A High-Resolution Compression Scheme for Ray Tracing Subdivision Surfaces with Displacement

ALEXANDER LIER, MAGDALENA MARTINEK, MARC STAMMINGER, and KAI SELGRAD
Computer Graphics Group, University of Erlangen-Nuremberg

Fig. 1. Our novel leaf quantization scheme provides much more exact results than earlier methods but still achieves similar compression rates. The model shown above is composed of 8 control points and a displacement map that introduces 9 spikes on each patch. A triangle-based reference (left), refined to level 6, is approximated very well by our method (center), also refined to level 6. Our earlier, voxel-based approach, even if refined to level 7 (right), does not capture strong displacement as well as our new method. Considering the very similar overall compression of down to 31.3% (new) and 41.6% (voxel-based), our new approach offers comparable compression rates accompanied with improved geometric fidelity.

Subdivision surfaces, especially with displacement, are one of the key modeling primitives used in high-quality rendering environments, such as, e.g., movie production. While their use easily maps to rasterization-based frameworks, they pose a significant challenge for ray tracing environments. This is due to the fact that incoherent access patterns require storing or caching fully tessellated and displaced meshes for efficient intersection computations. In this paper we use a two-tier hierarchy built on a scene's patches. It relies on compressed and quantized bounding volumes on the second tier to reduce the size of the BVH itself. Based on this acceleration structure, we propose a quantized, compact approximation for leaf nodes while being faithful to the underlying patch-geometry. We build on recent advances and present a system that shows competitive performance regarding run-time speed, which is close to full-resolution pre-tessellation methods as well as to previous compression approaches. Ultimately, we provide strong compression of up to a factor of 5 : 1 compared to state-of-the-art methods while maintaining high geometrical fidelity surpassing similarly compact approximations and getting close to uncompressed geometry.

CCS Concepts: • **Computing methodologies** → **Ray tracing**; **Parametric curve and surface models**;

Authors' address: Alexander Lier, alexander.lier@fau.de; Magdalena Martinek, magdalena.martinek@fau.de; Marc Stamminger, marc.stamminger@fau.de; Kai Selgrad, kai.selgrad@fau.de, Computer Graphics Group, University of Erlangen-Nuremberg.

2577-6193/2018/8-ART33 $15.00
https://doi.org/10.1145/3233308

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

ACM Reference Format:
Alexander Lier, Magdalena Martinek, Marc Stamminger, and Kai Selgrad. 2018. A High-Resolution Compression Scheme for Ray Tracing Subdivision Surfaces with Displacement. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 2, Article 33 (August 2018), 17 pages. https://doi.org/10.1145/3233308

1 INTRODUCTION

Parametric surfaces are a well-established tool for describing smooth, continuous, and optionally displaced patches and models. Together with ray-tracing techniques for global illumination, these primitives can be employed to create convincing renderings with impressive detail. Therefore, e.g., feature film productions rely on these primitives [Christensen et al. 2006; DeRose et al. 1998]. Rendering displaced parametric surfaces is not easily possible using direct evaluation. Alternative approaches usually require the conversion from these higher-order shapes to finely tessellated quad or triangle meshes. However, as a consequence of the fact that the number of generated vertices grows exponentially with each individual subdivision step, such detailed meshes demand enormous amounts of memory.

In this paper, we describe an approach to improve the visual quality of compressed, tessellated parametric surfaces, while simultaneously maintaining competitive compression rates and rendering performance. In earlier work [Selgrad et al. 2016], we presented a method for strongly compressing parametric patches (such as subdivision surfaces [Catmull and Clark 1978] and NURBS) by applying quantization, sharing bounding box components, and approximating leaf nodes. Even though achieving high compression rates and showing lower rendering error than competing methods, when viewed close-up, or under strong displacement, the leaf-level approximation becomes visible. We propose a remedy, as shown in the teaser (Figure 1), for these cases by introducing a novel approximation for the leaf nodes: Instead of interpreting the leaf-level bounding boxes as voxels and using the intersection values of those to find $(u, \quad)$- and surface hit information, we employ an additional geometry level composed of quantized bounding-volume approximations based on the encasing bilinear patches. This method is in line with the original algorithm [Selgrad et al. 2016], but the image quality achieved is much closer to using a pre-tessellated triangle reference than to the leaf-voxelization.

As the algorithm is closely related to our earlier work. [Selgrad et al. 2016], we will provide only a very short overview of the broader spectrum of related work in Section 2 and then provide a more detailed description of our primary reference in Section 3. Following that, we give an in-depth description of our new leaf-level approximation in Section 4 before comparing it to previous work in terms of image quality, rendering time, and overall compression in Section 5. We conclude our presentation with Section 6.

2 RELATED WORK

Ray tracing of parametric surfaces is a well researched topic. The easiest method may be fine tessellation of the surfaces in combination with conventional ray tracing of the resulting triangle mesh [Benthin et al. 2015; Catmull and Clark 1978]. Yet, for complex production-scale scenes the huge number of necessary triangles leads to a tremendous memory consumption, which can be reduced by applying adaptive tessellation [Christensen et al. 2006, 2003]. In particular for GPU-ray-tracing [Aila and Laine 2009], high memory consumption is prohibitive, but also on CPU-systems performance drops dramatically as soon as swapping becomes necessary [Yoon et al. 2006].

An alternative is employing direct numerical computation of intersections between a ray and B-Spline, NURBS, or subdivision surfaces [Abert et al. 2006; Geimer and Abert 2005; Kajiya 1982; Tejima et al. 2015]. These approaches work directly on the input description of the surface, and may cache intermediate results, requiring less memory than full tessellation. However, these approaches suffer from numerical instability and long computation times. Even worse, most of these approaches cannot handle displacements applied to the surfaces [Nie□ner and Loop 2013], which makes them almost useless for production rendering.

Sorting rays prior surface intersection removes the need for caching. Hanika et al. [2010] propose a two-layer BVH, where the second-tier BVH is generated on-the-fly with geometry shaders and the leaf-level resembles micro-polygons. Before traversing the second layer, all relevant patch/ray intersection candidates are recorded. Then, the individual patch BVHs are constructed and traversed with the matching candidate-sets.

Another approach is to use a voxelization of the objects. Such voxelizations can be used for interactive rendering [Crassin 2011] or fast interactive global illumination [Crassin et al. 2011]. Voxelizations are a good means to prefilter detailed geometry, e.g. for fast occlusion queries [Lacewell et al. 2008] or for fast global illumination [Christensen and Batali 2004]. Usually, voxelizations are based on a uniform 3D grid, but also voxel octrees [Laine and Karras 2010] or leaves of a BVH or kd-tree [Áfra 2012] can be interpreted as voxels. To avoid the blocky appearance of voxels, per-voxel clipping planes can be defined [Áfra 2012; Laine and Karras 2010; Selgrad et al. 2016].

Considering the simplicity and advantages of tessellation and acknowledging its drawback regarding memory requirements, a reasonable consequence is additionally applying compression on the generated meshes. Lauterbach et al. [2007; 2008] and Segovia at al. [2010] describe such data structures for leaf triangle data, which can be directly ray traced with only moderate impact on performance.

Increasing numbers of primitives are accompanied with rapidly growing acceleration structures, which demands compression of them as well. A useful approach for BVH nodes is defining their bounds in relation to their parent's extent. Combined with a conserving reduction in numerical precision, this represents a hierarchical quantization [Hubo et al. 2006; Mahovsky 2005; Rusinkiewicz and Levoy 2000]. Quantized values lead to small overestimations of the actual box dimension, which results in some performance loss [Bauszat et al. 2010]. Kim et al. [2010b] compress whole subtrees of BVHs with a ratio of 12 : 1 and decompress them lazily during traversal. In other work, Kim et al. [2010a] present a scheme that can be directly traversed but also exhibits a weaker compression ratio of 3.6 : 1. Novak et al. [2012] propose an alternative BVH, where the leaves are height fields that can be stored and ray traced efficiently. Policarpo et al. [2006; 2005] propose a related approach for real-time rendering, where intersections with height-field and non-height-field representations are applied to add surface details in the fragment shader.

BVHs for primitives originating from parametric surfaces can be considered as a special case. First, good BVHs can be derived directly from a patch's quadtree, which can be obtained from consecutive subdivision or tessellation. Second, using the full tree [Benthin et al. 2015], its nodes can be addressed implicitly and a bilinear geometry representation is most suitable. Third, lower-level subtrees of patches often describe rather flat areas, the bounds of which can be predicted well, and thus quantization with only few bits can suffice. In combination with other optimizations to reduce memory consumption, we applied these findings in earlier work [Selgrad et al. 2016] to compress BVHs to 8 bits per node. Furthermore, instead of storing leaf triangles, we directly exploited the BVH's leaf nodes as surface voxels, similarly to R-LOD [Yoon et al. 2006]. As a result, we could keep very finely tessellated surfaces even in GPU memory and ray trace such surfaces with only moderate impact on performance.

3 COMPRESSED BVHS AND LEAF-VOXELS

As our method is based on earlier work [Selgrad et al. 2016], we will give a brief overview of the foundational algorithm in this section. We suggested a two-level-BVH representation for compressing quadrilateral geometry derived from potentially displaced parametric surfaces. The top-level is a standard, uncompressed BVH composed of globally axis-aligned bounding volumes that contain the scene's subdivided patches. These patch fragments, or *subpatches*, must satisfy a certain criterion in flatness, which can be determined with the opening angle of the surface's normals. The bottom-level, and thus the bulk of the hierarchy, is then made up of compressed and quantized 4-wide BVHs (called CBVHs).

A finely tessellated, quadrilateral surface, to be more specific the resulting *vertex grid*, is the foundation of the two-tier BVH. As seen in Figure 2, subsets of the vertex grid are considered as subpatches and are used to define local coordinate frames. These local frames are aligned to the surface's parameter space and only fully balanced trees are used for CBHVs. Storing the tree's nodes in a linear fashion allows implicit addressing of CBVH nodes. In addition, applying Morton decoding on a node index yields the relative position of the subpatch or leaf node on the surface. Therefore, a leaf's $(u, \quad)$-coordinates, as well as its vertex indices, can be derived implicitly from the leaf's index in the z-curve. According to that, CBVHs consist solely of implicitly ordered bounding volumes and do not require additional information such as pointers, $(u, \quad)$-coordinates, or vertex indices. Consequently, compressing only the bounding volumes has an immense impact on the actual size of the whole BVH.

Projection. The per-CBVH transformation ensures that the vertices of the local subpatch are mostly planar and rectangularly arranged in the local space. As a result, bounding volumes of such vertices are aligned to the subpatch's parameter space. Furthermore, the transitions into these spaces are projective as also trapezoid patches should map to the unit-square in local $(u, \quad)$. This mapping leads to tight bounds for crooked patches, where the former are still axis-aligned, which limits the required information to be stored and is very beneficial for later intersection tests.

Compression. The first element of the compression is an aggressive hierarchical quantization, which stores bounds of subnodes relative to their parent nodes and requires only few bits (e.g., three bits in local x, y and two in z). The second compression aspect is folding of similar bounds. As it can be seen in Figure 3, directly adjacent bounding volumes have very similar bounds. In such cases, storing the more general bound is sufficient and introduces only a small overestimation due to the projective local mapping. The final compression component is reusing the parent's bounding volume in a similar fashion as described for node siblings. Folding of neighboring bounds is conceptually depicted in Figure 4.

Voxelization. Previously shown compression schemes can be used in two ways: to plainly compress the BVH and to also approximate the leaf geometry. In the first case, the leaf-geometry remains as is and only the BVH is compressed, which overall results in low compression rates of $2:1$, but does not compromise on image quality. The impact of storing geometry is limited by efficiently encoding the leaf geometry [Lauterbach et al. 2008; Wald et al. 2014] as a vertex grid and calculating required indices implicitly and ad-hoc. The second case is visualized in Figure 3. As it can be seen, projectively patch-aligned BVHs can envelop the underlying geometry tightly enough that the actual geometry can be omitted. To yield the full potential, the leaf-geometry is dropped entirely and approximated by the local bounding boxes. This yields overall compression rates of up to $16:1$, but at the cost of image fidelity.

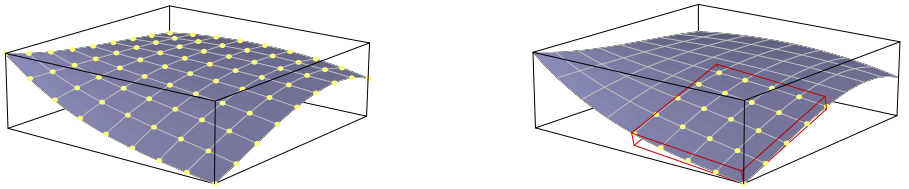


Fig. 2. For almost flat sub-patches, compressed and quantized CBVHs are built and embedded in the standard BVH. The left panel shows a whole patch and its surface vertices. As depicted right, the root of a CBVH is aligned along a subset of vertices of the whole patch and all child-nodes are understood to be in this CBVH-specific frame of reference.

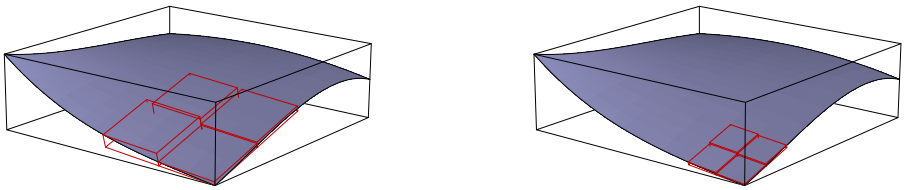


Fig. 3. Local coordinate frames are defined only once per CBVH and subnodes remain in the local frame. As it can be seen in the right panel, leaf-level bounding volumes can be exploited as a voxelized geometry representation.

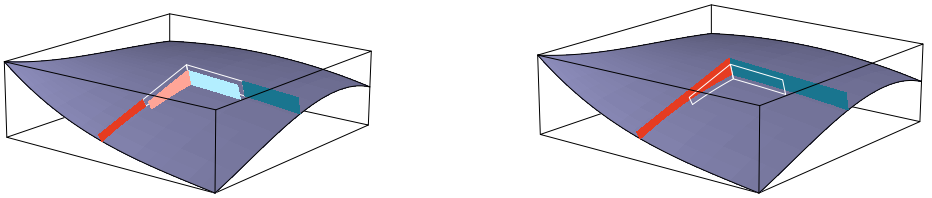


Fig. 4. Inner bounds (left, light colors) can be approximated by more general bound of neighboring bounding volumes (right, dark colors), thus inner bounds can be completely discarded.

Adaptivity. The baseline algorithm [Selgrad et al. 2016] does not tessellate patches adaptively but rather can flexibly adjust the level at which it switches to the local coordinate frame and adapts the height of the BVH to current needs. Even if a specific view requires only a low subdivision level, we favoured maximal tessellation for each patch. Doing this ensures that every single layer of the CBVH is a conservative approximation of the finest tessellation level. This approach enforces a crack-free representation for adjacent bounding volumes, even if they reside on different levels of the CBVH. Since there is no inherent difference between inner and leaf nodes, each node on each level can be directly utilized as a leaf voxel by decoding its $(u, \quad)$-coordinate and span. As a result, switching to a coarser approximation for a specific region requires only pruning the corresponding CBVH and optionally switching earlier to the local frame. With this approach, the number of leaves and thus the overall size of the BVH can be greatly reduced without introducing cracks.

Limitations. The main problem with using the leaf-boxes as geometry approximation is that those boxes are aggressively quantized and compressed on each level of a CBVH down to the leaf. Therefore, even though the bounds are still reasonable for ray culling, they are usually overestimated and, regarding the actual leaf-geometry, poorly aligned. Overestimation follows from repeated quantization and folding of box components, while poor alignment is caused as there is only one transformation stored for each path from the hierarchy's root to each leaf. Thus, the alignment fits well for the interior node for which it is constructed, and provides a better frame of reference for compression, but further subdivision can yield patches that are rotated somewhat with respect to this frame, and will thus be approximated by a non-planar box. Therefore, images rendered

Fig. 5. High tessellation levels are required if the box approximation is used for close-ups with heavily displaced models. The left image shows refinement level 6 with 3 uncompressed levels and the center image shows refinement level 8 with 4 uncompressed levels. The triangle reference for refinement level 6 is shown in the right image.

with this method show good quality when viewed at a certain distance, but the approximation shows when viewed close-up, as displayed in Figure 12, and even breaks if displacement is applied excessively, such as illustrated in Figure 5.

The suggested alternative is utilizing the BVH only in its conventional sense and to preserve the vertex data of the finely tessellated mesh for the ad-hoc reconstruction of triangles during hierarchy traversal (as also suggested by previous approaches [Lauterbach et al. 2008; Wald et al. 2014]). This does not introduce further approximations on top of tessellation but reduces the compression ratio to $2:1$, which, in this case, is a severe decrease.

4 EXTENDED SURFACE APPROXIMATION

The method described in the previous section serves as the foundation for this paper, from which many properties have naturally been adopted. We utilize the same two-tier BVH and make extensive use of the locally oriented lower level. We also exploit its quantized, compressed node structure but avoid the issues described above by introducing a novel, compact leaf representation. Our leaf representation is specifically tailored to be embedded in the leaf voxels of the previous approach [Selgrad et al. 2016]. The additionally required memory is well amortized by the fact that the improved accuracy allows to decrease the depth of the BVH, which has a strong impact on overall memory consumption. In this section, we demonstrate how a better surface approximation can be obtained. Furthermore, we will introduce a custom-made, fast intersection test for our surface approximations with only little demand in extra computation.

Overview. Instead of directly using voxels as a geometry representation we store additional voxel-relative surface information. In short, our approach snaps vertices of the finest tessellation level onto the vertical edges of leaf-level bounding volumes. This allows reusing the bounding volume's x, y values of the corners, thus we only have to store the z value of the snapped vertices. To avoid cracks, we set the reconstructed geometry to a specific thickness that conservatively bounds original vertex positions and suffices as an approximate surface representation. This leaf-level approximation shows improved quality, while still allowing us to fall back to the original adaptive traversal for configurations that call for more coarse approximations. Falling back on the voxelized representation can reduce the memory requirements and does not introduce cracks due to the conservative quality of the voxelized approximation and our leaf representation, which holds true even for boundary vertices shared by both approximations on transition regions.

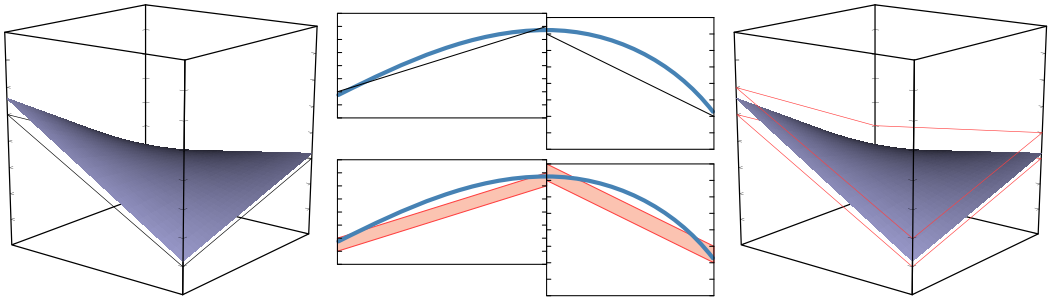
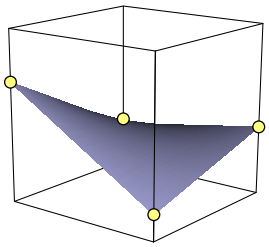


Fig. 6. Using only a single quantized value for approximating the leaf-geometry causes visible cracks (left, center-top). Using intervals prevents cracks (center-bottom, right).

4.1 Vertex Quantization

The bounding box depicted in the inset figure represents a single leaf-level voxel, slightly exaggerated in height for the sake of visual clarity. With the original method [Selgrad et al. 2016] this voxel would be used as geometry approximation (as illustrated in Figure 3). In contrast, instead of dropping the vertices of the underlying surface, we store the z-component of the four vertices enclosed in relation to the bounding voxel. To this end we quantize z at the corners of the bounding box by k bits (3 bits in Figure 6, 4 bits in our renderings). Since a single vertex is covered by four neighboring leaf-level voxels (see Figure 2 and 3), the relative z value must be stored individually for each leaf-node. This circumstance is caused by the fact that each vertex/voxel pair can result in different quantization due to the varying height of the bounding volumes. Despite the need to store each quantized value four times $\left(\frac{4\times 4 \text{ bits}}{8} = 2 \text{ bytes}\right)$, this approach requires less memory than storing each (x, y, z) value only once in full precision (3×4 bytes $= 12$ bytes), which is equal to a compression factor of 6 for the underlying geometry.



Flat Voxelization. Using a single quantized, box-relative value per x, y-corner introduces a grave potential for cracks in the approximation. Figure 6 (center-top) illustrates this in 2D: With shifted boxes it is very unlikely that the quantized edges coincide exactly. To counter this, we instead use intervals at each x, y-corner to provide a form of geometry-aligned voxelization on a per-leaf basis. As the limit surface of the patch smoothly transitions from one box to its neighbor, and therefore the limit vertex in-between is conservatively enclosed by both their bounding volumes, it is sufficient to set the thickness of these intervals to one bin. Utilizing the described flat interval-based voxels avoids cracks inside a CBVH and, as this allows us to only store the lower bound, it is free in terms of memory. Note that, for patches on CBVH-borders, neighboring boxes can also be rotated differently, but due to the conservative nature of the approximation, our method works well in practice and we did not notice cracks even in extreme cases. Still, being conservative requires an appropriate mapping of possibly off-bounds shifted vertices to corresponding z-intervals on the x, y-corners.

Extrapolation. During hierarchy construction, the leaf-patch can be found to not fully cover the x, y-range of a leaf box, leading to leaf-level vertices not coinciding exactly with their voxel's x, y-bounds. Figure 7 (left) illustrates such a configuration. This can be due to repeated overestimation during box quantization and compression, but also due to non-trapezoidal bending of the underlying surface, since only strictly trapezoidal patches can be captured exactly with a perspective projection to the local frame. Simply stretching the patch to cover the entire range naturally adds distortion,

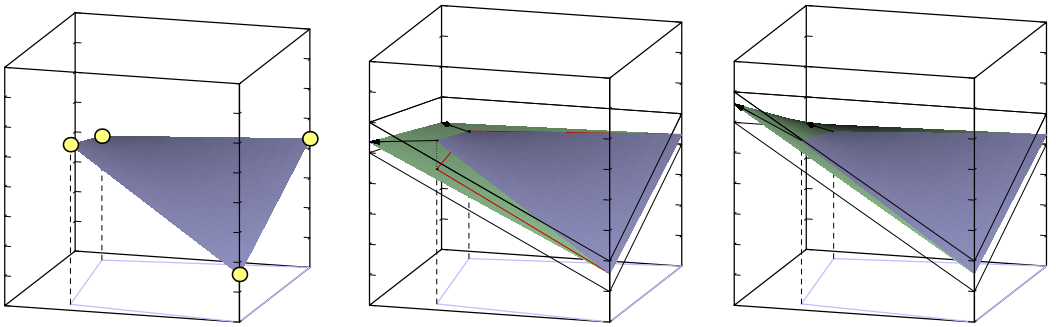


Fig. 7. Illustration of the orientation and position of the patch within the bounding volume (left). Directly utilizing a vertex's z-values leads to defective approximations (center). The solution is extrapolating the z-value onto the box-corners, prior to quantization and storing (right).

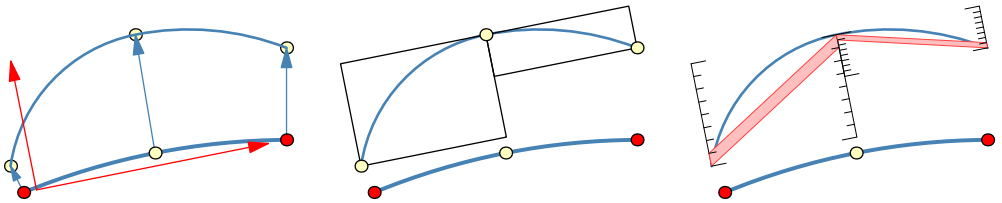


Fig. 8. The local coordinate frame, depicted with red arrows in the left image, is aligned to the non-displaced surface vertices highlighted in red. Such a setup leads to larger bounding volumes, as seen in the center, but the displacement primarily affects the height of the bounding volumes, which perfectly maps to our additional surface approximation, as can be seen in the right image.

as demonstrated with the green surface in Figure 7 (center). Even worse, with that approach it cannot be guaranteed that the patch approximation still contains the stretched vertices' original positions. A likewise intuitive solution is extrapolating the linear patch prior to quantization, as shown in Figure 7 (right), thus overestimating the surface for such configurations. This mapping can, however, result in the new patch's z-bounds leaving the original leaf-box. Nevertheless, we extend the z-range escaping the leaf-box but limit intersections to be valid only within the actual bounding volume. The possible extent is determined and defined on a per-CBVH basis, i.e. adaptive per aligned sub-tree. Therefore, we provide high-resolution quantization for patches not needing the extra extent, but we avoid heavy distortion and preserve conservative bounds for most cases that do. In practice, we limit the z-scaling to three times the height of the local box.

Frame Alignment. An additional aspect we identified as advantagous is changing the basis on which local coordinate frames are aligned. In contrast to the earlier method, we align the local system along non-displaced patch vertices. This can lead to looser bounds for the leaf nodes [Selgrad et al. 2016], but this is not critical with our approximation, as illustrated in 2D in Figure 8. In fact, the adjusted alignment is even beneficial, since the variation introduced via displacement affects mainly to the local z-component, which matches well with our interpretation of a patch varying mostly in z and can easily be captured with our approximation.

4.2 Intersection

An additional benefit of our patch approximation is the fact that it can be efficiently tested for ray intersections. Naïvely searching for intersections with our patch approximation requires us to test multiple surfaces. In that case, we have to compute intersection tests for at least 4 quads and 2

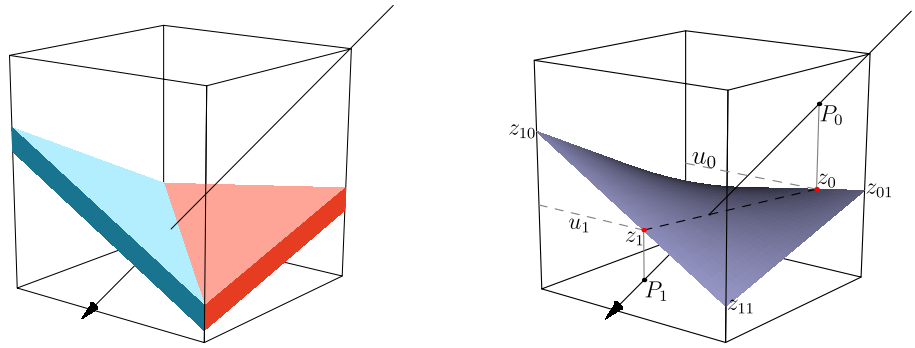


Fig. 9. A naïve approach requires intersection tests with multiple individual surfaces, e.g. 4 triangles and 4 quads (left) or 2 bilinear patches and 4 quads. For our approximation, one single 2D line intersection test is sufficient (right).

bilinear patches, or alternatively 4 triangles as depicted in Figure 9. However, these approaches do not take into account that the entry and exit point of the surrounding bounding box are known during traversal. Exploiting this fact, our approximation can be tested for intersection with only one single 2D line intersection test. To this end, we use an approximate intersection test that projects the ray onto a straight line on the surface. The line is constructed from the entry and exit points of the bounding box projected to surface points (see Figure 9). This approach yields the same results as bilinear-patch intersections if the ray is exactly aligned to the x or y axis. Results for misaligned rays differ since saddle-shaped surfaces cannot be represented faithfully. Still, the approximation is exact at patch borders, crack-free within the surface, and neither bilinear patches nor triangles are as simple and lightweight regarding the computation of intersections.

Thus, after initial parameter finding, the ray-leaf intersection reduces to a 2D line intersection in the ray/z_0, z_1-plane, as depicted in Figure 9 (right). For the test we rely on the ray parameters of the leaf's bounding box intersection, t_0 and t_1, as well as the leaf's box values, $B_{\min}$ and $B_{\max}$. The entry and exit positions, P_0 and P_1, can be computed from the ray and the parameter values above. Note that all of these are defined and used in the local, patch-aligned frame. Then, the $(u, \quad)$-parameters at the ray's entry and exit point on the box are:

$$u_i = \frac{(P_{i,x} - B_{\min,x})}{(B_{\max,x} - B_{\min,x})} \qquad {}_i = \frac{(P_{i,y} - B_{\min,y})}{(B_{\max,y} - B_{\min,y})}$$

With these, the z value of the approximated patch at the entry and exit $(u, \quad)$-parameters can be computed via linear interpolation of the restored values at the box's x, y-corners, $z_{00}, z_{01}, z_{10}, z_{11}$:

$$z_i = z_{00}(1-u_i)(1- \ {}_i) + z_{10}u_i(1- \ {}_i) + z_{01}(1-u_i) \ {}_i + z_{11}u_i \ {}_i$$

This gives the line that the linear patch describes along the ray, and thus the ray intersects the patch if it intersects this line.

As we approximate bilinear surfaces with the upper and lower bounds of the leaf geometry, we distinguish between three cases for intersecting leaves. If the ray's z-value at the box-entry is above the patch, $P_{0,z} > z_0 + \Delta$, where Δ is the bin-size (we only store the lower bounds), we use the upper bound for intersection testing. In the mirrored case, $P_{0,z} < z_0$, we intersect the lower bounds with the ray. Finally, if $z_0 \leqslant P_{0,z} \leqslant z_0 + \Delta$ we know that there is an intersection at the ray's entry point on the box.

Using the entry point's distance t_0, the point's z-value $P_{0,z}$, and the surface's interpolated z-value z_0 (for the exit point t_1, $P_{1,z}$, and z_1) the distance t to the actual hit point on the surface can be

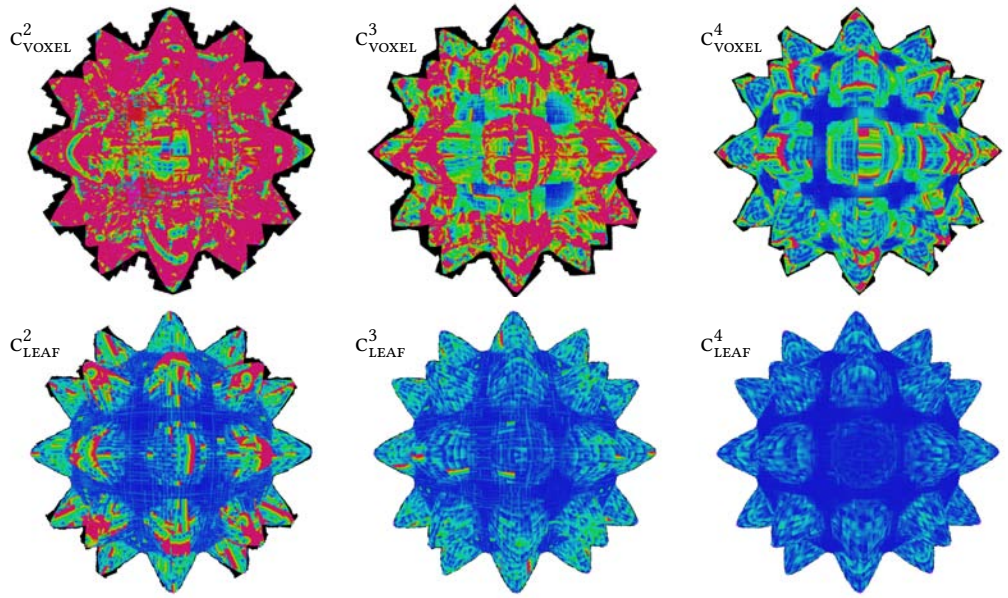
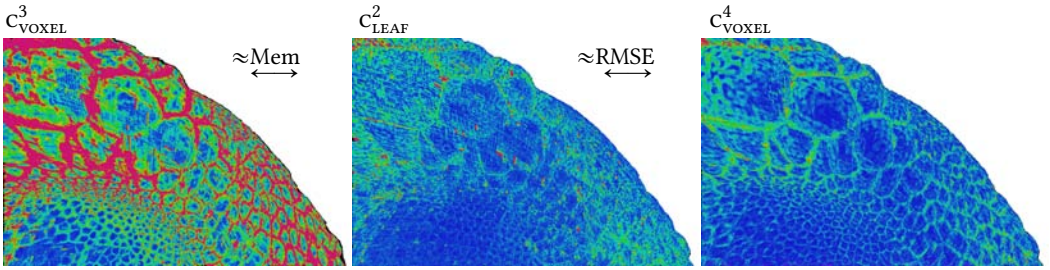


Fig. 10. Hit-point difference in pixels with respect to a triangle mesh of the same subdivision level. The top row shows cubical patch approximations and the lower row shows our new patch approximations. All figures use a subdivision of level 6 but switch to the locally oriented frame on different levels. From left to right we switched after level 2, 3, and 4 to the local frame. Error ranges from blue (no error), pink (distance greater than 5 pixels), and black (false-positive hit).



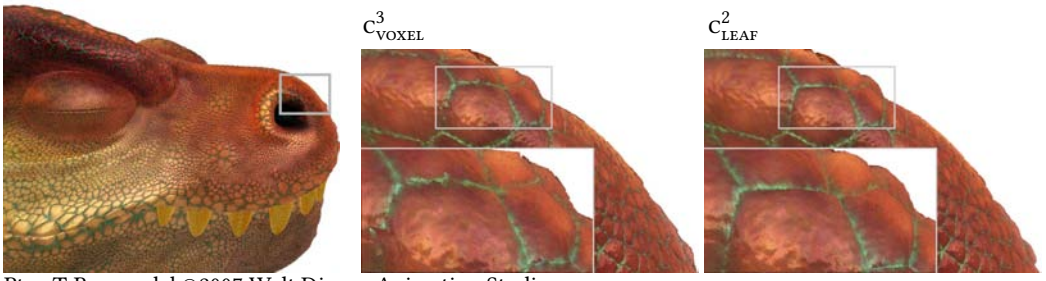
Ptex T-Rex model ©2007 Walt Disney Animation Studios

Fig. 11. Hit-point difference in pixels with respect to a finely tessellated mesh (shown is the T-rex's nose, see Figure 12). A subdivision level of 6 was apply for c^3_{VOXEL} (left), c^2_{LEAF} (center) and subdivision level 8 was used for c^4_{VOXEL} (right). The overall RMSE from left to right is 0.146, 0.064, and 0.093. Note the visual comparison of c^3_{VOXEL} to c^4_{VOXEL} shown in Figure 5.

evaluated utilizing a simple 2D-line intersection test. The calculations for the hit point's z-value can be omitted, since we only require the t component of the line intersection:

$$t = \frac{t_0(P_{1,z} - z_1) + t_1(z_0 - P_{0,z})}{(P_{1,z} - z_1) + (z_0 - P_{0,z})}$$

Testing only once per patch applying this intersection scheme is considerably more lightweight than identifying the intersection on each face of our patch approximation individually.



Ptex T-Rex model ©2007 Walt Disney Animation Studios

Fig. 12. The center panel above shows that the silhouette of the subdivided and displaced patches of the T-rex's nose are visibly approximated, and that inner regions can become distorted using c^3_{VOXEL}. Those artifacts are not visible with c^2_{LEAF} (right panel), both refined to level 6. The visible differences are minor, but being able to transition to local CBVHs one level sooner, the c^2_{LEAF} version requires 94 MiB for the entire model (101.4 MiB with c^3_{VOXEL}). Considering this, and that traversal times are very similar (422 versus 439 millions of rays per second, respectively), the increase of quality (RMSE 0.064 for c^2_{LEAF} versus 0.146 for c^3_{VOXEL}) seems worthwhile. See also the equal-memory comparison shown in Figure 11 (left/center).

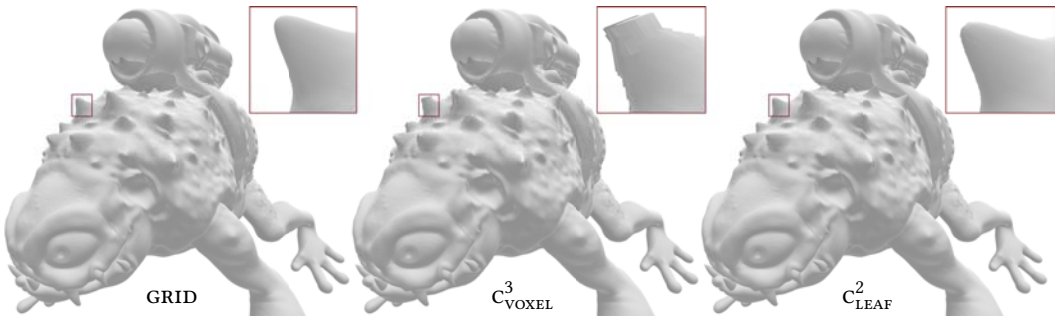


Fig. 13. While the Monster Frog on subdivision level 6 can be represented nicely with the voxelization (center, c^3_{VOXEL}) compared to the reference (left, GRID), the former is not sufficient to cover highly displaced regions, e.g., individual spikes, on that subdivision level. Our representation does not exhibit such limitations (right, c^2_{LEAF}) despite using the same subdivision level and switching to CBVHs one level sooner.

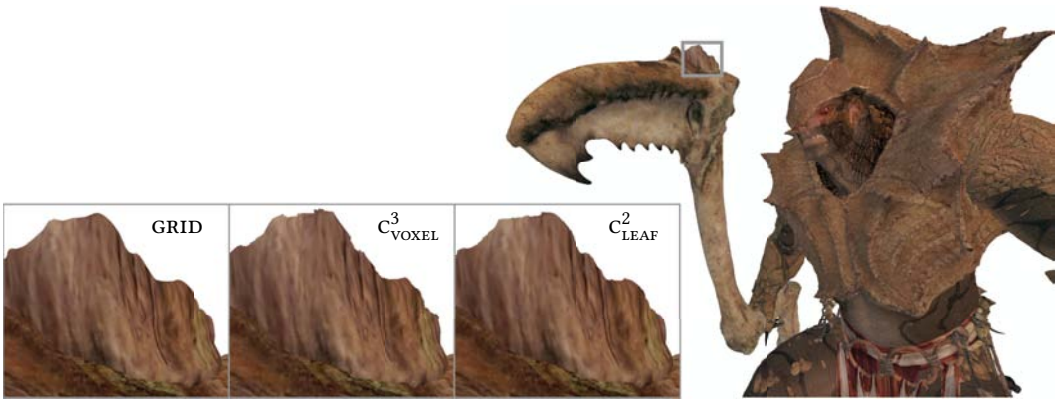


Fig. 14. Challenging regions are composed of rather large displaced patches. Applying 6 subdivision steps, our approach (right, c^2_{LEAF}) is not able to cover the tip of the scythe perfectly but represents the actual silhouette more faithfully than the voxelization (center, c^3_{VOXEL}) compared to the reference (left, GRID).

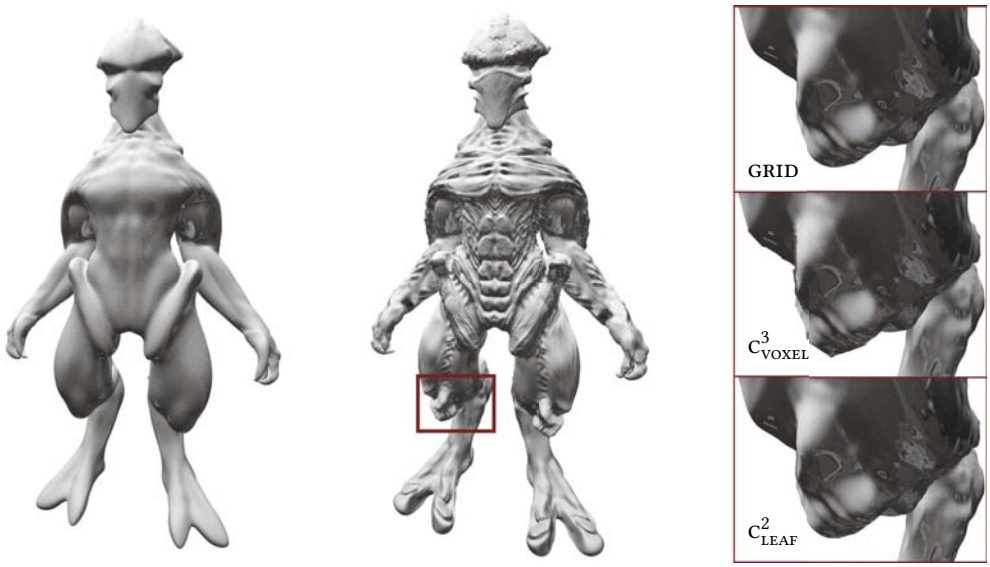


Fig. 15. Our approach benefits from our non-displaced frame alignment especially for high-curvature patches, e.g, the knee. The voxelization (c^3_{VOXEL}) applies coordinate frames aligned to the displaced geometry (center) and approximates patches with flat surfaces. This results in individually aligned but flat areas on the knee. Coordinate frames used in our method (c^2_{LEAF}) are aligned with the not-displaced geometry (left) and displacement is handled wholly by the patch approximation. Compared to the reference (GRID) our approach offers a visually comparably smooth curvature.

5 EVALUATION

In this section, we evaluate our method in context of previous approaches, all of which are implemented for GPU and CPU ray tracing. The former are implemented according to Aila and Laine's method [Aila and Laine 2009], while the latter are integrated in Embree [Wald et al. 2014].

Overall we compare the following methods:

NAIVE A full-precision, memory intensive BVH according to the state of the art in fast ray traversal [Aila and Laine 2009; Wald et al. 2014].

GRID A BVH using a cleverly packed, full-precision, lossless, grid-based leaf-layout [Lauterbach et al. 2008], specifically tailored to subdivision patches [Wald et al. 2014].

$\mathbf{c^n_{GRID}}$ A strongly compressed BVH (with CBHVs starting from level n) [Selgrad et al. 2016] using a full-precision grid-based layout for the leaf geometry (as with GRID).

$\mathbf{c^n_{VOXEL}}$ The same CBVH-based layout where leaf geometry is approximated by the compressed leaf bounds, only [Selgrad et al. 2016].

$\mathbf{c^n_{LEAF}}$ Our CBVH-based method with our novel approximation for leaf geometry as described in Section 4.

Image Quality. Our novel leaf approximation shows much better results than previous leaf-voxel approximations [Selgrad et al. 2016]. Rendering errors for our teaser image, shown in Figure 1, are displayed in Figure 10. It depicts the error progression of hit-point differences with increasing subdivision level at which we move to the CBVH, while keeping the maximal refinement level capped to 6. Intersection points originating from c^n_{VOXEL} (top) and c^n_{LEAF} (bottom) are compared to intersection points obtained from a triangle mesh of the same tessellation level with their distance normalized to pixel footprint at the given intersection depth. The figure shows that our new method

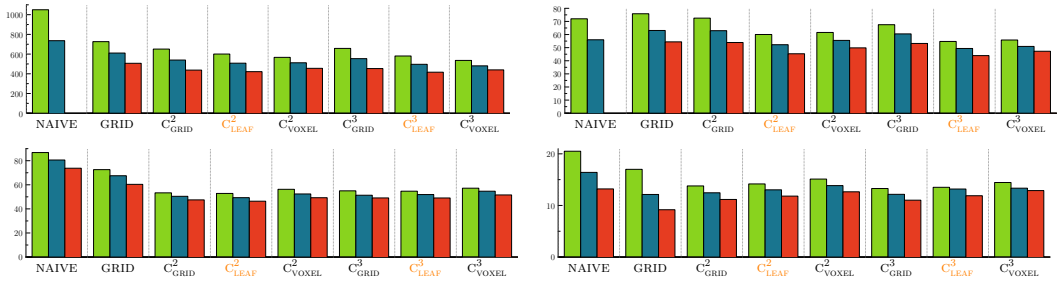


Fig. 16. Performance of coherent (left) and incoherent (right) ray traversal in millions of rays per second at subdivision levels 4 (green), 5 (blue), and 6 (red), on a GPU (Nvidia GeForce GTX 1070, top) and on a CPU (Intel Core I7-6700, bottom), times taken on the full T-rex model (see Figure 12). The missing entry in the top row indicates that the naïve version was too large to be stored on the GPU.

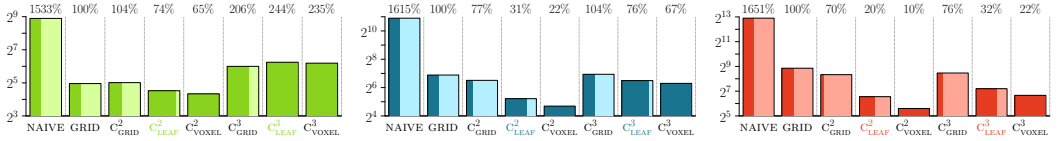


Fig. 17. Memory requirements (in MiB) for subdivision levels 4 (green), 5 (blue), and 6 (red) for the T-rex model shown in Figure 12. Note the log-scale and different maximum per plot. The darker part of each bar represents the memory required for the inner nodes, the lighter part indicates the fraction of memory required for the leaf geometry. With higher subdivision levels the fraction of leaf geometry increases much faster for c_{GRID} than for our new representation c_{LEAF}. The percentages given at the top are relative to state of the art lossless compression [Lauterbach et al. 2008; Wald et al. 2014], i.e. GRID.

results in significantly lower rendering error when compared to the plain voxelization with a CVBH of equal height.

Furthermore, the comparison demonstrates that the higher precision of our leaf-approximation warrants earlier alignment, even more so as it still results in a lower error. This is nicely exemplified by comparing the bottom left image, c^2_{LEAF}, with the top right one, c^4_{VOXEL}. Therefore, our new method enables us to switch to the local frame sooner, meaning that it is sufficient for the patch approximation to rely on more coarsely aligned frames, resulting in a much lower overall memory footprint. Alternatively, its application can improve the visual quality compared to leaf-voxel approximations [Selgrad et al. 2016] while not increasing memory consumption.

Figure 11 illustrates the rendering error for a production model that is more well-behaved than the example shown in Figure 10. Even in this case our novel method, c^2_{LEAF}, produces considerably lower error than previous methods at similar memory requirements as c^3_{VOXEL} (left vs center). Targeting a similar (but still inferior) quality, c^4_{VOXEL} is required, diminishing the compression by c_{VOXEL}. Figure 12 highlights the differences that appear when viewed close-up. Note that we do not further consider c^4_{VOXEL} and c^4_{LEAF}, since storing coordinate frames at such high subdivision levels renders the compression almost useless for practical subdivision levels, due to the heavily increased memory requirements.

Figures 13, 14, and 15 show various models on subdivision level 6 and highlight individual cases, e.g., strong displacement, large patches, high curvature, where applying c^2_{LEAF} over using c^3_{VOXEL} highlights the improved quality.

Rendering Performance. Rendering performance is at a maximum when using the NAIVE BVH representation with full-precision leaf nodes. This is to be expected as compression schemes generally introduce decompression overhead. However, the missing entries in the top-row of Figure 16 already suggest that this approach is severely limited by memory consumption. At some

point, slower storage (e.g. falling back to the CPU, or swapping of main memory) will result in much inferior performance [Yoon et al. 2006]. In the following, listed methods will be annotated with min%-max%, ∅ avg%, indicating the minimum, maximum, and average performance normalized to GRID.

Of the grid-based methods, the standard GRID generally shows faster rendering times than further compressed versions. c_{GRID} are CBVHs that keep the grid-based leaf-level and show the lowest performance [65%-99%, ∅84%] because two full triangle intersections have to be computed for each leaf hit. Dropping the triangle tests entirely by using c_{VOXEL} provides better performance [73%-105%, ∅86%], however at the cost of rendering quality.

Our new method, c_{LEAF} [69%-92%, ∅80%], performs very close to c_{GRID} [∅84%] and c_{VOXEL} [∅86%]. It should be noted, however, that c^2_{GRID} consumes considerably more memory than c^2_{LEAF} on realistic subdivision levels, as shown in Figure 17. Furthermore, to achieve similar image quality to c^2_{LEAF} on subdivision level 6 using the voxel-based approximation, c^4_{VOXEL} should be used but still requires subdivision level 8 (see Figure 11). This version, at the given refinement level, however, performs [64%-72%, ∅68%] inferior to c^2_{LEAF} [∅80%] and requires much more memory since the shift to CBVHs is delayed, as well as due to the high refinement level. Overall, this suggests that the increase in image quality of c_{LEAF} will not come at an increased rendering time when compared to methods with similarly high compression rates, especially under strong displacement or for close-up views.

Memory Requirements. As described above, our approach offers a reduction in rendering error compared to the voxelized baseline approach. One result of using more information on the leaf-level is that our structure is larger than c_{VOXEL}, which does not store leaf information, while we store 2 bytes per leaf. Figure 17 shows the memory consumption of different BVH variants grouped by overall subdivision level. For very low levels, GRID works very well: the compressed variants cannot amortize the overhead of storing transformations. Medium and higher levels, however, are much more compact with c^2_{VOXEL}, only taking 10%, and c^2_{LEAF}, which is only 20% of GRID. The differently shaded areas in each bar give the relative memory for inner nodes and leaves for each variant. This shows that, even though the inner nodes for, e.g. c^3_{GRID} and c^3_{LEAF} are of the same size, the leaf-portion of c^3_{GRID} is larger.

However, the relationships illustrated in Figure 17 are only true when we utilize exactly the same CBVHs, meaning that we switch to the local frame on the same level and compute the same number of refinement levels, such as level 6 with c^3_{LEAF} and c^3_{VOXEL}. Based on the improved overall approximation quality demonstrated above, we can, however, choose a lower level for the transition into local frames (c^2_{LEAF} vs. c^3_{VOXEL}).

Even more importantly, using our higher-quality representation we can limit overall refinement to lower levels. With leaf-voxelization, this is problematic as the underlying structure will become apparent, while our representation produces more faithful results. This reduction is not in the order of multiplying a constant to the number of leaf nodes but can lead to generating exponentially fewer leaves in the first place. As an example, consider the comparison shown in Figure 11. To achieve the image quality provided by our novel c^2_{LEAF} at subdivision level 6, the leaf-voxelization c_{VOXEL} has to be built on subdivision level 8 with a transition into local frames after half of those, i.e. c^4_{VOXEL}. This, however, results in worse rendering times (see above) and totals to 555 MiB of memory, as compared to c^2_{LEAF}'s 94 MiB. A similar example can be found in Figure 12.

Full tabulation of all measurements for the T-rex (Figure 12), Barbarian (Figure 14), Monster Frog (Figure 13), and the Alien (Figure 15) can be found in the supplemental material. It covers ray-tracing performance and memory requirements on the CPU and GPU for all considered BVHs, leaf types, and ray queries. Furthermore, our implementation is available online[1].

[1] https://github.com/lispbub/embree-compressed

Fig. 18. The left panel shows our patch approximation with 3 compressed levels and the right panel shows a triangle representation. Both show a subdivision level of 6 and a fivefold displacement compared to Figure 1.

Displacement. The limitations of our approach become apparent when displacement exceeds the point of being applied for adding surface structure with rather moderate amplitudes but is used for large-scale deformation of the geometry. While the limitations of the voxelized approximation are already depicted in Figure 1, the limitations of our approach become apparent only with heavily increased displacement scale. Figure 18 shows a comparison between our approach and a triangle representation with fivefold displacement compared to the teaser. However, we find such applications rather unlikely, since even the triangle representations starts to lose geometric fidelity at such high displacement levels. Increasing the subdivision level would naturally ease this situation for the triangle representation as well as for our approximation, but at the cost of increased memory requirements.

6 CONCLUSION

In this paper we have presented a very simple method to obtain higher quality renderings from an existing compressed representation for subdivision surfaces. We have shown that at the same render times and memory consumption, our leaf approximation generates images more faithful to the underlying surface. This is especially noticeable at silhouettes, where more coarse approximations reveal their structure, but also for close-up views and strong displacement.

Another way to look at these results is that we can achieve image quality similar to, e.g. leaf-voxelization, with earlier projection into the local frame or even by not subdividing to similarly high levels. The latter maps to an even more pronounced decrease in memory consumption, as even the most compact representation will become intolerably large when requiring high subdivision levels, which can be avoided using our leaf-representation.

ACKNOWLEDGMENTS

The authors would like to thank the Walt Disney Animation Studios for giving access to the T-rex model and its detailed textures, Bay Raitt for the Monster Frog model, Autodesk for providing the Barbarian model, and Aidlez R. for the Alien model. We also gratefully acknowledge the generous funding by the German Research Foundation (GRK 1773).

REFERENCES

Oliver Abert, Markus Geimer, and Stefan Müller. 2006. Direct and Fast Ray Tracing of NURBS Surfaces. *IEEE Symposium on Interactive Ray Tracing 2006* (2006), 161–168.

Attila T. Áfra. 2012. Interactive Ray Tracing of Large Models Using Voxel Hierarchies. *Computer Graphics Forum* 31, 1 (2012), 75–88.

Timo Aila and Samuli Laine. 2009. Understanding the Efficiency of Ray Traversal on GPUs. *Proceedings of the conference on high performance graphics 2009* (2009), 145–149.

Pablo Bauszat, Martin Eisemann, and Marcus A Magnor. 2010. The Minimal Bounding Volume Hierarchy. *Vision, Modeling, and Visualization* (2010), 227–234.

Carsten Benthin, Sven Woop, Matthias Nießner, Kai Selgrad, and Ingo Wald. 2015. Efficient Ray Tracing of Subdivision Surfaces using Tessellation Caching. In *Proceedings of the 7th High-Performance Graphics Conference*. ACM.

Edwin Catmull and James Clark. 1978. Recursively Generated B-Spline Surfaces on Arbitrary Topological Meshes. *Computer-aided design* 10, 6 (1978), 350–355.

Per H. Christensen and Dana Batali. 2004. An Irradiance Atlas for Global Illumination in Complex Production Scenes. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques (EGSR'04)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 133–141.

Per H. Christensen, Julian Fong, David M Laur, and Dana Batali. 2006. Ray Tracing for the Movie 'Cars'. *IEEE Symposium on Interactive Ray Tracing 2006* (2006), 1–6.

Per H. Christensen, David M Laur, Julia Fong, Wayne L Wooten, and Dana Batali. 2003. Ray Differentials and Multiresolution Geometry Caching for Distribution Ray Tracing in Complex Scenes. *Computer Graphics Forum* 22, 3 (2003), 543–552.

Cyril Crassin. 2011. *GigaVoxels (a Voxel-Based Rendering Pipeline for Efficient Exploration of Large and Detailed Scenes)*. Ph.D. Dissertation. Universite de Grenoble.

Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. 2011. Interactive Indirect Illumination Using Voxel Cone Tracing. *Computer Graphics Forum* 30, 7 (2011), 1921–1930.

Tony DeRose, Michael Kass, and Tien Truong. 1998. Subdivision Surfaces in Character Animation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. ACM, New York, NY, USA, 85–94.

Markus Geimer and Oliver Abert. 2005. Interactive Ray Tracing of Trimmed Bicubic Bézier Surfaces without Triangulation. *WSCG 2005 Conference Proceedings* (2005), 71–78.

Johannes Hanika, Alexander Keller, and Hendrik P. A. Lensch. 2010. Two-level Ray Tracing with Reordering for Highly Complex Scenes. In *Proceedings of Graphics Interface 2010 (GI '10)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 145–152.

Erik Hubo, Tom Mertens, Tom Haber, and Philippe Bekaert. 2006. The Quantized kd-Tree: Efficient Ray Tracing of Compressed Point Clouds. *IEEE Symposium on Interactive Ray Tracing 2006* (2006), 105–113.

James T Kajiya. 1982. Ray Tracing Parametric Patches. *ACM SIGGRAPH Computer Graphics* 16, 3 (1982).

Tae-Joon Kim, Yongyoung Byun, Yongjin Kim, Bochang Moon, Seungyong Lee, and Sung-Eui Yoon. 2010a. HCCMeshes: Hierarchical-Culling Oriented Compact Meshes. *Computer Graphics Forum* 29, 2 (2010), 299–308.

Tae-Joon Kim, Bochang Moon, Duksu Kim, and Sung-Eui Yoon. 2010b. RACBVHs: Random-Accessible Compressed Bounding Volume Hierarchies. *IEEE Transactions on Visualization and Computer Graphics* 16, 2 (2010), 273–286.

Dylan Lacewell, Brent Burley, Solomon Boulos, and Peter Shirley. 2008. Raytracing prefiltered occlusion for aggregate geometry. In *2008 IEEE Symposium on Interactive Ray Tracing*. 19–26.

Samuli Laine and Tero Karras. 2010. Efficient Sparse Voxel Octrees. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '10)*. ACM, New York, NY, USA, 55–63.

Christian Lauterbach, Sung-Eui Yoon, and Dinesh Manocha. 2007. Ray-Strips: A Compact Mesh Representation for Interactive Ray Tracing. In *2007 IEEE Symposium on Interactive Ray Tracing*. 19–26.

Christian Lauterbach, Sung-eui Yoon, Ming Tang, and Dinesh Manocha. 2008. ReduceM: Interactive and Memory Efficient Ray Tracing of Large Models. *Computer Graphics Forum* 27, 4 (2008), 1313–1321.

Jeffrey A Mahovsky. 2005. *Ray Tracing with Reduced-Precision Bounding Volume Hierarchies*. Ph.D. Dissertation. University of Calgary.

Matthias Nießner and Charles Loop. 2013. Analytic Displacement Mapping Using Hardware Tessellation. *ACM Transactions on Graphics (TOG)* 32, 3 (2013), 26.

Jan Novák and Carsten Dachsbacher. 2012. Rasterized Bounding Volume Hierarchies. *Computer Graphics Forum* 31, 2 (2012), 403–412.

Fábio Policarpo and Manuel M. Oliveira. 2006. Relief Mapping of Non-height-field Surface Details. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games (I3D '06)*. ACM, New York, NY, USA, 55–62.

Fábio Policarpo, Manuel M. Oliveira, and João L. D. Comba. 2005. Real-time Relief Mapping on Arbitrary Polygonal Surfaces. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games (I3D '05)*. ACM, New York, NY, USA, 155–162.

Szymon Rusinkiewicz and Marc Levoy. 2000. QSplat: A Multiresolution Point Rendering System for Large Meshes. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), 343–352.

Benjamin Segovia and Manfred Ernst. 2010. Memory Efficient Ray Tracing with Hierarchical Mesh Quantization. *Proceedings of Graphics Interface 2010* (2010), 153–160.

Kai Selgrad, Alexander Lier, Magdalena Prus, Christoph Buchenau, Michael Guthe, Franziska Bertelshofer, Henry SchÃfer, and Marc Stamminger. 2016. A Compressed Representation for Ray Tracing Parametric Surfaces. *ACM Transactions on Graphics (TOG)* (2016), 5:1–5:13.

Takahito Tejima, Masahiro Fujita, and Toru Matsuoka. 2015. Direct Ray Tracing of Full-Featured Subdivision Surfaces with Bézier Clipping. *Journal of Computer Graphics Techniques (JCGT)* 4, 1 (2015), 69–83.

Ingo Wald, Sven Woop, Carsten Benthin, Gregory S Johnson, and Manfred Ernst. 2014. Embree–A Ray Tracing Kernel Framework for Efficient CPU Ray Tracing. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)* (2014).

Sung-Eui Yoon, Christian Lauterbach, and Dinesh Manocha. 2006. R-LODs: Fast LOD-Based Ray Tracing of Massive Models. *The Visual Computer* 22, 9-11 (2006), 772–784.