DOI: 10.1111/j.1467-8659.2010.01728.x

Eurographics Symposium on Rendering 2010
Jason Lawrence and Marc Stamminger
(Guest Editors)

Volume 29 (2010), Number 4

Two Methods for Fast Ray-Cast Ambient Occlusion

Samuli Laine Tero Karras

NVIDIA Research

Abstract

Ambient occlusion has proven to be a useful tool for producing realistic images, both in offline rendering and interactive applications. In production rendering, ambient occlusion is typically computed by casting a large number of short shadow rays from each visible point, yielding unparalleled quality but long rendering times. Interactive applications typically use screen-space approximations which are fast but suffer from systematic errors due to missing information behind the nearest depth layer.
In this paper, we present two efficient methods for calculating ambient occlusion so that the results match those produced by a ray tracer. The first method is targeted for rasterization-based engines, and it leverages the GPU graphics pipeline for finding occlusion relations between scene triangles and the visible points. The second method is a drop-in replacement for ambient occlusion computation in offline renderers, allowing the querying of ambient occlusion for any point in the scene. Both methods are based on the principle of simultaneously computing the result of all shadow rays for a single receiver point.

1. Introduction

Ambient occlusion has become one of the standard tools used in high-quality production rendering. It was introduced by Zhukov et al. [ZIK98] and soon picked up by the rendering community [Lan02]. Today it is easy to find numerous tutorials and examples on how to improve the realism of rendered images by rendering an ambient occlusion pass and using it in the final composition phase.

The amount of ambient occlusion for a point is most often defined as the cosine-weighted fraction of the hemisphere where incoming ambient light cannot reach the surface. This simple definition is, however, unsatisfactory. For example, the rendering situation determines how far we want to look for occlusion. For indoor scenes, light from the sky never reaches the surfaces, so a shorter distance is appropriate, whereas outdoor scenes may use a longer distance. Also, the simple definition does not tell how the occlusion tapers off with distance.

A practical and flexible definition is obtained by utilizing a falloff function that defines how the distance to occluding surfaces affects the amount of incoming light. Following Zhukov et al., we define the amount of ambient occlusion as

$$W(\mathbf{p},\mathbf{n}) = \frac{1}{\pi}\int_{\Omega} \rho(D(\mathbf{p},\omega))(\omega\cdot\mathbf{n})\,d\omega. \qquad (1)$$

Here, W is the amount of occlusion, and $\mathbf{p}$ and $\mathbf{n}$ are the surface point receiving the occlusion and its normal, respectively. The integral is taken over the hemisphere oriented towards $\mathbf{n}$. $D(\mathbf{p},\omega)$ measures the distance to nearest occluder from $\mathbf{p}$ towards direction ω, and ρ is the falloff function that converts this distance to an occlusion factor between 0 and 1. The choice of ρ depends on rendering needs, so a generic ambient occlusion algorithm should be able to support any such function. We may, however, assume that there is some distance r after which ρ is zero. This allows us to consider a bounded region in space instead of the entire scene when computing ambient occlusion for a single point.

Usually the integral is approximated by sampling the domain in a number of points. By distributing the points according to cosine weighting, we may dispose of the normalization factor and the dot product in the end. This yields

$$W(\mathbf{p},\mathbf{n}) \approx \frac{1}{N}\sum_{i=1}^{N} \rho(D(\mathbf{p},\omega_i)). \qquad (2)$$

The traditional method to evaluate this formula is to cast N

rays from point $\mathbf{p}$, mapping the first-hit distances through ρ and taking the average. The expensive part is the casting of the rays, but limiting their length to r makes them faster than generic rays. In production rendering, N may be set as high as 1024 for obtaining noise-free results.

For our purposes, we need that the rays that are cast are shadow rays, i.e., ones that only report if the ray was occluded or not, and therefore we cannot evaluate the distance function D. However, we can reformulate the above equation in terms of shadow rays by incorporating the falloff function into the sampling points by stochastically distributing the shadow ray lengths l_i according to ρ. This gives our final formula

$$W(\mathbf{p},\mathbf{n}) \approx \frac{1}{N}\sum_{i=1}^{N} O(\mathbf{p},\omega_i,l_i), \tag{3}$$

where $O(\mathbf{p},\omega_i,l_i)$ is a binary occlusion function that returns one if the ray between points $\mathbf{p}$ and $\mathbf{p}+l_i\omega_i$ is blocked, and zero otherwise. We define the ambient occlusion radius r as the length of the longest ray.

For interactive applications, high-quality sampling would be prohibitively slow, and a screen-space approximation is commonly used instead. The principle is to use the depth buffer information to infer enough of the 3D structure of the scene to be able to render convincing ambient occlusion. The problem with this approach is that there are cases which appear similar in the depth buffer but should produce different results, leading to systematic errors. We will briefly touch screen-space ambient occlusion methods at the end of Section 2, but they are otherwise outside the scope of this paper.

1.1. Contributions

In this paper, we present two methods for efficient computation of ambient occlusion according to Equation 3. Both methods are based on storing the status of individual shadow rays in bit masks and updating the masks using precomputed bit patterns stored in look-up tables. This process is explained in Section 3.

The first of our methods is targeted for rasterization-based engines. It exploits fixed-function hardware rasterization and depth test to find pixels where a given triangle may contribute to ambient occlusion. This is similar to a previous method by McGuire [McG10], but instead of flattening the ambient occlusion into a scalar value, we maintain accurate per-ray occlusion information. To improve performance, we support one-sided triangles, i.e., ones that occlude shadow rays from one direction only. This optimization is possible with most real-time content, as it has usually been designed to support backface culling. The rasterization-based method is detailed in Section 4.

The second method is designed to be a drop-in replacement for ambient occlusion computation in ray tracers. It is based on traversing a bounding volume hierarchy of the scene triangles, as such a hierarchy can be assumed to be present in a ray tracing system. Unlike the rasterization-based method, this method can provide ambient occlusion information for any point in the scene. This makes it suitable for handling the shading of secondary rays as well. The traversal-based method is described in Section 5.

2. Previous work

In this section, we will focus on methods that are directly related to our work or are equivalent to ray tracing-based ambient occlusion calculation. For a recent and comprehensive survey of exact and approximative ambient occlusion techniques, we refer the reader to Méndez-Feliu and Sbert [MFS09]. In addition, quality comparisons between a few approximative methods can be found in [McG10].

Both of our methods are based on storing the status of individual shadow rays in bit masks. The same approach has been previously used for area light sources and environment light sources (e.g. [KLA04, LA05, SS07]).

The hemispherical rasterization technique of Kautz et al. [KLA04] uses precomputed bit mask look-up tables for rapidly identifying which shadow rays are occluded by a triangle. However, they compute shadowing from distant environment maps and hence only need to support infinitely long rays. Handling limited-length rays for ambient occlusion requires a more versatile look-up table.

Our rasterization-based method rasterizes volumes that bound the influence of individual triangles. Compared to ambient occlusion volumes presented by McGuire [McG10], our bounding volumes are more compact. We also avoid the construction and rendering of a bounding volume when ambient occlusion radius is large compared to triangle size, and instead render a polygonal approximation of the influence hemisphere, which produces tighter bounds.

Our hierarchy traversal-based method bears resemblance to the BVH packet tracing algorithm of Wald et al. [WBS07]. They traverse the hierarchy once for a packet of rays and keep track of closest hit for each ray. We optimize this by using bit operations to handle multiple rays at once, and by ordering the traversal according to estimated occlusion power.

As we are focusing on the evaluation of Equation 3, techniques that require further approximations are outside the scope of this paper. For a recent screen-space ambient occlusion method, see e.g. the horizon-based method of Bavoil and Sainz [BS08] that has been extended to utilize multiple depth layers [BS09] to battle artifacts caused by the single-depth approximation. The results are still far from ray casting, albeit obtained orders of magnitude faster. Besides screen-space ambient occlusion, it is possible to propagate occlusion in a hierarchy of surface elements [Bun05, HJ07]. This approach is more computationally intensive and works only for models for which a satisfactory surface element approximation can be constructed.

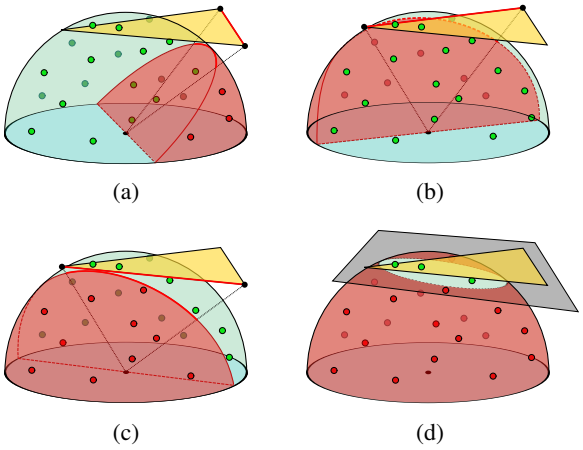


Figure 1: *Occluding the shadow rays. Colored dots on the hemisphere indicate sampling points that correspond to shadow rays between the receiver point and the sampling point. Note that some samples are on the far side of the hemisphere. Our method allows the sampling points to reside anywhere in space, but in this example all points are on the surface of a hemisphere for clarity. (a)–(c) Each edge of the triangle forms a plane with the receiver point. Sampling points on the negative side of the plane (red) are rejected. (d) Sampling points on the negative side of the triangle plane are also rejected. The samples that are accepted by all four planes are blocked by the triangle.*

3. Occlusion computation

Both of our methods compute ambient occlusion in a number of receiver points by determining which shadow rays from each point are blocked. To be able to use precalculated occlusion masks, we use the same pattern of shadow rays for every receiver point. This pattern is centered at the receiver point and oriented towards its normal.

However, producing a coordinate basis based on the normal of the receiver point alone is not enough, because the orientation would be constant on flat surfaces. This kind of coherent sampling pattern would cause banding, so we jitter the orientation in two ways. Firstly, we rotate the basis around the receiver point normal by a random amount. To retain some amount of coherence which speeds up processing, we keep the rotations relatively small. Rotation jitter alone is not enough to remove all banding, so we also randomly perturb the normal slightly. We make sure that the normal perturbation is small enough not to cause any rays to cross below the original tangent plane.

3.1. Occlusion masks

Our methods are based on storing the status of individual shadow rays in bit masks, and updating their status using bitwise operations and look-up tables. Figure 1 illustrates the process of finding the shadow rays blocked by a triangle after it has been transformed to the coordinate system of the receiver point. First, the three edges of a triangle are used for finding shadow rays that point towards the triangle. Then, the plane of the triangle is used for ignoring rays that are too short to reach the triangle.

The handling of triangle edges is similar to the hemispherical rasterization method of Kautz et al. [KLA04]. They also construct planes that pass through the origin for the edges of a triangle, and look up precalculated bit masks based on the plane normal. However, we need to handle generic planes, and not only those that pass through the origin, to take the plane of the triangle into account. Therefore, the two-dimensional cube map look-up used by Kautz et al. is not sufficient for our purposes.

We express a generic plane facing towards origin (i.e., receiver point) using its normal vector $\mathbf{n_p}$ and distance d from the origin. Given the maximum ray length r, we only need to consider planes where $0 \leq d \leq r$, because triangles whose plane is farther away cannot occlude any of the rays.

The bit masks for our look-up table are stored into a cube-shaped 3D table. To address the table, we take the plane normal $\mathbf{n_p}$ and multiply it by suitably scaled distance d_{norm} to produce a 3D coordinate with components in range $[-1, 1]$. The trivial choice would be to take $d_{norm} = d/r$, but this has two drawbacks. First, the resolution is bad when d is close to zero, because there are few available locations near the center of the table. In particular, the planes with $d = 0$ that are needed for triangle edges end up in the same table entry. Second, the regions near the corners remain unused, because $\mathbf{n_p}$ is on a sphere.

To avoid the loss of resolution near the center of the table, we take $d_{norm} = 1 - \frac{7}{8}(d/r)$. This way, the center of the table is left unused, but this only wastes a tiny fraction of space and greatly alleviates the resolution problem near the center. The angular resolution is best for planes with d close to zero, because they are stored near the outermost layer of the table. We have found the factor of $\frac{7}{8}$ to offer a good trade-off between loss of angular resolution for planes with large d and loss of distance resolution in general. Finally, we stretch the utilized region to reach the corners of the cube by further dividing d_{norm} by the largest absolute component of $\mathbf{n_p}$.

With this arrangement, the outermost cells of the 3D table correspond to planes with $d = 0$, which are used for triangle edge tests (Figure 1a–c). The outermost layer of our table therefore corresponds to the cube map of Kautz et al. The interior cells are used for handling the plane of the triangle (Figure 1d).

We produce the look-up table based on a given sampling pattern as a pre-process on the GPU before starting the ambient occlusion computation. With 128 sampling points, a typical 128^3 table consumes 32MB of GPU memory and takes a few milliseconds to generate.

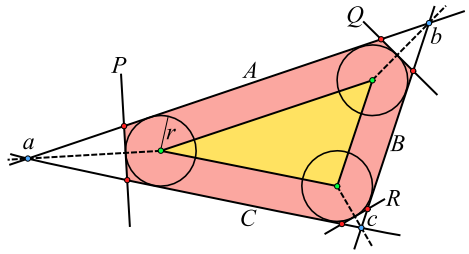


Figure 2: *Construction of the base of the hexagonal prism bounds for a triangle. The original triangle is at the center. First, we construct lines A, B, C that are parallel to the edges of the triangle at distance of r. Then, we intersect these lines with each other to produce points a, b, c. It would be possible to construct a triangular prism using these points as the base [McG10], but sharp corners would produce volumes that cover many superfluous pixels when rasterized. Instead, we take the lines between triangle vertices and points a, b, c and construct perpendicular lines P, Q, R at distance r from each vertex. These are intersected with lines A, B, C to produce the final base hexagon.*

4. Method 1: Rasterization

Our first method is based on rasterizing the region of influence of each triangle in the scene, and accumulating the results into a set of occlusion buffers. The occlusion buffers store the status of each shadow ray in one bit that is initially zero, corresponding to the ray being unblocked. With four-component 32-bit integer buffers, we can store up to 128 bits per pixel in a single buffer. For larger sample counts, we use multiple buffers.

The outline of our algorithm is as follows:

1. Render the scene, producing depth and normal for each pixel (i.e. receiver point).
2. Clear the occlusion buffers to zero.
3. For each triangle in the scene:
   a. Construct a shape that covers the triangle's region of influence.
   b. Rasterize far side of the shape with backface culling.
   c. Use an inverted depth test to cull fragments that lie in front of the receiving geometry.
   d. For each rasterized fragment:
      – Compute occlusion between the triangle and the receiver point.
      – Accumulate occlusion bits into the occlusion buffers.
      – Do not perform depth writes.
4. For each receiver point, evaluate Equation 3 using the occlusion bits and perform deferred shading.

We implemented the algorithm in OpenGL, performing step 3.a in the geometry shader and step 3.d in the fragment shader. We maintain the occlusion buffers in `GL_RGBA32UI` textures, and update them simultaneously in step 3 using `glDrawBuffers` and `glLogicOp`. For 128 samples per pixel in 1024×768 resolution, the total memory footprint of the occlusion buffers is 12MB.

4.1. Bounding the region of influence

For each triangle in the scene, we construct a shape in the geometry shader that covers all the receiver points whose shadow rays are potentially occluded by the triangle. Depending on the case, we use either a hexagonal prism that acts as a bounding volume for the triangle's region of influence, or a screen-space hemispherical billboard. The choice between the two is made based on the ratio between the radius of the triangle's bounding sphere R and ambient occlusion radius r. We use a threshold parameter α that determines the point where the switch happens. When triangle radius is large, i.e., $R/r > \alpha$, the region of influence is governed mainly by the shape of the triangle, and a hexagonal prism yields a tight bounding volume. When $R/r < \alpha$, the region of influence is closer to a hemispherical shape, so we construct a billboard for obtaining more efficient bounds. In our tests, $\alpha = 0.2$ produced consistently good results, and this value has been used in the benchmarks.

Figure 2 shows how the base of the hexagonal prism-shaped bounding volume is constructed for a triangle. The cover of the prism is formed by simply lifting the vertices of the base to distance of r towards the direction of triangle normal. For one-sided triangles, we offset the base of the prism slightly inside the surface to avoid precision issues during rasterization. This ensures that surface points on concave edges will be inside the bounding volumes of adjacent triangles. For two-sided triangles, we offset the base to distance of r away from the triangle normal, producing twice as large volume.

The resulting bounding volume is composed of 20 triangles organized into 2 triangle strips. We use backface culling to rasterize only the triangles that are on the far side of the prism, and perform the depth test in an inverted fashion. This avoids difficulties with clipping when the camera is inside the prism.

When the triangle is small compared to r, the region of influence for an one-sided triangle is approximately hemispherical, making the hexagonal prism a bad fit. In these cases, we construct a billboard, i.e., a flat polygonal shape that covers the region of influence in screen space. For two-sided triangles, a simple circular polygon is sufficient. For one-sided triangles, the shape consists of two polygons as illustrated in Figure 3. Each polygon has 12 vertices, and is organized into a strip of 10 triangles.

To handle perspective projection, we generate the polygons in object space and then correct their depth so that the billboard appears to be located just behind the sphere of influence. This way, projection and clipping are handled automatically by the rasterization hardware, while all points inside the region of influence are still guaranteed to pass the inverted depth test.

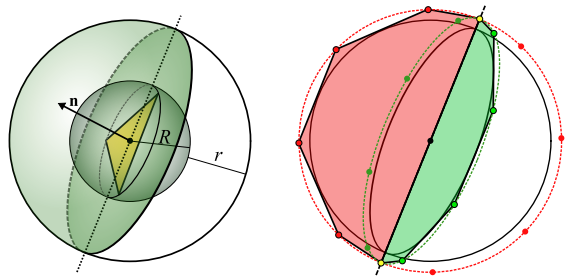


Figure 3: *Construction of hemisphere billboard for a one-sided triangle. Left: Triangle, as seen from the camera, surrounded by its bounding sphere with radius R. Adding ambient occlusion radius r gives a sphere that bounds the region of influence. The shaded region that needs to be rasterized is defined by the circle which is the silhouette of the sphere as seen by the camera, and the circle which is the intersection between triangle plane and the sphere (projected to an ellipse). Right: We approximate these circles with regular polygons placed on suitably enlarged circles (dashed). One polygon is drawn for each circle. Vertices on the wrong side of the separating line are replaced by the intersection points between circles (yellow dots), yielding no overlap between the rasterized polygons.*

4.2. Fragment processing

After the prism or billboard is constructed, it is rasterized to produce a set of fragments corresponding to points on its surface. Fragments that pass the inverted depth test are processed in the fragment shader to accumulate occlusion caused by the triangle into the occlusion buffers. To determine the receiver point $\mathbf{p}$ where the ambient occlusion is to be computed, we consult the previously rendered depth buffer. We also fetch the normal for the point and construct a coordinate basis that defines the orientation of the sampling pattern. The triangle is transformed to this basis so that subsequent calculations can be performed with the receiver point at the origin and the z-axis pointing towards its normal.

To reduce the number of mask lookups, the fragment shader performs a number of additional culling tests to terminate processing if the receiver point $\mathbf{p}$ cannot be influenced by the triangle we are processing. The culling tests, in the order they are processed, terminate the shader if:

1. Depth of the fragment is greater than depth of receiver point plus ambient occlusion radius.
2. $\mathbf{p}$ is outside the bounding sphere of triangle's region of influence.
3. The triangle is one-sided and faces away from $\mathbf{p}$.
4. Distance between $\mathbf{p}$ and triangle plane is greater than r.
5. All vertices of the triangle are below the horizon of $\mathbf{p}$.

The effectiveness of each of the tests is analyzed in Section 6. For points that survive all of the culling tests, we construct the occlusion mask as described in Section 3. This mask is blended to the occlusion buffers using logical OR blending mode.

4.3. Level of detail

It is easy to notice that small geometric details do not cause discernible features in ambient occlusion except near the surface itself, and ambient occlusion to distant surfaces could be just as well calculated using simplified geometry. However, the occlusion between nearby surfaces always needs to be calculated using the original geometry, because otherwise artifacts can appear.

Based on this observation, we developed an approximation scheme where ambient occlusion is calculated using simplified geometry depending on the distance from the receiver point. We start with the original geometry and then construct a series of simplified approximations with an increasing amount of allowed approximation error ϵ_i. The original geometry corresponds to distance range $[0, k \cdot \epsilon_1]$, while each simplified level corresponds to range $[k \cdot \epsilon_i, k \cdot \epsilon_{i+1}]$. k is an arbitrary scaling factor, usually between 2 and 5, selected to prevent visible artifacts due to the approximation.

We process all depth ranges in a single rendering pass using an additional attribute array to store $[r_{min}, r_{max}]$ for each triangle. When constructing bounding volumes and evaluating occlusion masks, we simply use r_{max} in place of r. To account for r_{min}, we need to perform an additional mask lookup in the fragment shader, similar to the one in Figure 1d. This prevents coarse simplification levels from influencing the occlusion of nearby surfaces.

By using simplified geometry, we cannot anymore claim that the results are consistent with a ray tracer. However, the perceptual difference to using original geometry all the way is tolerable, as long as the distance ranges and allowed simplification error are chosen appropriately. With a considerable amount of hand-waving, one could argue that using simplified geometry corresponds to having the ambient occlusion rays wiggle around by at most the amount of simplification error. It seems that this kind of approximation error, occurring far away from the receiver point, is less likely to cause noticeable artifacts than some other approximations such as flattening the intermediate results to a scalar value, or attempting to cope with a single depth layer. Nonetheless, our main benchmarks are run without any geometry simplification, and its effectiveness is analyzed in a separate test.

5. Method 2: BVH traversal

Our second method is targeted to offline ray tracers where ambient occlusion would otherwise be computed using ray casts. We assume that a bounding volume hierarchy of scene geometry is available, and exploit it for efficiently finding triangles that may contribute to ambient occlusion for a given receiver point.

We process each receiver point $\mathbf{p}$ independently. During the processing of a receiver point, we maintain a cumulative occlusion mask that is initially zero, corresponding to all shadow rays being unblocked. After initialization, we execute a BVH traversal to find nodes that intersect the hemisphere defined by receiver point $\mathbf{p}$, its normal, and ambient occlusion radius r. The BVH traversal algorithm follows the speculative while-while structure [AL09] which we found to give best performance. The idea is to organize the traversal loop so that all SIMD lanes always transition between traversing nodes and processing triangles together. The traversal phase continues until all lanes have found at least one triangle to process. Lanes that have already found triangles will continue visiting nodes speculatively while waiting for the transition.

When traversing an internal node, we first check the bounding boxes of its children against the hemisphere. If the distance between a box and $\mathbf{p}$ is greater than r or the box is entirely below the horizon of $\mathbf{p}$, we know that it cannot intersect the hemisphere and cull the corresponding nodes.

Next, we use the lookup tables to calculate a conservative node occlusion mask for each remaining child. We are primarily interested in the amount of additional occlusion that we can observe by processing the corresponding subtree. To estimate this, we count the number of rays that are still unblocked (zero in the cumulative mask) but may potentially be blocked by the subtree (one in the node mask). We cull any nodes for which this estimate is zero, since processing them would have no effect on the result. In case both child nodes are still eligible for traversal, we choose to first visit the one with a higher estimate. This leads to a traversal order that tends to build up occlusion quickly in the beginning and then use the built-up occlusion to cull most of the nodes later on.

We have experimented with two techniques for calculating the occlusion mask for a node. The first is to find the silhouette edges of the node bounding box, as seen from $\mathbf{p}$, fetch bit masks for these edges from the look-up table, and combine them using AND operation to form the occlusion mask for the bounding box. Since there are usually six silhouette edges, the mask lookups become fairly expensive, but the resulting occlusion mask is as tight as possible. If $\mathbf{p}$ is inside the node bounding box, we can avoid the lookups and simply output a mask with all bits set.

The second technique produces an approximate occlusion mask by constructing a bounding sphere for the node and calculating the corresponding apex angle and direction from $\mathbf{p}$. Figure 4 illustrates two cases that need to be considered. If all shadow rays are of equal length, we can use the plane look-up table described in Section 3 directly. Otherwise, we need to have a separate lookup table with normalized rays for the projection case (Figure 4 right). Similarly to the first technique, we can avoid the lookups if $\mathbf{p}$ is inside the bounding sphere.

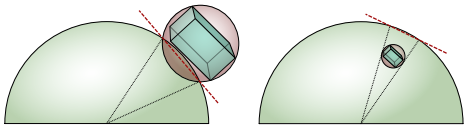


Figure 4: *Calculation of circular occlusion mask for a node. We construct a bounding sphere for the node and use one or two plane lookups to determine the rays blocked by the sphere. Left: if the sphere intersects the hemisphere surface, we can find rays pointing towards the intersection using a separating plane. Right: if the sphere is near the origin, we also consider a plane corresponding to its projection on the hemisphere surface.*

When a leaf node is reached, we first perform culling tests for each triangle before constructing occlusion masks for them and accumulating the occlusion as described in Section 3. In the order of execution, the triangle is ignored if:

1. All vertices of the triangle are below the horizon of $\mathbf{p}$.
2. The triangle is one-sided and faces away from $\mathbf{p}$.
3. Distance between $\mathbf{p}$ and triangle plane is greater than r.
4. $\mathbf{p}$ is outside the bounding sphere of triangle's region of influence.

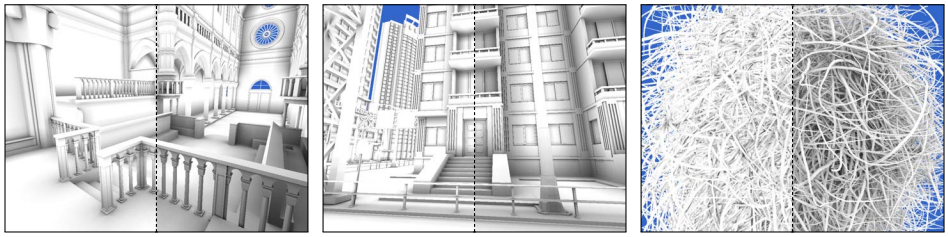
These are a subset of tests done in the rasterization method. The ordering is not the same because the cost-effectiveness of the tests is different in rasterization and traversal-based methods.

We implemented the BVH traversal method in CUDA, assigning one receiver point for each computation thread. The receiver points are sorted in Morton order to increase coherence between nearby threads. As in the rasterization-based method, the look-up tables are accessed through textures to take advantage of texture caches, whereas the bounding volume hierarchy is accessed using ordinary memory loads that utilize the L1 and L2 caches of the GPU. We found that circular node occlusion masks gave better overall performance than box-shaped ones, so we chose to use them in the benchmarks.

6. Results

We compared the performance of our methods against the state-of-the art GPU ray caster of Aila and Laine [AL09]. The comparison ray caster was optimized for shadow rays and for the GPU used. All tests were run on an NVIDIA GTX 480 GPU with 1.5 GB of RAM installed in a PC with 2.5 GHz Q9300 Intel Core2 Quad CPU and 4 GB of RAM. The operating system was 32-bit Windows XP Professional.

The shadow ray sampling pattern is generated by constructing a low-discrepancy distribution on unit disc and lifting it onto the surface of hemisphere to produce a cosine-weighted direction distribution. We generate the distribution on disc by first randomly placing N samples and performing a large number of Lloyd relaxation steps to reach a semi-



		Sibenik, 77K tris		City, 879K tris		Hairball, 2.88M tris	
Method	N	$r = 0.75$	$r = 2.25$	$r = 75$	$r = 225$	$r = 0.1$	$r = 0.3$
Rasterization	128	1874	323	670	339	128	32
	1024	2149	328	2076	493	112	36
BVH traversal	128	2251	649	3007	1285	795	145
	1024	2498	585	4181	1467	1320	156
Ray caster	128	323	261	272	241	52	27
(comparison)	1024	323	260	273	242	43	23

Table 1: *Performance results of our methods and a ray caster used as a comparison method. Performance figures are given in millions of rays per second. N is the number of ambient occlusion samples per pixel. The results for two ambient occlusion radii are shown for each scene. The renderings above the table show the results using these radii, and were rendered using 128 ambient occlusion rays per pixel.*

regular pattern. We then reintroduce randomness by replacing each point with a random point within its Voronoi cell, and finally perform a few more relaxation passes. This two-step procedure ensures uniform overall density with suitable local randomness.

We chose to use equal-length shadow rays for all tests. This requires the smallest r for obtaining a desired apparent ambient occlusion radius, and is therefore the most efficient choice for algorithms whose execution time depends on r. Especially the rasterization method would suffer from having to compensate long falloff with larger radius. The set of shadow rays is the same for all methods, including the comparison ray caster.

The bounding volume hierarchy for our traversal-based method and the comparison ray caster is constructed using an ordinary top-down builder with greedy surface area heuristic. Each leaf node contains between 1 and 8 triangles. The same BVH is used for our method and the comparison ray caster.

All test renderings were performed in 1024×768 resolution, and every result is an average over multiple representative viewpoints for each scene. In the test scenes used, all triangles were treated as one-sided.

Table 1 shows the performance of our methods and the comparison ray caster in the three test scenes. The figures are in millions of ambient occlusion rays per second. The traversal-based method is the most efficient in all test cases, and for some cases substantially faster than the comparison ray caster. The rasterization-based method is faster than the ray caster in this test setup, but not as fast as the traversal-based method. On the other hand, it does not require a bounding volume hierarchy of the scene like the other methods.

The ray casting performance of the comparison ray caster is only slightly affected by the number of ambient occlusion rays cast per receiver point. Our methods mostly benefit from the increased number of rays, as most of the computation is shared between rays.

Figure 6 shows the frame rendering times for our methods and the comparison ray caster as a function of ambient occlusion radius r. We see that the the rasterization-based method scales worse than the other two, but the scalability of the traversal-based method is fairly good compared to the ray caster. We expect the ray tracer to ultimately win for long enough rays due to lost coherence between ambient occlusion rays, but the traversal-based method remains faster for a wide range of practically interesting values of r. It should be noted that the longest rays used in the plots are already fairly long, and the breakeven point between BVH traversal and ray caster is still far from being reached.

Table 2 details the effectiveness of the fragment culling tests in the rasterization method described in Section 4.2. Using all culling tests decreases the number of occlusion mask computations by a factor of about 6–7 in the test scene used, and more than triples the overall performance.

Figure 5 illustrates the rendering quality of our methods and the ambient occlusion volumes of McGuire [McG10]. A reference image was calculated with ray tracer using 1024 ambient occlusion rays per pixel, and the results of each method was compared to that. In both BVH traversal and rasterization methods, the majority of pixels are correct after quantizing the results to 8 bits, and the average error is

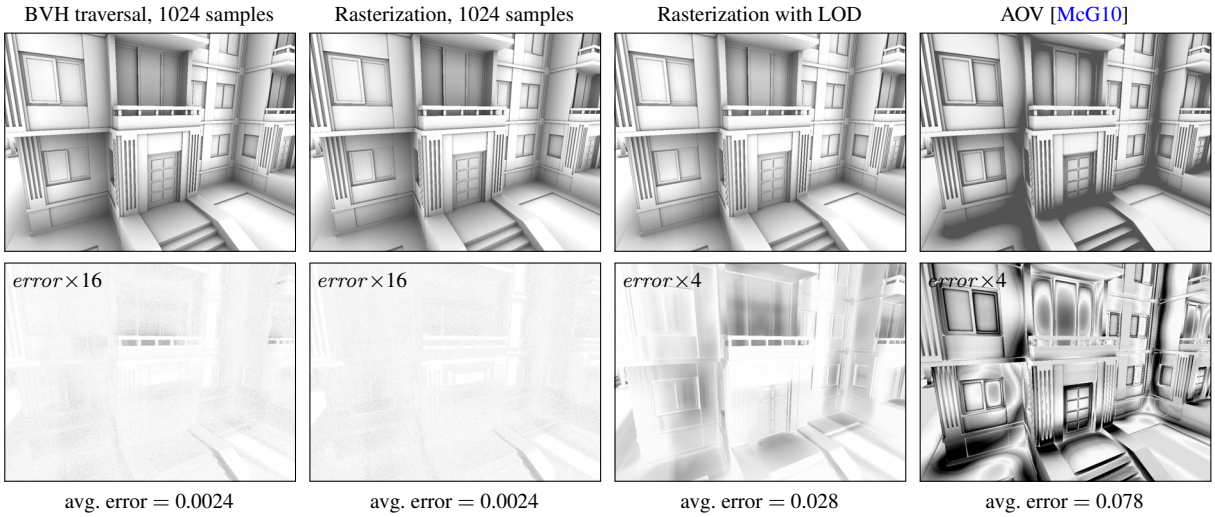


Figure 5: *Illustration of quality produced by our methods and the ambient occlusion volumes method of McGuire [McG10] in City scene with $r = 225$. For each method, the top image shows the rendering result, and the bottom image shows the approximation error, i.e., magnitude of deviation from ray-traced reference. In the error images, white indicates correct result, and black indicates absolute error of 1/16 (two leftmost images) or 1/4 (two rightmost images). Note that the two leftmost error images have been boosted four times more than the two rightmost ones to bring the errors visible.*

Culling tests	$r = 0.75$		$r = 2.25$	
	#frag	perf	#frag	perf
Depth test only	7.06	0.30	5.79	0.30
+ 1. Far depth	3.78	0.47	3.90	0.40
+ 2. Bounding sphere	3.22	0.52	3.11	0.47
+ 3. Triangle facing	2.11	0.73	1.95	0.70
+ 4. Plane distance	2.03	0.74	1.92	0.70
+ 5. Hemisphere facing	1.00	1.00	1.00	1.00

Table 2: *Effect of fragment culling tests in the rasterization-based method. The relative effect on fragment count (#frag) and performance (perf) is shown for two ambient occlusion ray lengths in Sibenik scene using 128 ambient occlusion rays per pixel. The first row shows the situation where occlusion computations are performed for all fragments that pass the hardware depth test, relative to all tests enabled. One additional culling test is enabled on each row. The row numbers refer to the list in Section 4.2.*

comparable to the quantization error. However, by boosting the error images significantly we can discern some systematic errors, which we suspect to be caused by discretization of the occlusion mask look-up tables. The level of detail optimization for the rasterization method incurs moderate estimation errors, but without comparison to ground truth these are quite hard to detect. The result of ambient occlusion volumes exhibits visible artifacts and has overall larger error than our methods.

Figure 7 shows the effect of using multiple levels of detail for the rasterization-based method as discussed in Section 4.3. As evidenced by the images, the rendering performance can be increased with tolerable loss of quality. The overall brightening is caused by loss of occlusion from thin, distant occluders. This effect is most visible in Hairball, where even the slightest amount of simplification changes the occlusion properties of the mesh considerably.

7. Future work

In addition to ambient occlusion, both of our methods are almost directly applicable to rendering soft shadows from spherical and disc-shaped area light sources. We have not ventured in this direction, but expect that significant speedups could be achieved.

Acknowledgements. Sibenik model courtesy of Marko Dabrovic.

References

[AL09] AILA T., LAINE S.: Understanding the efficiency of ray traversal on GPUs. In *Proceedings of High-Performance Graphics 2009* (2009), pp. 145–149.

[BS08] BAVOIL L., SAINZ M.: Screen space ambient occlusion. NVIDIA whitepaper. http://developer.download.nvidia.com/SDK/10.5/direct3d/Source/ScreenSpaceAO/doc/ScreenSpaceAO.pdf, 2008.

[BS09] BAVOIL L., SAINZ M.: Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH '09 Talks* (2009).

[Bun05] BUNNELL M.: Dynamic ambient occlusion and indirect lighting. In *GPU Gems 2* (2005), Addison Wesley, pp. 223–234.

[HJ07] HOBEROCK J., JIA Y.: High-quality ambient occlusion. In *GPU Gems 3* (2007), Addison Wesley, pp. 257–274.

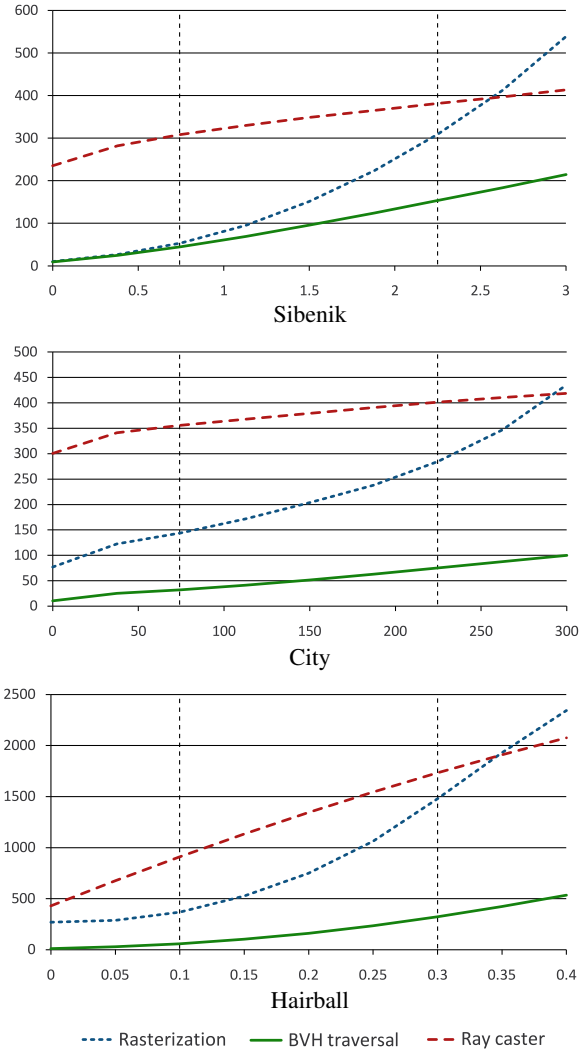


Figure 6: *Performance of our methods and the comparison ray caster. Horizontal axis is the ambient occlusion radius r, and vertical axis is the time in milliseconds to render a frame using 128 ambient occlusion rays per pixel. The two dashed vertical grid lines indicate the ambient occlusion radii used for the images and results in Table 1.*

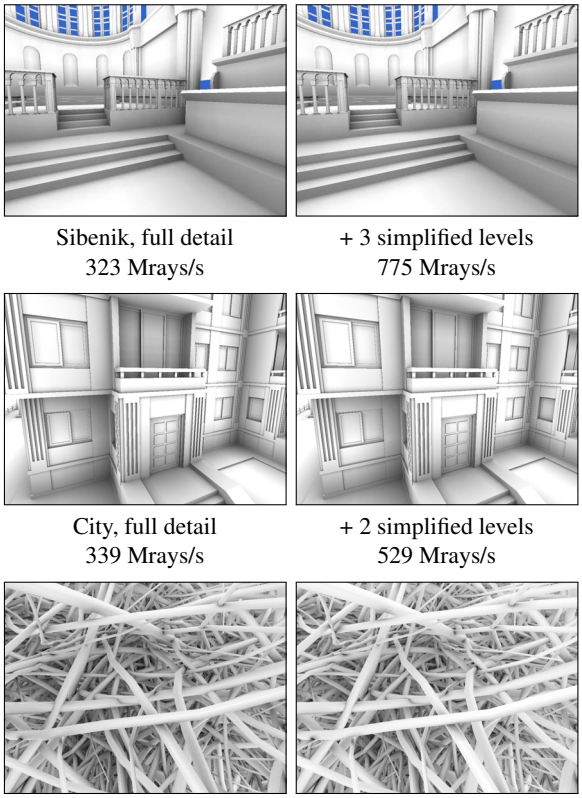


Figure 7: *The effect of varying level of detail according to distance in the rasterization-based method. The tests were run using 128 ambient occlusion rays per pixel and the larger r used in Table 1 for each scene. Allowed simplification error and minimum occlusion range were chosen empirically for the first simplified detail level and tripled in each subsequent level. The images on the left were constructed using original scene geometry, and the images on the right using multiple levels of detail. The resulting quality is perceptually identical, but rendering performance is increased.*

[KLA04] KAUTZ J., LEHTINEN J., AILA T.: Hemispherical rasterization for self-shadowing of dynamic objects. In *Proceedings of Eurographics Symposium on Rendering 2004* (2004), pp. 179–184.

[LA05] LAINE S., AILA T.: Hierarchical penumbra casting. *Computer Graphics Forum (Proc. Eurographics) 24*, 3 (2005), 313–322.

[Lan02] LANDIS H.: Production-ready global illumination. Course notes for SIGGRAPH 2002 Course 16, RenderMan in Production, 2002.

[McG10] MCGUIRE M.: Ambient occlusion volumes. In *Proceedings of High-Performance Graphics 2010* (2010).

[MFS09] MÉNDEZ-FELIU À., SBERT M.: From obscurances to ambient occlusion: A survey. *Visual Computer 25*, 2 (2009), 181–196.

[SS07] SCHWARZ M., STAMMINGER M.: Bitmask soft shadows. *Computer Graphics Forum (Proc. Eurographics) 26*, 3 (2007), 515–524.

[WBS07] WALD I., BOULOS S., SHIRLEY P.: Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics 26*, 1 (2007).

[ZIK98] ZHUKOV S., IONES A., KRONIN G.: An ambient light illumination model. In *Rendering Techniques 98 (Proceedings of Eurographics Workshop on Rendering)* (1998), pp. 45–55.