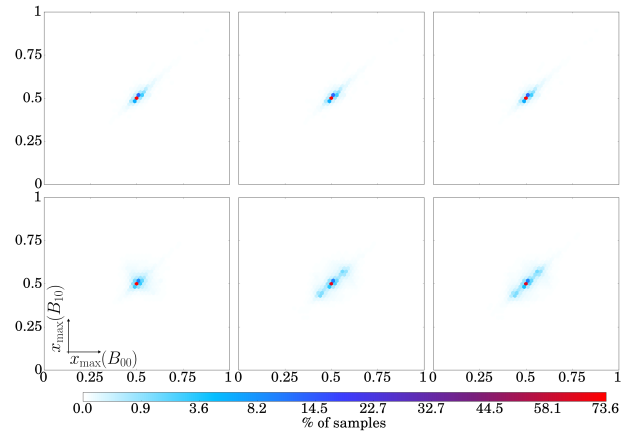


Fig. 8. Distribution and correlation of the $x_{\max}(B_{00})$ (horizontal in sub-plots) and $x_{\max}(B_{10})$ (vertical in sub-plots) fragment bounds illustrated in Figure 5 (left). The top row shows the results for uniform subdivision with 3 levels of uncompressed BVH nodes and 1 level using CBVHs; the bottom row shows 2 levels of uncompressed BVH nodes as well as 3 levels using CBVHs. Left column: reference distribution without quantization; middle column: quantization using Q332; right column: compression using C332.

3.5 BVH Compression

The second part of our compression scheme is jointly encoding bounding-box components of sibling nodes. In the following we present two compression methods which are orthogonal to applying quantization and can be used to further increase the overall compression rate.

Slab Compression. The basic structure of our slab compression is already hinted at, e.g., in Equation 1 (a) which defines two similar expected values,

$$E[x_{\max}(B_{0j}) - x_{\max}(B_{1j})] \approx 0. \tag{2}$$

The geometric interpretation is as follows: The subdivision of a patch P yields four sub-patches in a 2×2 grid, i.e. B_{ij}. Therefore, there are two sub-patches along the v domain (which corresponds to the y axis in the local frame), i.e. B_{i0} and B_{i1}. It can be expected that the x bounds of those two sub-patches, e.g. $x_{\text{min,max}}(B_{0j})$, are very similar. Figure 5 illustrates this relation and shows a histogram of the distribution of the presented values ($x_{\min}(B_{00})$ and $x_{\min}(B_{10})$), based on an analysis of the T-rex model shown in Figure 1, including displacement textures. Figure 8 indicates the correlation of these values (left row) under quantization (middle row) and compression (right row). Note how the values are distributed along the diagonal, indicating strong correlation which is captured nicely with our compression scheme.

Similarly to previous computations we set $x_1 = x_{\max}(B_{00})$ and $x_2 = x_{\max}(B_{10})$ and determined $E[x_1 - x_2] = 8.97 \cdot 10^{-5}$ with $\sigma = 0.020$. We exploit this similarity by conservatively choosing the larger (i.e. conservative) bound to represent both, e.g.

$$x^C_{\max}(B_{0j}) = \max\left(x^Q_{\max}(B_{0j}), x^Q_{\max}(B_{1j})\right) \tag{3}$$

and proceed analogously for the other bounding values. We call this method 'slab compression' because we effectively fit four slabs around the sibling boxes, as illustrated in Figure 9 (a) and (b). This yields a total of 8 bounds for four boxes, which should be compared to storing 4 bounds per box, increasing the compression rate by a factor of two. Using the same quantization as with Q332 and adding slab compression we arrive at 8 bits per local CBVH node, which results in an overall compression factor of 24. We denote this particular combination C332.

In contrast to the Q332 quantization (where both z-bounds were stored with 2 bits, each), the C332 compression only leaves two bits for the whole z-approximation. Figure 6 illustrates how the z-part of a compressed value is interpreted relative to the height of the parent-box. Figure 8 exemplifies how well the original bounding values are captured using C332 (note the non-linear scale). The slight distortion of the distributions shown is a result from overestimation as described in Section 3.4.

Half-Slab Compression. Even further compression can be achieved by considering even more similarities. Given the parent box B_P of the four boxes B_{ij}, Figure 9 (a) shows that, e.g., $x^C_{\min}(B_{i0}) \approx x_{\min}(B_P)$. This similarity holds for all bounds of B_P to the respective bounds of the sub-boxes B_{ij}. This scheme, using the slabs-analogy introduced above, corresponds to only storing the inner part of each slab, see Figure 9 (c) and (d). Concrete evaluation on the T-rex model yields that $E[x_{\min}(B_P) - x^C_{\min}(B_{i0})] = 3.79 \cdot 10^{-5}$ with $\sigma = 0.038$.

This yields another factor of 2 in compression rate, and application on top of C332, which we denote H332, results in a 4-bit representation for local CBVH nodes, which corresponds to a compression rate of 48. Section 5 provides a comparison of the image quality, rendering performance and memory savings achieved by these compression schemes.

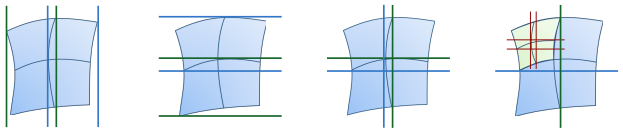


Fig. 9. Adjacent child bounding boxes can be expected to have similar bounds on one axis and those are then combined to slabs. (a) B_{00} and B_{10}, as well as B_{01} and B_{11} exhibit similar bounds on the x axis, while (b) B_{00} and B_{01}, as well as B_{10} and B_{11} share bounds on the y axis. (c) Furthermore, a node's parent-box can also contribute a conservative estimate which is exploited using half-slab compression. (d) Complete bounds are formed by the interaction of two levels of such a CBVH.

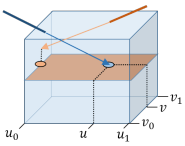
3.6 Traversal Scheme

Ray traversal using our data structure is a two-step process which we implemented for GPUs and CPUs. In the first phase the top-level tree is traversed until a transition node to the compressed BVH is encountered. Using the information in this node the ray data is transformed to the local frame and we compute the intersection with the quantized nodes while decompressing on-the-fly.

Our traversal on the GPU is implemented according to state-of-the-art techniques [Aila and Laine 2009]. We keep traversing the top-level tree and speculatively collect further transition nodes as long as there are threads which have not found a transition node, yet. Then all threads collectively enter the traversal of the quantized hierarchy, thus avoiding divergence that would otherwise greatly impact performance when different node types are intersected in the same warp. We dynamically fetch new rays as soon as at least 12 rays (out of 32) in a warp have terminated. On the CPU our traversal is integrated in Embree [Wald et al. 2014], making heavy use of SIMD instruction sets and wider trees.

During traversal of the quantized part of the hierarchy we keep track of the current node's bounds which are stored incrementally with our method. The parameter interval covered during traversal is implicit in the uniform subdivision of each compressed BVH and is consequently not tracked, but directly computed for leaf nodes.

Since we do not store the leaf mesh for our approximate method, we assume that the surface is hit as soon as a leaf of our hierarchy is reached and we approximate the parameter (u, v) of the hit-point. The fastest approximation is to return the center of the parameter interval as result. Better results are achieved by approximating the (u, v)-coordinates at the hit-point on the surface with the bounding box's z-median as shown in the inset image. There, the ray (blue) is intersected with the z-median surface of the box. Hit points outside the parameter interval of the box are clamped (orange).



If the original geometry is still available (e.g. the subdivision cage or NURBS control points) it is also possible to evaluate the four points spanning the parameter interval on-the-fly, and intersecting the ray with the bilinear patch. As detailed in Section 4, a deferred shading stage can be exploited to provide further computations (such as linear approximations or complete evaluation) while providing a coherent execution path. Furthermore, the hit-point found can also be used as a starting point for iterative methods [Abert et al. 2006]. Our compression is also compatible with keeping the full-precision vertex positions of the leaf mesh to compute exact intersection points. Using our compression scheme in this way does not provide as strong a compression as when the leaf mesh is dropped and approximated by our oriented boxes, but does remove the potential for rendering error due to box-approximation.

However diverse the options of on-the-fly and deferred evaluation, we would like to stress that at subdivision level six the approximation using interpolation on the leaf node's (u, v) bounds (see inset image above) yield very accurate results, as exemplified by Figures 1 and 3 as well as by the images shown in Section 5.

4. USE IN PRODUCTION ENVIRONMENTS

Our method integrates nicely into existing production rendering pipelines. We focus on Catmull Clark subdivision surfaces with high frequency detail stored in displacement maps.

Ptex. The state of the art for storing high resolution material and surface detail in production rendering is Ptex [Burley and Lacewell 2008]. Ptex uses an implicit parametrization on a per face basis as opposed to a global (u, v)-parameterization. This makes our scheme an ideal companion: Ptex frees us from storing explicit (u, v) coordinates at the CBVH root nodes and we provide the patch number and parameter space location needed to fetch data from the textures.

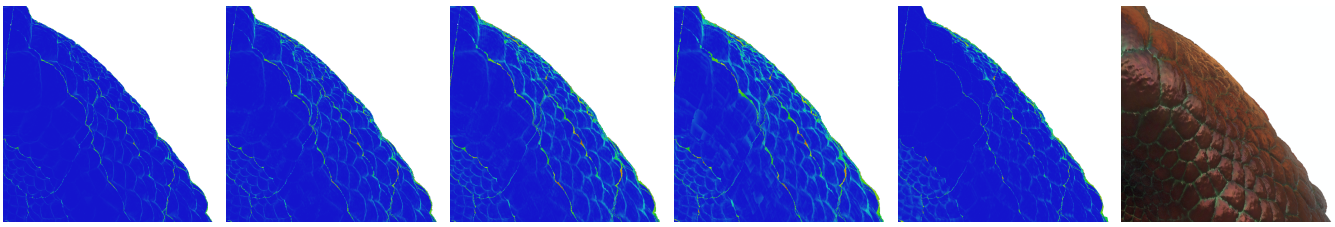
Decoupled Shading. Similar to Eisenacher et al. [2013], ray traversal and shading are two separate stages in our rendering pipeline. Using our compressed BVH we can perform ray traversal even for large production models on the GPU. Output of the traversal are a patch-id and (u, v)-coordinates per ray, but no normal (nor other attributes). We thus use feature adaptive subdivision [Nießner et al. 2012] to evaluate the point at this (u, v)-position and compute the normals of the displaced surface analytically using the approach described by Nießner and Loop [2013]. This step is only a small fraction of the trace time and thus does not contribute much to rendering time. The re-evaluated surface point will be slightly off the original ray, but we can use it well for further shading. Examples of this approach for rendering subdivision with displacement are shown in Figure 1, 3 and 11. Higher resolution images can be found in our supplemental material.

Fig. 11. Barbarian: a displacement mapped model rendered using only box approximation with motion blur and deferred shading. Note the deformation of the face as the barbarian's mouth opens.

Application as Seed for Direct Methods. Methods for the direct intersection of parametric surfaces are prone to numerical problems and rely on good starting values for computing accurate results (under the assumption, that no displacement map is applied). Our approach can be used as a light-weight tool to provide such well behaved starting points to these methods. Given the quite accurate starting point, a single Newton iteration would suffice. This single iteration leads to a very small overhead in the shading stage to evaluate the additional surface point and the two additional derivatives. Compared to directly using the (u, v)-parameters of the starting point, this only doubles the cost of the hit-point and normal calculation. The accurate starting point can also reduce artifacts due to divergence of the Newton iteration [Abert et al. 2006].

Level of Detail. Each approximation that is employed to reduce memory consumption possibly introduces error, and might affect rendering performance. Section 5 evaluates our method with regard to these characteristics. The variety of different settings proposed in Section 3 also provides a reasonable basis for level of detail (LOD) approaches: while the fidelity of our representation degrades under half-slab compression, results are still appropriate for objects when quality requirements are reduced due to LOD. Similarly, the flatness threshold, which influences where compression starts, can be varied continuously with the LOD coefficient. The same holds for the maximum level to which the CBVH is build. This is guided by a maximal area allowed to be covered by the leaf which can be varied analogously.

Motion Blur. Our method also nicely extends to motion blur. The common solution is storing two bounding boxes per node and interpolating between these boxes during traversal, and to proceed similarly for leaf geometry. Our approach supports the same method. As a consequence of keeping the BVH of two key frames we also store two local frames at transition nodes and, in addition to interpolating between CBVH boxes, also interpolate their frames. We have implemented this scheme by enforcing that the two key frames hold transition nodes at the same positions in the BVH. Figure 11 shows an image rendered with our approximate method and motion blur. Note the deformation of the character's face. The respective key frames can be found in our supplemental material.



(a) (b) (c) (d) (e) (f)

Fig. 10. Error in (u, v)-difference for a close-up to the T-rex's nose between different quantization settings and the triangle reference (generated by 7 levels of uniform subdivision). Images (a)-(d) use uniform subdivision to s_{34} (see Section 3.1) and rely only on our box-approximation (see Section 3.6) to estimate the intersection point. (a) box reference: RMSE 0.034, (b) Q332: RMSE 0.048, (c) C332: RMSE 0.062, (d) H332: RMSE 0.069, (e) C332 using adaptive subdivision with $\theta_{\text{flat}} = 72°$: RMSE 0.042, and (f) final result using (e).

5. EVALUATION

In this section we compare ray tracing using our compressed structure to different, recent methods in terms of image quality, rendering time and memory requirements. We have implemented our method for both GPUs (using CUDA, based on the findings of Aila and Laine [2009]) as well as CPUs (exploiting the available SIMD instruction sets), where we integrated it in the open source Embree ray tracing framework [Wald et al. 2014] (our branch is available online[1]). Therefore we compare to both GPU and CPU-based approaches:

Q332, C332, H332 Variants of our approximate method, implemented with state-of-the-art CPU and GPU techniques.

NAIVE A naive pre-tessellation of the subdivision model to the same levels as with our approach and rendered with state-of-the-art ray tracers using CUDA [Aila and Laine 2009] on the GPU and Embree [Wald et al. 2014] on the CPU.

RBVH Rasterized BVHs [Novák and Dachsbacher 2012] replace sub-trees of the BVH with height field atlases rasterized on the GPU. After rasterization the pre-tessellated triangle data is no longer required for rendering and can be discarded. During BVH traversal the method can switch to traversing the sampled height data. To support parametric patches we extended this method to also keep the parameter values of the patches which would otherwise be lost.

PREGEN A pre-tessellated and losslessly compressed method, very similar to Lauterbach et al. [2008], but tailored to parametric patches. Vertices are stored tightly packed and in groups such that low-precision indices can be used to index a patch's vertices, thereby removing the need to store any stripification information. As the method is integrated in Embree the implementation is highly tuned using SSE and AVX data layouts and intersection routines.
We also provide comparison to a GPU implementation which is tuned the same way as our NAIVE GPU version is.

PREGEN^C Our exact variant (see Section 3.6), using oriented, quantized and compressed bounding boxes with a vertex layout inspired by Embree's PREGEN.

CACHED A lazy-build scheme exploiting a shared cache to keep on-the-fly constructed sub-trees. It is tailored to modern many-core architectures [Benthin et al. 2015] and integrated into Embree, making use of, e.g., SSE and AVX. Its performance is coupled to the shared cache's size.

[1] `github.com/kiselgra/embree-compressed`

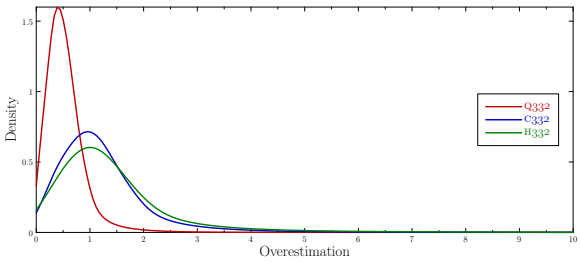


Fig. 12. Overestimation of the surface area of the T-rex model shown in Figure 1 subdivided to s_{33} using Q332 (red), C332(blue) and H332 (green)

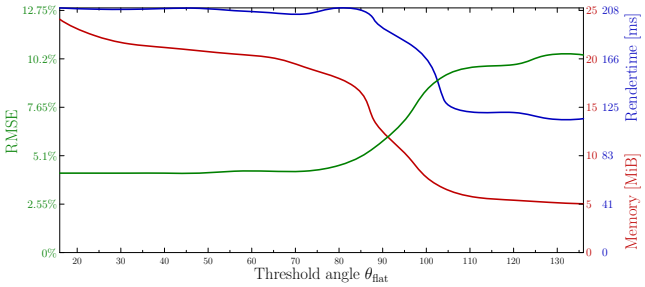


Fig. 13. Memory requirements in comparison to image quality (see also Figure 10) and rendering time (see also Figure 16) for flatness-adaptive subdivision, when rendering the close-up shown in Figure 10 (f). With increasing threshold-angle θ_{flat}, i.e. more relaxed flatness, memory consumption drops from 24 MiB to 5 MiB for the T-rex model shown in Figure 1 while the average error increases from 4% to over 10.2%. Render-time also drops from 210 ms to 115 ms.

5.1 Image Quality

Our proposed method supports compression with two different targets: very strong compression that can introduce overestimation due to box-approximation and still strong, but lossless compression that does not introduce any rendering error. In this section we show that rendering error under lossy compression is not as strong as with common methods, even while providing better compression. We evaluate the quality achieved using our technique by comparing the (u, v)-parameters obtained to those on a highly tessellated mesh (called 'triangle reference') and contrasting the results to using RBVH. Our method uses bounding boxes in patch-local coordinate frames to approximate the actual displaced parametric surface. We refer to this as an oriented, projective, object-space voxeliza-

Table I. Rendering error in (u, v)-space, performance and memory footprint of three versions of RBVH compared to configurations of our method that partly share those characteristics. Values compared in Figure 14 are marked.

Method		RMSE		Performance [Mrps]		Memory [MiB]
Figure 14(a) $\text{RBVH}_{\text{high}}$	$*$	0.071		91.2		2195.3
Figure 14(b) RBVH_{med}		0.084	$*$	201.9		889.2
Figure 14(c) RBVH_{low}		0.110		340.9	$*$	205.9
Figure 14(f) $\text{C}332\ \text{S}_{34}$		0.062		120.9	$*$	186.4
Figure 14(d) $\text{C}332\ \text{S}_{33}$	$*$	0.073		138.4		95.6
Figure 14(e) $\text{C}332\ \text{S}_{22}$		0.092	$*$	191.2		18.6

tion. Even though we change the orientation of this voxelization only once, a non-quantized representation can already introduce overestimation. For a cleaner evaluation we therefore also consider the difference of this representation (called 'box reference') to the triangle reference, as well as the difference of the quantization schemes.

Figure 10 indicates the error introduced by different quantization settings. The most pronounced areas are where different (u, v)-coordinates meet at an edge. However, as shown in Figure 10 (f), the resulting image is still of very high quality. Our supplemental video shows this evaluation over the entire model and under animation.

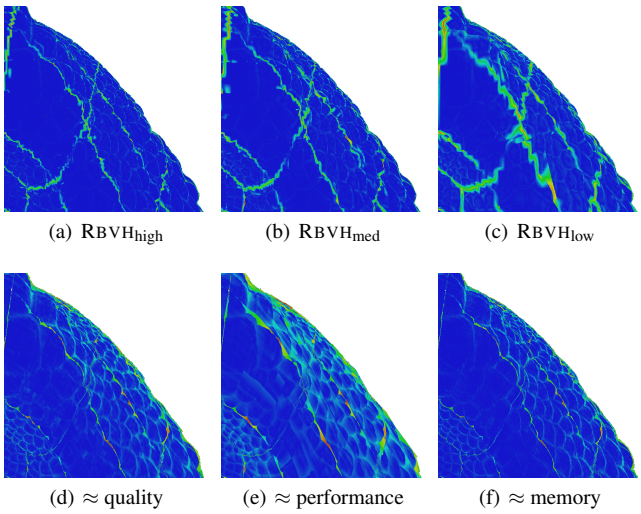
Figure 12 describes the distribution of overestimation for the T-rex model shown in Figure 1. The model was subdivided three times using full-precision, and then another three times using CBVHs (i.e. S_{33}). Note how the overestimation of the quantized, compressed and half-slab compressed variants can also be found in Figure 10 (b), (c) and (d). This is mostly noticeable towards (u, v)-discontinuities and silhouettes where the original patch is slightly enlarged (e.g., see Figure 10 (d)).

Even though our compression scheme introduces an error our supplemental video demonstrates that, overall, it is very low and, most importantly, does not introduce noise or inter-frame inconsistencies. This is also clearly visible in the visualization of the direct (u, v)-coordinates obtained from our traversal.

Furthermore, Figure 13 shows that with increasing threshold-angle θ_{flat} memory requirements and render time drop while the rendering error grows. Note that, in contrast to the uniform evaluation provided in Section 5.2, the figure presents the memory footprint of adaptive subdivision for the close-up to the T-rex's nose, see Figure 10 (f).

In settings where no rendering error is tolerable our compression scheme can still be applied to the hierarchy while keeping leaf triangles. As shown in Section 5.2 and 5.3 memory consumption is much increased in contrast to using our approximation, but still reduced to around 60% compared to PREGEN, while also yielding up to 40% increase in traversal performance on the GPU. Note, however, that the images shown in Figures 1, 3, 11 and 10 are all rendered using box-approximation, only. As can be seen in Figure 10, using half-slab compression introduces stronger overestimation at patch-boundaries and we thus usually rely on $\text{C}332$ when using our box-approximation.

A different approach to approximate scene geometry is using RBVH [Novák and Dachsbacher 2012]. The approach is similar to our method in that a two-level hierarchy is employed, where a transition to an approximate representation is introduced as soon as the geometry contained in a sub-tree is flat enough. In contrast to our approach, RBVH uses rasterized height fields in a local frame and stores them in texture atlases. Therefore, image quality is mainly



(a) $\text{RBVH}_{\text{high}}$ (b) RBVH_{med} (c) RBVH_{low}

(d) $\approx$ quality (e) $\approx$ performance (f) $\approx$ memory

Fig. 14. We show equal quality 14(d), performance 14(e) and memory 14(f) comparisons of our method to RBVH at three quality levels. Full details are listed in Table I.

driven by the resolution of the texture atlases (and the flatness criterion), not directly by subdivision level.

To evaluate this method in the context of parametric patches we extended the implementation of Novák and Dachsbacher [2012] to also store patch-ids and (u, v)-parameters, allowing us to evaluate the limit surface position and normal during shading. For comparison we chose three RBVH quality settings, comparable to our method's rendering error and speed. The top row of Figure 14 indicates the rendering error of three different versions of RBVH. The high-quality $\text{RBVH}_{\text{high}}$ version (a) exhibits a similar error as our method in (d), $\text{C}332\ \text{S}_{33}$, which both takes up less memory and renders faster. The medium-quality RBVH_{med} version (b) renders a little faster than our low-quality version (e), $\text{C}332\ \text{S}_{22}$, which also produces stronger rendering error, but is much more compact. Finally, the low-quality RBVH_{low} version (c) requires a similar amount of memory as our high-quality version (f), $\text{C}332\ \text{S}_{34}$. However, even though RBVH_{low} performs better than all other versions, the rendering error is severe. Table I also demonstrates that using our method the rendering error can be much reduced (e.g. compare high quality $\text{C}332\ \text{S}_{34}$ with $\text{RBVH}_{\text{high}}$, showing similar performance, and RBVH_{low}, of comparable size).

5.2 Memory Requirements

Our main concern addressed by reducing the memory footprint of BVH and primitive representations is to be able to keep the ray traversal-part of a decoupled shading system (see Section 4) as local as possible. By this we mean that, regardless of the working set required for shading, we strive to achieve a compact enough representation to allow ray traversal on the GPU (when otherwise the computation would have to be run on the host CPU) or in-core (when otherwise virtual memory thrashing would bring down system performance). Therefore our analysis of required memory focuses on the memory required during ray traversal. We also list memory requirements during shading, showing that our method only adds a small fraction to it. To ensure equal tessellation levels across all considered methods we compare uniformly subdivided patches. The resulting relationships carry over to adaptive subdivision with equal levels. To this end we will show that, for reason-

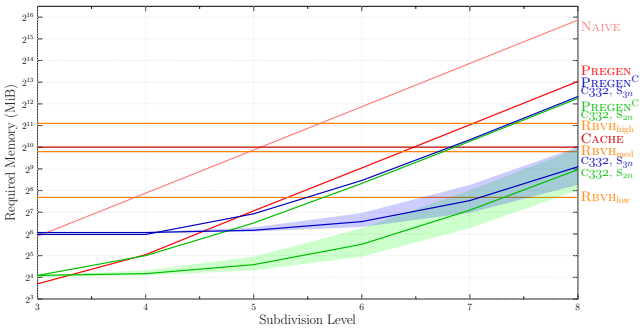


Fig. 15. Memory requirements (in MiB) for the BVH and leaf data of the reference approaches (red) described at the outset of Section 5 compared to different configurations of our C332 method (green: with full precision subdivision until level 2; blue: until level 3). Measurements were taken on the T-rex shown in Figure 1, consisting of 5812 patches. Shaded regions span the difference of the corresponding Q332 (top) and H332 (bottom) versions. Note the logarithmic scale, and that a full table can be found in our supplemental material.

ably finely tessellated surfaces, both the lossy as well as the lossless compression we propose require significantly less memory as compared to most established methods. The following section will then show how these compare in terms of rendering performance and demonstrate that the constant-memory solutions can entail severe performance drawbacks.

Figure 15 indicates the development of memory requirements for the different methods described at the outset of Section 5 as compared to our method with different configurations. The figure shows the complete memory footprint of the respective methods for traversal (i.e. including the top-level, local frames, and, if applicable, leaf geometry). It can be seen that, with the exception of on-the-fly tessellation (CACHED) and RBVH, memory grows exponentially in all cases. Figure 15 demonstrates that there is already a strong compression in Embree's PREGEN implementation (red) as compared to the NAIVE representation (pink). The figure also shows two lines representing C332 S_{2n} (green) and C332 S_{3n} (blue) with the corresponding half-slab compressed (H332) and quantized-only (Q332) versions at the lower and upper limits of the shaded regions, respectively. The compression rate of both of these instances of Q332 is already as strong as the difference between NAIVE and PREGEN, while using compression decreases memory requirements by another factor of 2. Half-slab compression can be used to reduce this by yet another factor of 2. Please note that our supplemental material contains a full tabulation of the data shown in Figure 15, for different subdivision and compression levels both, for the T-rex shown in Figure 1, consisting of 5812 patches, as well as for the barbarian shown in Figure 11, consisting of 51949 patches.

From Figure 15 it is visible that for higher subdivision levels, such as required for close-ups or highly detailed and displaced setups, the compression factor achieved using our approximate method is $16 : 1$ over lossless state-of-the-art compression (and up to $60 : 1$ to a naive indexed face set). As shown in the evaluation of image quality above, this compression also introduces an error due to overestimation, however, the error is minor and well below that of competing methods. Even for settings in which no error can be tolerated, our compression scheme can be used while keeping exact, full precision vertex data (PREGEN^C) and still achieve close to $2 : 1$ compression over the state of the art.

Table II. Breakdown of the memory requirements (in MiB) of our method, divided by the different stages of a decoupled shading system (see Section 4). The first column uses our compressed, approximate method, C332, followed by using our method with leaf triangles and aggressive half-slab compression, PREGEN^{H332}, and using NAIVE BVH construction. The upper part lists requirements during ray traversal, the lower part data required during shading, including the structures used by feature adaptive subdivision (FAS) [Nießner et al. 2012] used to evaluate limit position and normal.

Method	C332	PREGEN^{H332}	PREGEN	NAIVE
Level	S_{33}	S_{24}	S_6	S_6
Traversal				
Top level	20.4	5.6	182.1	2728.0
Matrices	45.4	5.7	0.0	0.0
Bottom level	29.8	15.1	0.0	0.0
Leaf geometry	0.0	281.0	281.0	1013.0
Sum	**95.6**	**307.4**	**463.1**	**3741**
Shading				
FAS	65.2	0.0	0.0	0.0
Displacement	300.0	0.0	0.0	0.0
Color texture	1300.0	1300.0	1300.0	1300.0
Sum	**1665.2**	**1300.0**	**1300.0**	**1300.0**

However, our method is not as effective for low subdivision levels, which also holds for when keeping full precision vertex data along our compressed structure (PREGEN^C). Using PREGEN^C is sensible starting from level 4 (e.g. with half-slab compression, PREGEN^{H332}, at S_{22}) where the same amount of memory is used and traversal time using our method is below PREGEN. Starting from level 5 PREGEN^C performs much better as PREGEN, while still providing a compression rate above $1.5 : 1$. Furthermore, starting from level 5 (e.g. S_{32}) the initial overhead (matrix storage) of our approximate method, C332, pays off and strong compression is achieved.

Table II gives a breakdown of the memory requirements of our compressed, approximate method C332 at level S_{33}, together with our exact version under half-slab compression, PREGEN^{H332}, at S_{24}. We generally use half-slab compression with PREGEN^C as it does not suffer from inaccuracies due to box-approximation and both, rendering performance (see Section 5.3) and memory consumption (see Figure 15), are improved. Table II contrasts these two versions to a NAIVE BVH. It can be seen that, at the same subdivision level, our approximate structure shows a much reduced memory footprint. The amount of memory saved by not storing leaf geometry easily outweighs the transition information (matrix storage). However, it is also visible that our method increases memory requirements in the shading stage as feature adaptive subdivision (FAS) [Nießner et al. 2012] is employed to find the limit position and normal, which are then modified according to the information in the displacement map. We only list a single color texture, but in contrast to multiple layers of textures, as is common [Eisenacher et al. 2013], the memory overhead introduced at the shading stage is negligible.

5.3 Rendering Performance

In this section we show how the methods evaluated regarding image quality and memory consumption compare in terms of rendering performance. It will be shown that, even though our compression scheme results in very compact structures, decompression overhead is not significant and performance is competitive, or even

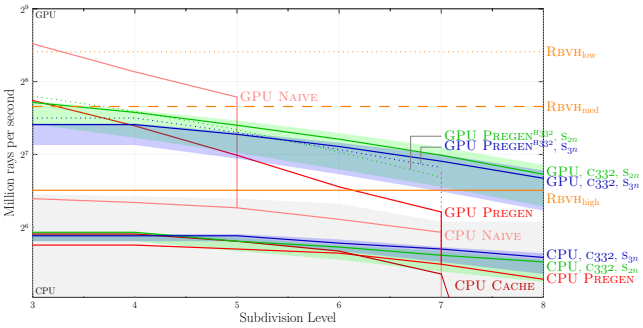


Fig. 16. Rendering performance in millions of rays per second of different configurations of our method on the T-rex model. Green: C332 at S_{2n}, blue: C332 at S_{3n}. The upper limits of the shaded regions are the respective H332 variants, the lower limits the Q332 variants. The graph is divided into methods run on the GPU (top part) and the CPU (shaded, bottom part). Vertical lines indicate that the respective method reached a memory limit. Note the logarithmic scale, and that a full table can be found in our supplemental material.

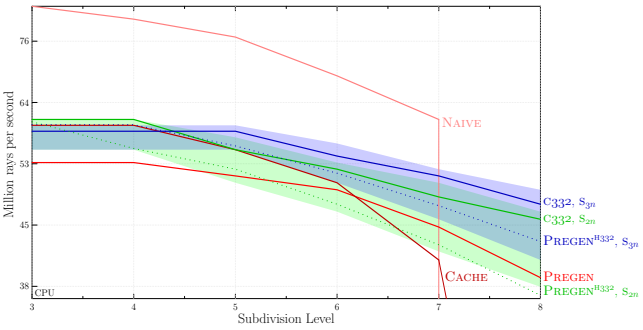


Fig. 17. Close-up of the rendering performance using Embree (i.e. on the CPU) from Figure 16. Note the logarithmic scale.

better than state-of-the-art pre-generated structures. We will also mention that very fast constant-memory solutions exhibit good performance, but as presented earlier show much stronger rendering error.

Figure 16 compares the rendering performance of different configurations of our approach to the methods described at the outset of Section 5. We used a Geforce GTX 780 in our evaluation and rendered the T-rex model (as shown in Figure 1) at 2k×2k resolution. The figure displays ray traversal performance in millions of rays per second for primary rays. We refer to our supplemental material for a full tabulation and performance for incoherent rays. Note that we also provide a detailed breakdown of rendering times using our approach at the end of this section.

Figure 16 lists ray traversal performance on the GPU (upper part) as well as on the CPU (lower, shaded area). The figure shows that using the compressed PREGEN (red) method is already much slower than using a NAIVE (pink) BVH on top of an indexed face set, both on GPU (using Aila and Laine's method [2009]) and CPU (using Embree [Wald et al. 2014]). Note how after subdivision level 5 NAIVE is no longer available on the GPU (shown by its performance dropping to the CPU level). This is also the case for PREGEN, however only at very high subdivision levels. The performance of the CACHED approach is very good until the cache size is only a fraction of the entire BVH. Figure 17 shows this more clearly, and also incorporates measurements for our Embree versions of PREGEN^{C}.

Furthermore, Figure 16 shows that RBVH with reduced quality settings exhibits very good rendering performance. However, it should be noted that for the fastest method, RBVH_{low}, image quality is far inferior to using one of the slower methods. Also, even though RBVH_{med} is still 46% faster than, e.g., C332 at S_{33} (which provides similar quality), RBVH_{med} also requires 9.3 times as much memory (see Table I).

Figure 16 also exemplifies that our approximate methods can run on the GPU for very high levels while performing much better than PREGEN. Similarly, our exact methods require less memory (see Figure 15) which allows keeping more data in cache and thus also shows better performance compared to PREGEN.

Table III shows rendering times for the T-rex (see Figure 1) for our compressed, approximate method C332 at level S_{33}, together with our exact version under half-slab compression, $\text{PREGEN}^{\text{H332}}$, at S_{24}, regular PREGEN at S_6 and NAIVE, also at S_6 accumulated over 32 samples. It shows that, for models with a single texture layer, shading time increases by 7.1% due to surface attribute evaluation. Compared to ray traversal on the GPU this evaluation is expensive, however, when compared to tracing on the CPU, or much worse, a system that would otherwise suffer from virtual memory thrashing, we believe this increase is tolerable. Furthermore, if our compressed hierarchy is applied to PREGEN, e.g. by using $\text{PREGEN}^{\text{H332}}$, no such penalty has to be payed.

Table III. A breakdown of the render time into the different stages of our system. We accumulated traversal, surface evaluation and shading time over five bounces and 32 samples per pixel in our path tracer on the T-rex model. Traversal and attribute evaluation was executed on the GPU while the shading stage ran on the CPU. All times are given in seconds. For memory requirements refer to Table II.

Method	C332	$\text{PREGEN}^{\text{H332}}$	PREGEN	NAIVE
Level	S_{33}	S_{24}	S_6	S_5
Ray traversal	1.60	2.29	2.48	1.27
Attribute evaluation	1.02	–	–	–
Shading	14.33	14.33	14.33	14.33

6. LIMITATIONS AND FUTURE WORK

One of the major obstacles in rendering very large scenes is preventing the system from degrading under virtual memory thrashing [Yoon et al. 2006]. This concern became even more apparent with the movie industry's shift towards path tracing [Christensen et al. 2006]. We believe that our proposed method can help to relax memory limits, firstly by providing an approximate representation with very strong compression and only small and controllable error. Secondly, our method can also be used to reduce the memory footprint of current, state-of-the-art compact representations (such as PREGEN) to almost half the original size.

The main limitation of our method is that, similar to other pregenerated approaches, its memory requirements are still exponential in the subdivision depth. Thus, using our compressed representation can only delay the use of swapping by a constant factor (16 over PREGEN, or 2 when using our exact method). However, in the critical interval where other approaches already suffer virtual memory degradation our scheme can be employed to continue tracing at full speed.

A result of storing matrices at the transition level is that the full compression factor of our method is not available at very low subdivision levels. As Figure 15 shows, our compressed representation

pays off starting from level 4 onwards. We believe that this is easily managed by deciding against a transition step for such low levels during adaptive subdivision.

As described in Section 5.1 our approximate methods (Q332, C332, H332) introduce some rendering error which, however small, might limit the available compression for certain applications to that of using our exact method (PREGEN^C). Further limitations are that with motion blur (see Section 4) both key-frames have to be fully tessellated to the same levels (i.e. memory requirements increase slightly) and the construction of our hierarchy exhibits an overhead of 15% as compared to standard top-down subdivision-surface BVH construction. Even though we have not optimized this part of our algorithm it is conceivable that this overhead is an obstacle to using our method with on-the-fly subdivision such as CACHED. On the other hand, such methods might benefit from being able to keep more data in a fixed-sized cache. We believe that this is an interesting topic for further research. Considering the benefits of half-slab compression it seems promising to look into compressing across more than two hierarchy levels.

Acknowledgments

The authors would like to thank Manuel Kraemer for maintaining the OpenSubdiv library and for being very open to feature requests and responsive to bug reports and Jan Novák for sharing his implementation of RBVH. The authors would also like to thank the Walt Disney Animation Studios for giving access to the T-rex model and its detailed textures, Jonathan Dupuy for providing the key-frames for the animation shown in our video, Autodesk for providing the barbarian model and Yasutoshi Mori for making the sports car model available under a CC-BY license. We also gratefully acknowledge the generous funding by the German Research Foundation (GRK 1773)

REFERENCES

ABERT, O., GEIMER, M., AND MULLE, S. 2006. Direct and Fast Ray Tracing of NURBS Surfaces. *IEEE Symposium on Interactive Ray Tracing 2006*, 161–168.

AILA, T. AND LAINE, S. 2009. Understanding the Efficiency of Ray Traversal on GPUs. *Proceedings of the conference on high performance graphics 2009*, 145–149.

BAUSZAT, P., EISEMANN, M., AND MAGNOR, M. A. 2010. The Minimal Bounding Volume Hierarchy. *Vision, Modeling, and Visualization*, 227–234.

BENTHIN, C., BOULOS, S., LACEWELL, D., AND WALD, I. 2007. Packet-based Ray Tracing of Catmull-Clark Subdivision Surfaces. *SCI Institute, University of Utah, Technical Report# UUSCI-2007-011*.

BENTHIN, C., WOOP, S., NIESSNER, M., SELGRAD, K., AND WALD, I. 2015. Efficient Ray Tracing of Subdivision Surfaces using Tessellation Caching. In *Proceedings of the 7th High-Performance Graphics Conference*. ACM.

BURLEY, B. AND LACEWELL, D. 2008. Ptex: Per-Face Texture Mapping for Production Rendering. *Computer Graphics Forum 27,* 4, 1155–1164.

CATMULL, E. AND CLARK, J. 1978. Recursively Generated B-Spline Surfaces on Arbitrary Topological Meshes. *Computer-aided design 10,* 6, 350–355.

CHRISTENSEN, P. H., FONG, J., LAUR, D. M., AND BATALI, D. 2006. Ray Tracing for the Movie 'Cars'. *IEEE Symposium on Interactive Ray Tracing 2006*, 1–6.

CHRISTENSEN, P. H., LAUR, D. M., FONG, J., WOOTEN, W. L., AND BATALI, D. 2003. Ray Differentials and Multiresolution Geometry Caching for Distribution Ray Tracing in Complex Scenes. *Computer Graphics Forum 22,* 3, 543–552.

COOK, R. L., CARPENTER, L., AND CATMULL, E. 1987. The Reyes Image Rendering Architecture. *ACM SIGGRAPH Computer Graphics 21,* 4, 95–102.

CRASSIN, C. 2011. GigaVoxels (a Voxel-Based Rendering Pipeline for Efficient Exploration of Large and Detailed Scenes). Ph.D. thesis, Universite de Grenoble.

DEROSE, T., KASS, M., AND TRUONG, T. 1998. Subdivision Surfaces in Character Animation. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 85–94.

EISENACHER, C., NICHOLS, G., SELLE, A., AND BURLEY, B. 2013. Sorted Deferred Shading for Production Path Tracing. *Computer Graphics Forum 32,* 4, 125–132.

GEIMER, M. AND ABERT, O. 2005. Interactive Ray Tracing of Trimmed Bicubic Bézier Surfaces without Triangulation. *WSCG 2005 Conference Proceedings*, 71–78.

HOPPE, H., DEROSE, T., DUCHAMP, T., HALSTEAD, M., JIN, H., MCDONALD, J., SCHWEITZER, J., AND STUETZLE, W. 1994. Piecewise Smooth Surface Reconstruction. *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 295–302.

HUBO, E., MERTENS, T., HABER, T., AND BEKAERT, P. 2006. The Quantized kd-Tree: Efficient Ray Tracing of Compressed Point Clouds. *IEEE Symposium on Interactive Ray Tracing 2006*, 105–113.

KAJIYA, J. T. 1982. Ray Tracing Parametric Patches. *ACM SIGGRAPH Computer Graphics 16,* 3.

KIM, T.-J., BYUN, Y., KIM, Y., MOON, B., LEE, S., AND YOON, S.-E. 2010. HCCMeshes: Hierarchical-Culling Oriented Compact Meshes. *Computer Graphics Forum 29,* 2, 299–308.

KIM, T.-J., MOON, B., KIM, D., AND YOON, S.-E. 2010. RACBVHs: Random-Accessible Compressed Bounding Volume Hierarchies. *IEEE Transactions on Visualization and Computer Graphics 16,* 2, 273–286.

LAINE, S., KARRAS, T., AND AILA, T. 2013. Megakernels Considered Harmful: Wavefront Path Tracing on GPUs. *Proceedings of the 5th High-Performance Graphics Conference*, 137–143.

LAUTERBACH, C., YOON, S.-E., TANG, M., AND MANOCHA, D. 2008. ReduceM: Interactive and Memory Efficient Ray Tracing of Large Models. *Computer Graphics Forum 27,* 4, 1313–1321.

MAHOVSKY, J. A. 2005. Ray Tracing with Reduced-Precision Bounding Volume Hierarchies. Ph.D. thesis, University of Calgary.

NIESSNER, M. AND LOOP, C. 2013. Analytic Displacement Mapping Using Hardware Tessellation. *ACM Transactions on Graphics (TOG) 32,* 3, 26.

NIESSNER, M., LOOP, C., MEYER, M., AND DEROSE, T. 2012. Feature-Adaptive GPU Rendering of Catmull-Clark Subdivision Surfaces. *ACM Transactions on Graphics (TOG) 31,* 1, 6.

NISHITA, T., SEDERBERG, T. W., AND KAKIMOTO, M. 1990. Ray Tracing Trimmed Rational Surface Patches. *Proceedings of the 17th annual conference on Computer graphics and interactive techniques 24,* 4, 337–345.

NOVÁK, J. AND DACHSBACHER, C. 2012. Rasterized Bounding Volume Hierarchies. *Computer Graphics Forum 31,* 2, 403–412.

PABST, H. F., SPRINGER, J. P., SCHOLLMEYER, A., LENHARDT, R., LESSIG, C., AND FROEHLICH, B. 2006. Ray Casting of Trimmed NURBS Surfaces on the GPU. *IEEE Symposium on Interactive Ray Tracing 2006*, 151–160.

PHARR, M., KOLB, C., GERSHBEIN, R., AND HANRAHAN, P. 1997. Rendering Complex Scenes with Memory-Coherent Ray Tracing. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 101–108.

ROCKWOOD, A., HEATON, K., AND DAVIS, T. 1989. Real-Time Rendering of Trimmed Surfaces. *Proceedings of the 16th annual conference on Computer graphics and interactive techniques 23,* 3, 107–116.

RUSINKIEWICZ, S. AND LEVOY, M. 2000. QSplat: A Multiresolution Point Rendering System for Large Meshes. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 343–352.

SCHÄFER, H., NIESSNER, M., KEINERT, B., STAMMINGER, M., AND LOOP, C. 2014. State of the Art Report on Real-Time Rendering with Hardware Tessellation. *Eurographics 2014 State of the Art Reports*, 93–117.

SEGOVIA, B. AND ERNST, M. 2010. Memory Efficient Ray Tracing with Hierarchical Mesh Quantization. *Proceedings of Graphics Interface 2010*, 153–160.

SHIRMUN, L. A. AND ABI-EZZI, S. S. 1993. The Cone of Normals Technique for Fast Processing of Curved Patches. *Computer Graphics Forum 12,* 3, 261–272.

TEJIMA, T., STUDIOS, P. A., FUJITA, M., AND MATSUOKA, T. 2015. Direct Ray Tracing of Full-Featured Subdivision Surfaces with Bézier Clipping. *Journal of Computer Graphics Techniques (JCGT) 4,* 1, 69–83.

TOTH, D. L. 1985. On Ray Tracing Parametric Surfaces. *ACM SIGGRAPH Computer Graphics 19,* 3, 171–179.

WALD, I., BOULOS, S., AND SHIRLEY, P. 2007. Ray Tracing Deformable Scenes Using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics (TOG) 26,* 1, 6.

WALD, I., WOOP, S., BENTHIN, C., JOHNSON, G. S., AND ERNST, M. 2014. Embree–A Ray Tracing Kernel Framework for Efficient CPU Ray Tracing. *ACM Transactions on Graphics (Proceedings 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.