Raytracing Prefiltered Occlusion for Aggregate Geometry

Dylan Lacewell[1,2] Brent Burley[1] Solomon Boulos[3] Peter Shirley[4,2]

[1] Walt Disney Animation Studios [2] University of Utah [3] Stanford University [4] NVIDIA Corporation

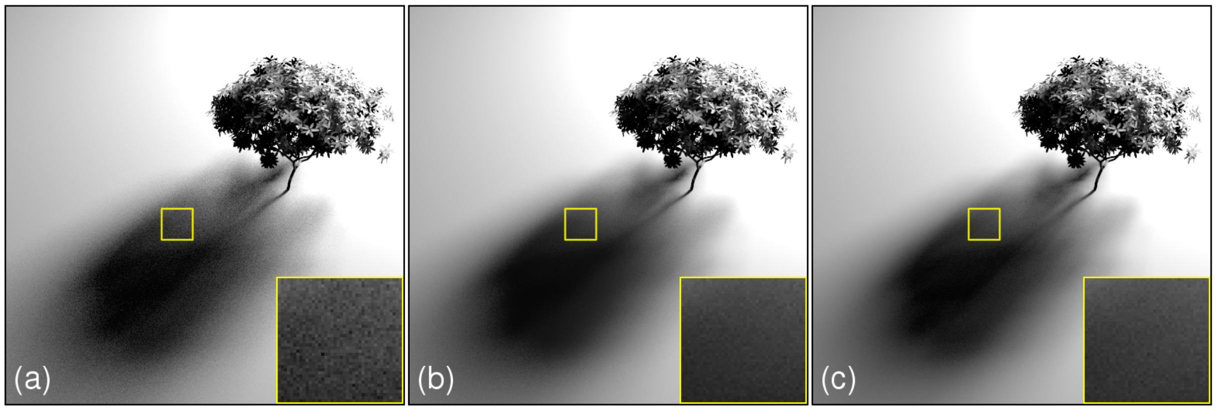

Figure 1: *Computing shadows using a prefiltered BVH is more efficient than using an ordinary BVH. (a) Using an ordinary BVH with* 4 *shadow rays per shading point requires* 112 *seconds for shadow rays, and produces significant visible noise. (b) Using a prefiltered BVH with* 9 *shadow rays requires* 74 *seconds, and visible noise is decreased. (c) Reducing noise to a similar level with an ordinary BVH requires* 25 *shadow rays and* 704 *seconds (about* $9.5\times$ *slower). All images use* 5×5 *samples per pixel. The scene consists of about* $2M$ *triangles, each of which is semi-opaque (*$\alpha = 0.85$*) to shadow rays.*

Abstract

We prefilter occlusion of aggregate geometry, e.g., foliage or hair, storing local occlusion as a directional opacity in each node of a bounding volume hierarchy (BVH). During intersection, we terminate rays early at BVH nodes based on ray differential, and composite the stored opacities. This makes intersection cost independent of geometric complexity for rays with large differentials, and simultaneously reduces the variance of occlusion estimates. These two algorithmic improvements result in significant performance gains for soft shadows and ambient occlusion. The prefiltered opacity data depends only on geometry, not lights, and can be computed in linear time based on assumptions about the statistics of aggregate geometry.

1 Introduction

Soft shadows and ambient occlusion are important tools for realistic shading in film production. However, these effects are expensive because they require integration of global visibility at each shading point. In a ray-based renderer, many shadow rays per shading point may be needed to accurately estimate visibility without noise artifacts. Variance is especially a problem with *aggregate* geometry, i.e., geometry consisting of a many small, disconnected parts, each of which may not be fully opaque. Examples frequently occurring in production include hair and foliage.

In this paper we show how to reduce the cost of visibility queries for aggregate geometry by raytracing against a modified bounding volume hierarchy (BVH) in which we store a prefiltered opacity at each node, representing the combined visibility of all geometry contained in the node. Prefiltering requires a modest increase in the time to build or update a BVH; we demonstrate empirically that for aggregate geometry, it suffices to use a prefiltering algorithm that is linear in the number of nodes in the BVH. Once built, a prefiltered BVH does not need to be updated unless geometry changes.

During rendering, we terminate shadow rays at some level of the BVH, dependent on the differential [10] of each ray, and return the stored opacity at the node. The combined opacity of the ray is computed by compositing the opacities of one or more nodes that the ray intersects, in any order. This early termination eliminates many ray-triangle intersections, and makes intersection cost independent of geometric complexity for rays with large enough differentials. In addition, prefiltering also reduces the variance of occlusion estimates on aggregate geometry, which in turn reduces noise artifacts for a given number of shadow rays. Both benefits are demonstrated in Figure 1.

The original contributions in this paper are as follows:

- We prefilter directional opacity; previous work focused on prefiltering color.
- We present a simple linear-time opacity prefiltering algorithm based on assumptions about the statistics of aggregate geometry.
- We store prefiltered data in a BVH, a popular sparse acceleration structure that can be updated in linear time for many types of dynamic scenes [25].
- We show empirically that prefiltering reduces the cost of computing soft shadows and ambient occlusion.

A note about terminology: we use *opacity* or *occlusion* interchangeably to mean the percentage of light that is blocked between two points, and *visibility* as the complement of occlusion.

2 Background

We discuss only the most relevant work on visibility queries. We refer the reader to the survey of shadow algorithms by Woo et al. [29], or the more recent survey of real-time algorithms by Hasenfratz et al. [7].

2.1 Raytracing

Distribution raytracing [5] is a general rendering method for many global illumination effects, including shadows, and can be used with prefiltered scenes. Mitchell [15] provided insights into how the convergence of raytracing depends on the variance of the scene. Variants of ray tracing such as packets [26], cone tracing [1], beam tracing [8, 17], and the shadow volumes method of Laine et al. [13] exploit geometry coherence to speed up intersection or reduce variance; these methods are less effective for aggregate geometry with small pieces but would benefit from prefiltered geometry.

Precomputed radiance transfer [24] stores a raytraced reference solution at each shading point, in a basis that allows environment lighting at an infinite distance to be updated dynamically. True area lights at a close distance are not easily accounted for.

2.2 Point Hierarchies

Many previous methods have estimated scene visibility using multi-resolution hierarchies of points, spheres, or other primitives. Very few point-based methods prefilter opacity, and most are demonstrated on solid, connected geometry.

Bunnell [3] uses a hierarchy of disks, where disk sizes and positions are averaged, but changes in opacity due to empty space are ignored. Multiple rendering passes are required to correctly handle overlapping occluders. Pixar's Photorealistic Renderman (PRMan) [18] appears to use a similar method. Wand [27] renders soft shadows by cone tracing a point hierarchy, but does not handle opacity changes due to empty space, and observes that shadows for foliage are overly dark. Ren et al. [20] approximate geometry with sets of opaque spheres, and splat spheres using a spherical harmonic technique that corrects for occluder overlap in a single pass. No solution for opacity filtering is given. Kontkanen and Lane [12] compute ambient occlusion by splatting opaque objects using precomputed visibility fields. Kautz et al. [11] rasterize scene visibility into a small buffer at each shading point; this is conceptually similar to point-based occlusion, but mesh simplification is used rather than a point hierarchy. Mesh simplification does not work well for aggregate geometry.

A number of previous methods rasterize or raytrace very large point sets, mostly for direct viewing [9, 22, 31]. The prefiltering method employed is almost always averaging of point attributes, without regard to accumulated directional opacity. Neyret [16] does more precise prefiltering on an octree: surface color and normals are modeled with ellipsoids, and filtered in a relatively principled way. Opacity is averaged, not composited, and only shadows from point lights are demonstrated.

Stochastic simplification [4] specifically targets aggregate geometry, and prefilters by removing primitives randomly and changing the sizes and colors of remaining primitives to maintain statistical properties. This method could be used for direct rendering while using our method for shadows.

2.3 Shadow Maps

Shadow maps are a common method for rendering shadows in games and film production. Traditional shadow maps precompute visibility in a scene for a frustum of rays originating at the light origin; no other visibility queries are supported. Deep Shadow Maps [14] and other multi-layer data structures [30] store a continuous opacity function along each ray, which is needed for aggregate geometry.

Shadow maps support constant-time queries, and are built by rendering the scene in a separate pass, which is fast for scenes of moderate complexity. However, shadow maps are less general than raytraced visibility. Occlusion from small area lights can be approximated with a single shadow map, but occlusion from large and/or non-planar lights cannot. Although it is possible to represent area lights using many shadow maps [21], computation time grows linearly with number of lights.

The cost of building a shadow map is high for very complex scenes. The resolution of shadow maps in production may be as much as $16K \times 16K$, to ensure there are no aliasing artifacts; at our studio we have occasionally even seen shadow map generation take longer than final rendering. Separate maps are needed for every light in a scene, and these maps must be recomputed when either geometry or lights change. In contrast, our prefiltering method can reuse the same precomputed data for any configuration of lights, and for all instances of a given object.

2.4 Other work

Probably the most common use of prefiltering is for mipmapping 2D surface color [28]. However, thinking only of this example can be misleading: averaging is (approximately) correct for diffuse surface color, but not for normals or 3D opacity.

Mesh simplification, e.g., using a quadric error metric [6], can be viewed as prefiltering, but does not explicitly preserve opacity, and does not work well for aggregate geometry.

3 Prefiltering Occlusion

Recall that our goal is to store prefiltered opacity in a BVH and use this information to speed up visibility queries during rendering. In this section we motivate this goal using Monte Carlo rendering theory, then give the details of how we prefilter a BVH, and how we intersect rays with a BVH using solid angle comparisons.

3.1 Theory

The irradiance at a surface point p with normal N is given by the following integral:

$$H(p,N) = \int_S V(p,\omega) L_f(p,\omega)(\omega \cdot N) d\omega$$

where $V(p,\omega)$ is a visibility function that returns a value in $[0,1]$, $L_f(p,\omega)$ is the incident radiance as seen by p in direction ω, and S is a region of the unit hemisphere with solid angle $\Omega(S)$. Ambient occlusion makes the approximation that $L_f = 1$ and also adds a distance term to visibility. Direct illumination for soft shadows can also be rewritten as a sum over many lights instead of as an integral over the hemisphere. Traditional Monte Carlo rendering samples the area source uniformly in area with M samples and estimates the integral with the following sum:

$$H' = \frac{A}{M} \sum_{i=1}^{M} V(p,p'_i) L_e(p'_i, p - p'_i) \frac{\cos\theta \cos\theta'}{\|p - p'_i\|^2}$$

where A is the area of the light source, p'_i is a point chosen on the light source, having normal N', and the angles θ and θ' are between the light direction vector $p - p'_i$ and the normals N and N' respectively. L_e is the surface radiance of the light, and is often constant. Visibility is evaluated by casting a shadow ray between points p and p'_i.

If instead the luminaire can be sampled directly according to the projected solid angle for the point p (see Shirley et al. [23] for examples where this is possible) the sum above can be rewritten as:

$$H' = \frac{1}{M} \sum_{i=1}^{M} V(p,\omega_i) L_f(p,\omega_i)(\omega_i \cdot N)$$

where $V(p,\omega_i)$ and $L_f(p,\omega_i)$ are visibility and radiance samples computed by shooting a single shadow ray, with origin p and direction ω_i. A single point sample is an estimate of irradiance over a small region $S_r(\omega_i)$ of the hemisphere having solid angle Ω/M.

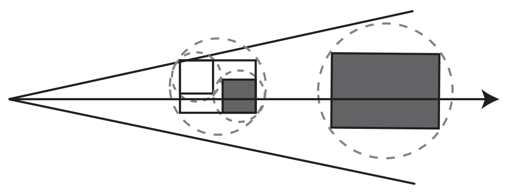

Figure 2: *A shadow ray has a virtual cone defined by solid angle. We terminate the ray at (shaded) BVH nodes that fit within its cone, and return the composited prefiltered opacities of the two nodes. Nodes which do not intersect the ray are ignored, unlike with cone tracing.*

In either case, the rate at which the sum H' converges to the integral H is largely dependent upon the variance of $V(p,\omega)$, since L_e is usually constant over the luminaire, and the cosine terms vary slowly. For scenes containing hair or vegetation, V may have very high variance, but a noisy estimate of H is not acceptable for production rendering.

We can eliminate much of the variance by prefiltering V and replacing it with a similar function, $\overline{V}$, that varies more slowly:

$$\overline{V}(p,\omega_i) = \frac{1}{\Omega(S_r)} \int_{S_r} V(p,\omega) d\omega$$

For efficiency, we compute $\overline{V}$ before rendering, and store its approximate value for many possible regions of integration. Precomputed values of $\overline{V}$ can then be fetched and interpolated quickly during rendering. This is the motivation for storing opacity in BVH nodes: the opacity α of a node is the approximate value of $1-\overline{V}$ for the geometry within the node.

3.2 Ray Intersection using Solid Angles

In our modified intersection scheme, each ray has an associated virtual cone defined by solid angle, $\Omega_r = \Omega(S_r)$ and ray length. A virtual cone is a simple isotropic form of ray differential [10]. When intersecting each ray with the BVH, we compare the solid angle of the ray to the projected solid angle of the box, Ω_b, and stop descending when $\Omega_r/\Omega_b > T$, where T is a user-defined threshold. Note that this is not the same as cone tracing; we are simply associating a solid angle with each ray but not performing exact intersection with a cone. For simplicity, instead of Ω_b we use the projected solid angle of the bounding sphere of the box, $\Omega_s = \pi R^2/|C-p|^2$, where C is the box center and R is the radius. An intersection is diagrammed in Figure 2.

Pseudo-code for a recursive ray-BVH intersection is shown in Algorithm 1; for clarity we assume that both child nodes exist and omit checking for null pointers, and also assume the existence of a *triangleOcclusion* function which returns 1 if a given ray and triangle intersect and 0 otherwise. When terminating, a ray-box intersection returns the approximate, pre-integrated opacity of the box, as viewed from the ray direction. We make the assumption that orthographic projection suffices, so that the opacity of the box depends only on viewing direction, ω, and not viewing position, p. This assumption has proven valid in practice, and becomes increasingly accurate as the number of rays increase and their solid angles decrease.

The final opacity of a shadow ray is computed by compositing the opacities of every interior node or leaf triangle at which the ray terminates. Nodes can be intersected in any order. This algorithm avoids all ray-box and ray-triangle intersections below the termination depth in the BVH. However, since by design prefiltered nodes are rarely opaque, it is likely that the ray intersects multiple interior nodes. We show in the results section that this trade off is beneficial for complex geometry and semi-opaque geometry.

Algorithm 1 occlusion(node, ray, t_0, t_1)

```
// Exit early if ray misses bounding box.
hitInterval = node.getBound().rayIntersect(ray)
if hitInterval.clip(t0, t1).isEmpty() then
    return 0.0
end if

// Descend immediately if node is below correlation height.
if node.height() <= K then
    if node.isLeaf() then
        return triangleOcclusion(node.getTriangle(), ray, t0, t1)
    else
        alpha0 = occlusion(node.left(), ray, t0, t1)
        alpha1 = occlusion(node.right(), ray, t0, t1)
        return alpha0 + alpha1 - alpha0*alpha1
    end if
end if

// Terminate ray based on solid angle
nodeSolidAngle = node.getSolidAngle(ray.origin)
if T * nodeSolidAngle < ray.solidAngle then
    return node.getStoredOpacity(ray.direction)
end if

// Recurse to child boxes.
alpha0 = occlusion(node.left(), ray, t0, t1)
alpha1 = occlusion(node.right(), ray, t0, t1)
leftSolidAngle = node.left().getSolidAngle(ray.origin)
rightSolidAngle = node.right().getSolidAngle(ray.origin)
childSolidAngle = min(leftSolidAngle, rightSolidAngle)

// Lerp between recursive opacity and prefiltered node opacity.
if T * childSolidAngle < ray.solidAngle then
    alpha = alpha0 + alpha1 - alpha0*alpha1
    nodeAlpha = node.getStoredOpacity(ray.direction)
    t = ray.solidAngle - T*childSolidAngle
    t = t / (T*node.solidAngle - T*childSolidAngle)
    t = clamp(0.0, 1.0, t)
    return t * nodeAlpha + (1-t) * alpha
end if
return alpha0 + alpha1 - alpha0*alpha1
```

3.3 Computing Directional Opacity

We now describe how to compute orthographic directional opacity, $\alpha(\omega) = 1-\overline{V}(\omega)$, for each node of a BVH. We define nodes using the sweep build from [25].

One algorithm for computing opacity is, after defining a node during the build, to render all triangles contained in the node and compute their average visibility. This algorithm has $O(N_T \log N_T)$ time complexity for the entire BVH, where N_T is the number of triangles.

We can improve on this algorithm if we assume that each node contains small, disconnected pieces of geometry that are not related to geometry in any other node. Porter and Duff make a similar assumption for pixels in their seminal work on image compositing [19]. We assume that nodes for aggregate geometry approximately satisfy this assumption above some height K in the BVH, where leaf nodes have height $K = 0$ and the height of an interior node is computed by averaging the height of its two children. We call K the *correlation height*. Note that strongly connected geometry, e.g., an opaque subdivided plane, would *not* follow this assumption, and in that case the opacity of a single BVH node should not be computed solely from the geometry within it or light leaks would occur.

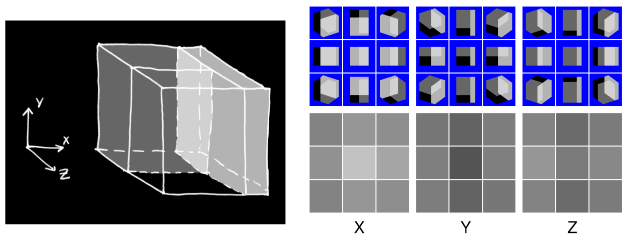

Figure 3: *One step in the bottom-up prefiltering of directional opacity. Left: An example BVH node, with opacities already assigned to the two child nodes. Right: Cubemap of opacities, rendered by projecting child boxes from multiple directions (top) and then averaging each projection (bottom). Pixels shown in blue lie outside the projection and are not averaged. Only three faces of the cubemap are stored, since opacity is independent of depth order.*

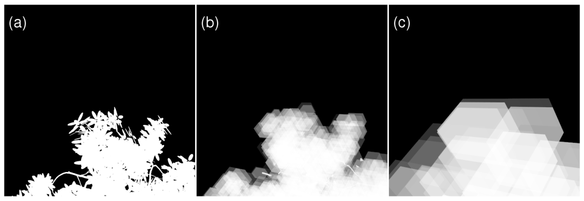

Figure 4: *"Worm's eye" view of the scene from a shading point, using (a) Unfiltered geometry (triangles), (b) Prefiltered BVH opacity with $T = 2$, (c) Prefiltered opacity with $T = 0.1$.*

Given a correlation height K as input, we can assign opacity to a node above K by rendering and compositing its two children using orthographic projection and alpha blending from a small number of directions. The projected image for each direction is averaged to compute an opacity. This process is diagrammed in Figure 3. We store directional opacity using a small cubemap with N_C^2 directions per face; the results in this paper use $N_C = 3$, and our projected images are each 10×10 pixels. Only 3 faces of the cube need to be rendered and stored since opacity is order-independent. For nodes near leaves, below level K, we do not prefilter. Rays that intersect these nodes during rendering will always descend the BVH, as shown in lines 7-16 of Algorithm 1. To prefilter nodes at level K, we compute an exact local opacity by rendering the triangles contained in the nodes. The combined opacity computation for the BVH using this hybrid method is $O(N_T)$, with storage requirements $O(N_T)$. It may be possible to determine K automatically, perhaps using a measurement of connectivity, but for our results we set K manually.

A final note: we linearly interpolate prefiltered opacity queries during rendering, by taking opacity values from both the level where the ray terminates with a hit or miss, and the next higher level, and interpolating between the two based on solid angle. This is shown in lines 31-39 of Algorithm 1. Interpolating helps eliminate any artifacts due to differences in the termination level of rays from nearby shading points, or during animation of the lights.

4 Results

4.1 Variance Analysis

Variance of scene properties causes visible noise in rendered images. To show that prefiltering the BVH reduces variance in the visibility function, we rendered "worm's eye" images of the bush scene with the camera positioned at a shading point on the ground plane, looking toward the area light. Worm's eye images are shown in Figure 4; black points indicate where rays strike the area light, and white points indicate occlusion.

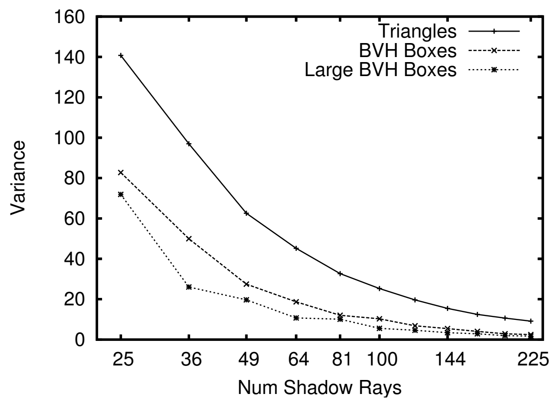

Figure 5: *Variance of estimating the image mean for a given number of jittered samples (shadow rays), for the images shown in Figure 4. Variance was computed using* 50,000 *trials.*

Table 1: *BVH Build Times*

Scene	Triangles	K	Prefilter (s)	Total Build (s)
Bush	2,020,600	5	24	36
Tentacles	3,823,893	10	11	35
Bamboo	5,181,376	5	108	144

The images shown here used $5^2 \times 512^2$ rays. However, for performance reasons we would like to estimate visibility during rendering by averaging many fewer shadow rays, perhaps less than 100. We computed the variance of such an estimate by sampling each worm's eye image repeatedly with M jittered samples. Plots of variance as a function of number of samples (shadow rays) are shown in Figure 5. The prefiltered images (b) and (c) of Figure 4, have lower variance, even though the top half of each image is constant. We conclude that if this image is representative, then for example using prefiltered geometry with 49 shadow rays would produce similar shadow noise as using the original geometry with 100 shadow rays.

4.2 Convergence

Figure 6 shows how our method approaches a reference solution as the number of shadow rays increases. Even with prefiltering, some minimum number of rays are required to adequately sample the silhouettes of BVH nodes; one ray is not sufficient. Note that we can also decrease noise for a given number of shadow rays by decreasing T; this will cause rays to stop at larger BVH nodes, producing a result that is smoother but less accurate. Generally we set T between 0.2 and 1.5, depending on the number of pixel samples.

4.3 Performance

First we show that prefiltering ray intersection reduces dependence on geometric complexity. Starting with the initial bush model, which contains about $500K$ triangles, we used linear subdivision to increase the triangle count by factors of $4\times$, to produce models with approximately $2M$ and $8M$ triangles ($32M$ would be out of core on our test machine). We rendered each model using identical camera and light parameters, using 4 parallel threads over image tiles, on a machine with 4 Intel Xeon 2.66 GHz cores and 8 GB of memory. Shadow ray intersection times were computed using software timers in each thread context, and we report the sum of thread time divided by number of cores. Shadow computation times for the prefiltered and unfiltered scenes are plotted in Figure 9, as a function

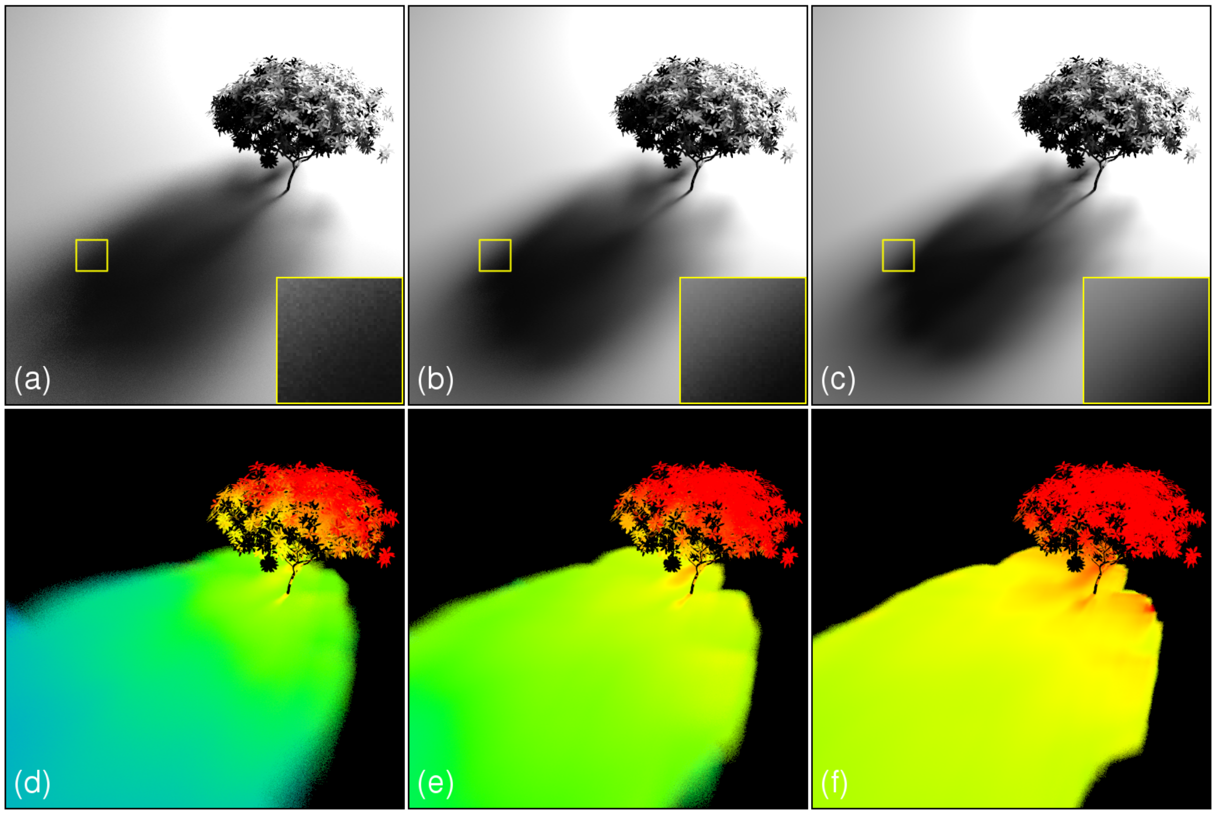

Figure 6: *Convergence of shadows from a prefiltered BVH with* 5×5 *samples per pixel,* $T = 0.2$*, and (a)* 4 *shadow rays, (b)* 16 *shadow rays, and (c)* 64 *shadow rays. Lower row shows a heatmap, with shading based on the BVH level at which rays terminate. Rays that terminate at larger nodes closer to the root are more blue.*

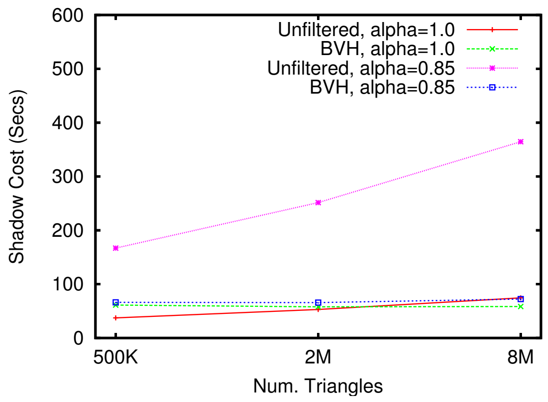

Figure 9: *Intersection times for the original bush model with* $500K$ *triangles, and linearly subdivided models with* $2M$ *and* $8M$ *triangles. We show times for both opaque and semi-opaque triangles.*

of subdivision level. We repeated the test using semi-opaque triangles, which frequently occur when rendering hair and foliage in production.

There are at least two notable features of Figure 9. First, the graph for the prefiltered geometry is nearly constant: shadow rays that travel only a short distance descend to triangles and incur a logarithmic intersection cost, but rays that stop early at a given level of the BVH continue to stop at the same level even after subdivision. Figure 9 also shows that for opaque geometry with $500K$ and $2M$ triangles, ray intersection is slower for the prefiltered geometry, ignoring differences in variance. This is because a shadow ray with prefiltered geometry may intersect and composite multiple non-opaque nodes, while a shadow ray for unfiltered geometry terminates as soon as it hits a single opaque triangle. As complexity increases, the time saved by not descending into prefiltered nodes increases while the time spent compositing nodes remains constant. Prefiltered shadows are slightly faster for $8M$ opaque triangles, and much faster for semi-opaque geometry at all subdivision levels.

Table 2: *BVH Memory Footprints*

Scene	Original (GB)	Prefiltered (GB)
Bush	0.82	1.22
Tentacles	1.50	2.28
Bamboo	2.21	3.79

Next we show that soft shadows from prefiltered geometry have less visible noise artifacts than shadows from unfiltered geometry computed in similar time. We used three test scenes: the bush, a “tentacles” scene that could represent hair or vegetation, and a bamboo forest. We compare a prefiltered solution to an unfiltered solution with similar cost, and a second solution with similar quality. Results are shown in Figure 1, Figure 7, and Figure 8. For all three cases, the prefiltered geometry is more efficient, with respective speedups of 9.5×, 6.3× and 3.9× for similar visible noise levels; the speedup is lowest for the bamboo because this scene is fully opaque and less dense than the other scenes. Speedups would continue to increase with additional geometric density, as suggested by Figure 9.

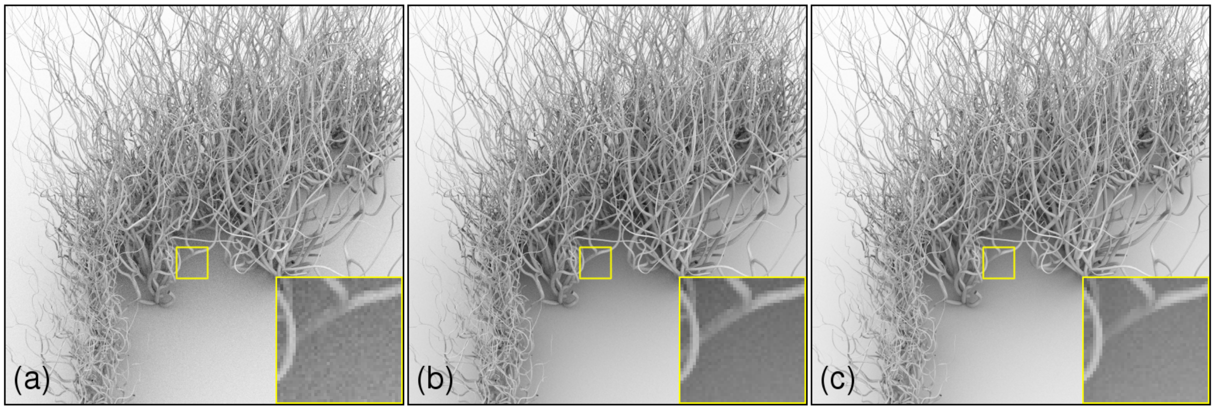

Figure 7: *Ambient occlusion on "tentacles" model with* $3,823,893$ *opaque triangles,* 5×5 *spp. (a) Unfiltered geometry,* 9 *shadow rays,* 106 *seconds. (b) Prefiltered geometry,* 9 *shadow rays,* 66 *seconds. (c) Unfiltered geometry,* 36 *shadow rays,* 420 *seconds.*

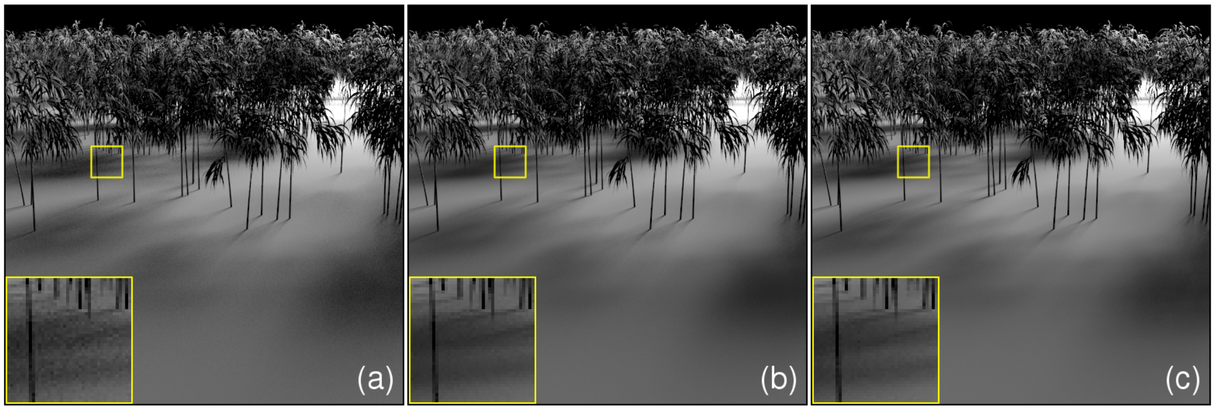

Figure 8: *Soft shadows on a bamboo forest with* $5,181,376$ *opaque triangles,* 5×5 *spp. (a) Unfiltered geometry,* 9 *shadow rays,* 64 *seconds. (b) Prefiltered geometry,* 9 *shadow rays,* 87 *seconds. (c) Unfiltered geometry,* 49 *shadow rays,* 339 *seconds.*

The prefiltered results are smoother and very slightly darker than the reference. Smoothing occurs when high frequency features in the shadow cannot be reconstructed for a given number of shadow rays; the alternative is visible noise, which we find much more objectionable. We suspect the darkening results from correlation between geometry in overlapping nodes which is not captured by our method, but this needs more investigation.

We report single-threaded BVH build and prefiltering times for all models in Table 1. We believe these times could be made faster, since our BVH build/prefiltering code is single threaded and not fully optimized. However, for static geometry, the times are still reasonable, especially since the prefiltered BVH can be reused for many images, including relighting scenarios. In the accompanying video, we show static geometry with an animated light, where the same BVH was used for all frames of the animation.

4.4 Compression

Prefiltered results shown thus far were generated using opacity cubemaps with 3×3 resolution per face. The cubemaps require 27 extra floats per BVH node, which increases the memory footprint of the BVH as shown in Table 2. To reduce the footprint, each cubemap could be represented more efficiently in the spherical harmonics basis. We have not yet implemented this type of compression, but as an early test we tried storing only the average value of the cubemap, which corresponds to the first, or "DC" spherical harmonics coefficient.

We also experimented with a simple, non-directional prefiltering method where we computed a scalar opacity per BVH node using an ad-hoc estimate of projected area:

$$\alpha = \alpha_0 A_0/A + \alpha_1 A_1/A - \alpha_0\alpha_1 A_0 A_1/A^2$$

Here A, A_0, A_1 are the cross sectional areas of the bounding spheres for the BVH node and the two child nodes, respectively. The goal here was to find a simple formula that might generate plausible results; we also tried using volume ratios of the child nodes to the parent node as weights in the compositing equation, but that almost always underestimated opacity.

In Figure 10 we compare a reference solution, directional opacity using a cubemap, "DC" average opacity, and the ad-hoc method described above. The ad-hoc method produces noticeably darker shadows and occlusion for the bush, tentacles, and bamboo scenes, although the darkening might be acceptable for some applications. We also constructed a scene with very directional behavior by distributing particles within a sphere, with every particle facing the same direction. In the row of Figure 10 labeled "Particles-transmit" all particles face 90 degrees away from the light, so that the maximum amount of light is transmitted; in the row labeled "Particles-block" all particles face the light, so that the maximum amount of light is blocked. The cubemap opacity captures much of the change in particle orientation, after increasing to 12×12 resolution for this case (the other cases use 3×3 resolution). The "DC" and ad-hoc results do not capture any change in particle orientation, and the ad-hoc method produces overly dark shadows for both orientations.

These tests lead us to believe that for many applications a small number of spherical harmonics coefficients would suffice for occlusion from aggregate geometry, although more coefficients would be needed for geometry that was strongly aligned.

5 Conclusion

We have described how to prefilter directional opacity for BVH nodes using $O(N_T)$ time and storage, and demonstrated that prefiltering both reduces the variance of occlusion queries, and makes ray intersection cost independent of geometric complexity for rays with large differentials.

As short term future work, we plan to further optimize the prefiltering and storage of opacity. Our current implementation renders opacity cubemaps by projecting child nodes using a software raytracer. It would probably be faster to project multiple nodes in parallel on the GPU. It might also be possible to use a lookup table for opacity based on canonical configurations of child nodes. Making prefiltering as fast as possible will be important for dynamic geometry, where the BVH would need to be updated or rebuilt every frame. As mentioned, we expect spherical harmonics compression of the opacity cubemaps to be very effective.

As longer term work, it should be possible to automatically estimate the correlation height K, perhaps using measurements of geometric connectivity. K could then vary locally over a scene. We are also interested in supporting other types of secondary ray queries involving color as well as opacity, such as indirect diffuse and subsurface scattering.

6 Acknowledgments

At Disney we wish to thank Joe Marks for supporting this research, and David Hermanson and Chuck Tappan for providing the test scenes. We also thank Dave Edwards for numerous discussions, and the anonymous reviewers for their helpful comments.

References

[1] J. Amanatides. Ray tracing with cones. *ACM SIGGRAPH 1984*, 18(3):129–135, 1984.

[2] L. Bavoil, S. P. Callahan, and C. T. Silva. Robust Soft Shadow Mapping with Backprojection and Depth Peeling. *Journal of Graphics Tools*, 2008.

[3] M. Bunnell. Dynamic ambient occlusion and indirect lighting. *GPU Gems*, 2:223–233, 2005.

[4] R. Cook, J. Halstead, M. Planck, and D. Ryu. Stochastic simplification of aggregate detail. *International Conference on Computer Graphics and Interactive Techniques*, 2007.

[5] R. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. *ACM SIGGRAPH 1984*, pages 137–145, 1984.

[6] M. Garland and P. Heckbert. Surface Simplification Using Quadric Error Metrics. *ACM SIGGRAPH 1997*, pages 209–216, 1997.

[7] J. Hasenfratz, M. Lapierre, N. Holzschuch, and F. Sillion. A survey of Real-Time Soft Shadows Algorithms. *Computer Graphics Forum*, 22(4):753–774, 2003.

[8] P. Heckbert and P. Hanrahan. Beam tracing polygonal objects. *ACM SIGGRAPH 1984*, 18(3):119–127, 1984.

[9] E. Hubo, T. Mertens, T. Haber, and P. Bekaert. The quantized kd-tree: efficient ray tracing of compressed point clouds. *IEEE Symposium on Interactive Ray Tracing, IEEE*, pages 105–113, 2006.

[10] H. Igehy. Tracing ray differentials. *ACM SIGGRAPH 1999*, pages 179–186, 1999.

[11] J. Kautz, J. Lehtinen, and T. Aila. Hemispherical rasterization for self-shadowing of dynamic objects. *Proceedings of the Eurographics Symposium on Rendering*, pages 179–184, 2004.

[12] J. Kontkanen and S. Laine. Ambient occlusion fields. *I3D 2005*, pages 41–48, 2005.

[13] S. Laine, T. Aila, U. Assarsson, J. Lehtinen, and T. Akenine-Moller. Soft shadow volumes for ray tracing. *ACM SIGGRAPH 2005*, pages 1156–1165, 2005.

[14] T. Lokovic and E. Veach. Deep shadow maps. *ACM SIGGRAPH 2000*, pages 385–392, 2000.

[15] D. Mitchell. Consequences of stratified sampling in graphics. *ACM SIGGRAPH 1996*, pages 277–280, 1996.

[16] F. Neyret. Modeling, Animating, and Rendering Complex Scenes Using Volumetric Textures. *IEEE Trans. Vis. Comput. Graph.*, 4(1):55–70, 1998.

[17] R. Overbeck, R. Ramamoorthi, and W. Mark. A Real-time Beam Tracer with Application to Exact Soft Shadows. *Proceedings of the Eurographics Symposium on Rendering*, 2007.

[18] Pixar. Prman application note: Point-based approximate ambient occlusion and color bleeding. 2006.

[19] T. Porter and T. Duff. Compositing digital images. *ACM SIGGRAPH 1984*, pages 253–259, 1984.

[20] Z. Ren, R. Wang, J. Snyder, K. Zhou, X. Liu, B. Sun, P. Sloan, H. Bao, Q. Peng, and B. Guo. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. *ACM SIGGRAPH 2006*, pages 977–986, 2006.

[21] T. Ritschel, T. Grosch, J. Kautz, and S. Müller. Interactive Illumination with Coherent Shadow Maps. *Proceedings of the Eurographics Symposium on Rendering*, 2007.

[22] S. Rusinkiewicz and M. Levoy. QSplat: a multiresolution point rendering system for large meshes. *ACM SIGGRAPH 2000*, pages 343–352, 2000.

[23] P. Shirley, C. Wang, and K. Zimmerman. Monte Carlo techniques for direct lighting calculations. *ACM Transactions on Graphics (TOG)*, 15(1):1–36, 1996.

[24] P. Sloan, J. Kautz, and J. Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM SIGGRAPH 2002*, pages 527–536, 2002.

[25] I. Wald, S. Boulos, and P. Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics*, 26(1):6:1–6:18, Jan. 2007.

[26] I. Wald, P. Slusallek, C. Benthin, and M. us Wagner. Interactive Rendering with Coherent Ray Tracing. In *Proceedings of EUROGRAPHICS*, pages 153–164, 2001.

[27] M. Wand and W. Straßer. Multi-Resolution Point-Sample Raytracing. *Graphics Interface Proceedings 2003*, 2003.

[28] L. Williams. Pyramidal parametrics. In *ACM SIGGRAPH 1983*, pages 1–11, 1983.

[29] A. Woo, P. Poulin, and A. Fournier. A Survey of Shadow Algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, nov 1990.

[30] F. Xie, E. Tabellion, and A. Pearce. Soft Shadows by Ray Tracing Multilayer Transparent Shadow Maps. *Proceedings of the Eurographics Symposium on Rendering*, 2007.

[31] S. Yoon, C. Lauterbach, and D. Manocha. R-LODs: fast LOD-based ray tracing of massive models. *The Visual Computer*, 22(9):772–784, 2006.

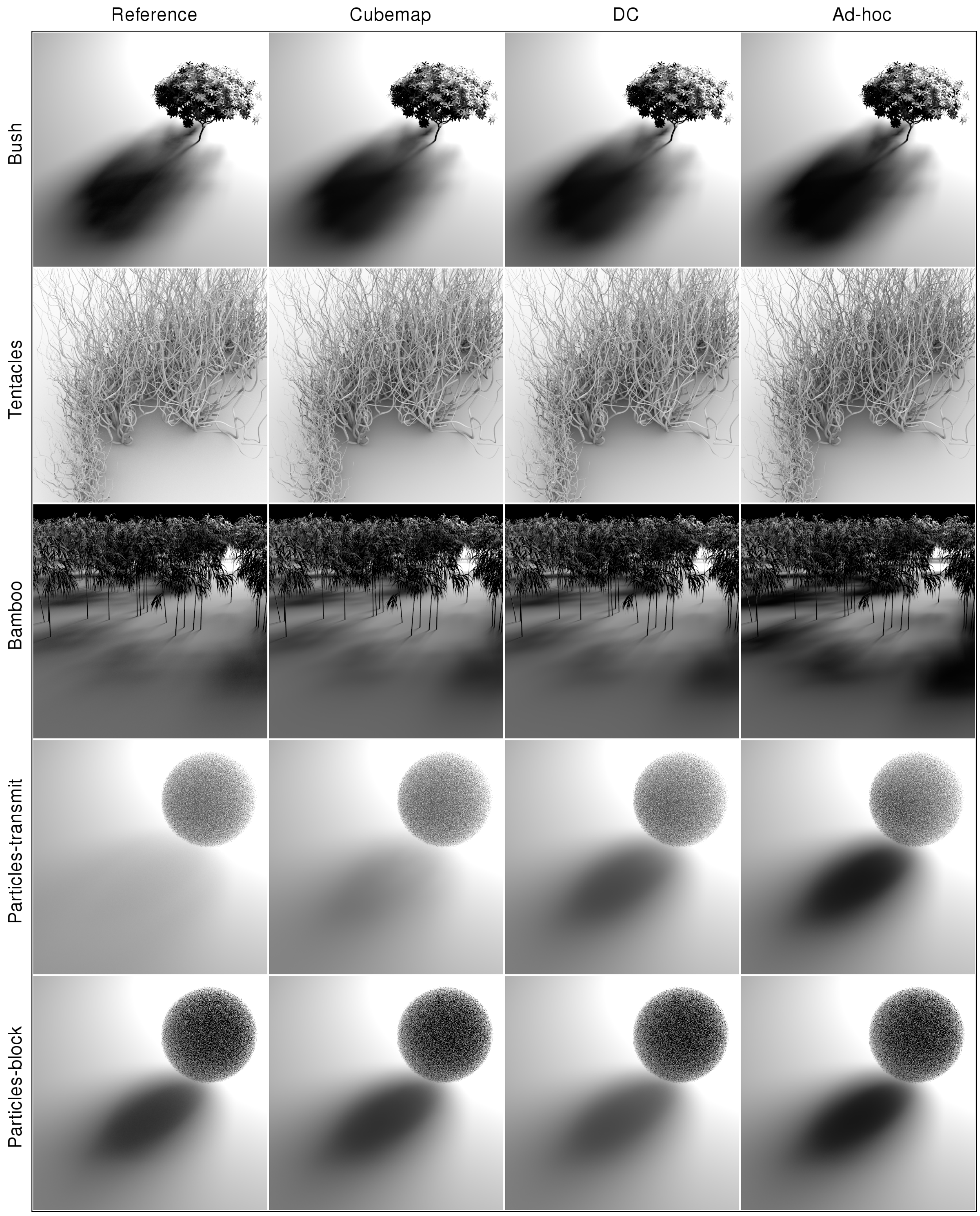

Figure 10: *Quality comparison of prefiltering methods to a reference solution shown in the leftmost column. The DC and Ad-hoc methods are less accurate than the Cubemap method but also require less memory. The Ad-hoc method produces overly dark shadows.*