Visibility Acceleration using Efficient Ray Classification

Niels Billen
Philip Dutré

Report CW 695, 26 April 2016

KU Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Visibility Acceleration using Efficient Ray Classification

Niels Billen
Philip Dutré

Report CW 695, 26 April 2016

Department of Computer Science, KU Leuven

Abstract

Evaluating the visibility between two points is a common task in global illumination algorithms. Traditionally, a ray is traversed through a hierarchical data structure that is able to cull primitives which are unlikely to be hit by the ray. However, a great amount of time is spent in traversing the acceleration structure when a scene contains highly detailed objects or complex geometry like hair or foliage. While these objects contain a lot of disconnected geometry, their visibility is often very regular. In this paper, we present an algorithm which is able to exploit this regularity. In a preprocessing step we create a data structure for each object which classifies large parts of the objects ray space as either blocked or uninhibited by the object. In a rendering phase, large amounts of intersection tests can be avoided by performing a fast look-up in our data structure. We show that our method can be advantageous in scenes containing complex geometrical objects since our algorithm is able to classify a large amount of shadow rays, avoiding expensive intersection tests using a traditional acceleration structure.

Keywords : ray tracing, visibility, shadows
CR Subject Classification : I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism — Ray Tracing

Visibility Acceleration using Efficient Ray Classification

Niels Billen and Philip Dutré

Department of Computer Science, KU Leuven, Belgium

Abstract

Evaluating the visibility between two points is a common task in global illumination algorithms. Traditionally, a ray is traversed through a hierarchical data structure that is able to cull primitives which are unlikely to be hit by the ray. However, a great amount of time is spent in traversing the acceleration structure when a scene contains highly detailed objects or complex geometry like hair or foliage. While these objects contain a lot of disconnected geometry, their visibility is often very regular. In this paper, we present an algorithm which is able to exploit this regularity. In a preprocessing step we create a data structure for each object which classifies large parts of the objects ray space as either blocked or uninhibited by the object. In a rendering phase, large amounts of intersection tests can be avoided by performing a fast look-up in our data structure. We show that our method can be advantageous in scenes containing complex geometrical objects since our algorithm is able to classify a large amount of shadow rays, avoiding expensive intersection tests using a traditional acceleration structure.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Ray Tracing

1. Introduction

Visibility determination is a common operation in global illumination algorithms. We can distinguish two types of visibility queries. The first determines the first visible surface along a ray (so called *first-hit* visibility). The second determines whether two points are mutually visible (so called *any-hit* visibility). In this paper, we will focus on the latter.

Evaluating the visibility between two points is generally costly since it requires testing whether any primitive in the scene intersects the line segment connecting the points. Additional data structures are commonly used to accelerate the visibility evaluation. These acceleration structures are able to cull primitives from the scene which are unlikely to intersect the ray. As soon as an intersection is found, all other primitives can safely be ignored.

In this paper, we introduce an acceleration structure which accelerates visibility queries. Our acceleration structure is precomputed for objects in the scene and is able to classify for a large amount of rays whether the ray is blocked or uninhibited by the object, without the need of traversing a commonly used acceleration structure (e.g. a Kd-tree or a bounding volume hierarchy). This is advantageous since traversing a ray through such an acceleration structure can incur a lot of performance overhead. Complex geometry such as hair or foliage are an example where a lot of intersections and traversal steps are required to find a blocking primitive.

The contributions of this paper are the following:

- We present a new data structure which precomputes the visibility for objects in the scene;
- We propose a practical algorithm which can efficiently classify rays as blocked or uninhibited;
- We show that we are able to reduce the required amount of intersection tests up to 40% in scenes with specific distributions of geometry.

2. Related Work

Testing every primitive for intersection with every ray is impractical even for small scenes. A lot of research has focused on additional data structures which allow quick determination of the relevant primitives for intersection testing (for a survey, see [WMG*09]). When large numbers of rays are traced through a scene, the cost of constructing such an acceleration structure is amortised by the reduction in the intersection calculations.

To reduce the amount of intersection tests, acceleration structures partition either the three-dimensional space containing the primitives, the list of primitives, or the space of rays which can hit the primitives.

2.1. Space Partitioning

By partitioning the three-dimensional space in cells, efficiency is gained by traversing a ray front-to-back from cell to cell, testing only primitives contained in cells which are intersected by the ray. The ray traversal can be stopped after the first intersection is found, since no further cell can contain a closer intersection. A disadvantage of space partitioning is that a primitive can be referenced multiple times when it is contained in more than one cell. This results in a higher memory footprint and requires mailboxing techniques [APB87] to prevent loss in intersection performance when a ray intersects the same primitive multiple times.

Regular grids [FCK*88], [LD08] partition three-dimensional space in equally sized voxels, where each voxel only contains a small number of primitives. An advantage of a regular grid is that it can be constructed efficiently using rasterization to determine the overlap of primitives with voxels. However, the ray tracing efficiency of a grid severely suffers in sparse scenes where a lot of time is spent on traversing empty space.

Kd-trees [WH06] recursively partition three-dimensional space in two partitions using axis-aligned planes. Therefore, Kd-trees are able to adapt better than grids to scenes with irregularly distributed geometry and achieve logarithmic time efficiency.

2.2. Primitive Partitioning

An advantage of partitioning the list of primitives is that each primitive is referenced only once in the acceleration structure. This leads to predictable memory consumption. Furthermore, a ray will intersect a primitive at most one time when it is traversed through the acceleration structure, eliminating the need for mailboxing. The disadvantage is that the volumes of the nodes in the acceleration structure can potentially overlap. Ray traversal cannot stop after finding an intersection in a node since another overlapping node can still contain primitives with a closer intersection point.

Bounding volume hierarchies [Wal07] are an example of an acceleration structure that recursively divides the list of primitives in two partitions. Ray traversal starts from the root and the traversal only descends in a child node when the ray passes through it, achieving logarithmic time efficiency.

2.3. Ray Space Partitioning

Rays in three-dimensional space can be represented as points in five-dimensional space. Ray classification [AK87] constructs a hierarchical acceleration structure by subdividing the five-dimensional ray space. Compared to algorithms that partition space or the list of primitives, this eliminates the need to traverse a ray front-to-back through multiple cells of an acceleration structure. We only need to perform a look-up of which cell contains the five-dimensional ray to retrieve the primitives which potentially intersect the ray. However, the memory consumption of this technique is several orders of magnitude higher than traditional acceleration structures. This is due to the fact that a cell in the five-dimensional acceleration structure corresponds to an infinite beam in three dimensions. Each cell stores the primitives which overlap with this infinite beam. However, for a given beam, there are many other beams which are overlap with its extend. This causes primitives the be referenced multiple times in each beam.

The memory consumption can be improved by parametrising a ray in four-dimensional space either as the intersection of an infinite plane and a direction [KKCS98] or the entry and exit point on a sphere [MKYS07]. This reduces the amount of cells and the overlap between the cells. However, there is a trade-off in performance since cells now contain more primitives.

In contrast to previous ray space partitioning algorithms, our algorithm is specifically designed to accelerate the evaluation of visibility. The cells of our acceleration structure only store the visibility of the cell, instead of a complete list of primitives which overlap with the cell. This improves the memory consumption and creates opportunities for compression of our data structure.

3. Algorithm

Our goal is to construct an acceleration structure which is able to efficiently classify whether an infinite ray r is blocked or uninhibited by an object O. The visibility for an object O, consisting of multiple primitives, is a binary function which is defined as:

$$V_O(r) = \begin{cases} 0 & r \text{ intersects a primitive of } O \\ 1 & r \text{ does not intersect a primitive of } O \end{cases} \quad (1)$$

The domain of V_O can be reduced significantly by only considering the rays intersecting the tightest bounding box $\mathcal{B}_O$ of the object. The visibility function with respect to the object will always evaluate to 1 for rays which miss the bounding box.

The rays which hit $\mathcal{B}_O$ can be parametrised in four dimensions using the barycentric coordinates (s,t) and (u,v) of the two intersection points p_{entry} and p_{exit} with faces f_i and f_j (see Figure 1). Every pair of faces f_i, f_j spans a subset of all the infinite rays that hit $\mathcal{B}_O$.

Our algorithm is composed of the following steps:

1. During a preprocess, we construct our acceleration structure locally for objects O in the scene. We hierarchically subdivide the four-dimensional space spanned by each

pair of faces of $\mathcal{B}_O$ into cells and classify for each cell in the hierarchy whether all rays contained in the cell are either completely blocked, completely uninhibited or unclassified.

2. During the rendering phase, when a ray intersects the bounding box of an object, we perform a look-up in our data structure to classify the ray. When a ray can be classified as blocked or uninhibited by the object O, we do not have to resort to an exact visibility evaluation.

Each of these steps is described in more detail in the following subsections.

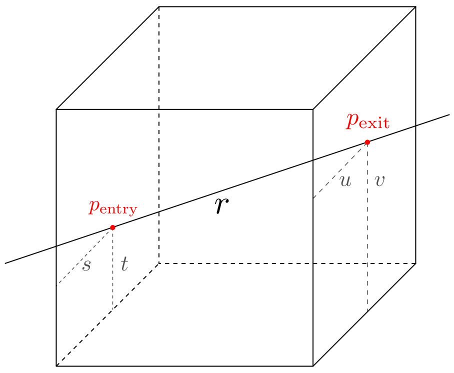


Figure 1: *Parametrisation of a ray to a four-dimensional subspace. An infinite ray r can uniquely be identified by the two faces it intersects and the barycentric coordinates of the intersection points* (s,t,u,v) *on those faces.*

3.1. Construction

Every ray intersecting $\mathcal{B}_O$ can be identified uniquely by the two faces it intersects and the barycentric coordinates $(s,t,u,v) \in [0,1]^4$ of its intersection points (see Figure 1).

Our algorithm constructs a tree over the four-dimensional space spanned by every pair of faces of $\mathcal{B}_O$. We start with the unity hypercube $[0,1]^4$ containing all the rays spanned by the two faces and recursively partition it in sixteen children of equal size by splitting the hypercube equally in all four dimensions.

To determine whether a node needs to be subdivided, we densely sample the node using rays stratified over the four-dimensional hypercube. We can distinguish between three cases:

1. all sample ray are blocked by the object: we stop the recursion and classify that the node is blocked by O;
2. all sample rays are uninhibited by the object: we stop the recursion and classify that the node is uninhibited by O;
3. the sample rays are partially blocked by the object: we continue with the recursion until the maximum depth is reached.

Limiting the maximum depth is required to end the recursion, otherwise it would continue endlessly around the edges of the object. A node remains unclassified when it cannot be classified as completely blocked or uninhibited by the object when the maximum recursion depth is reached.

Classifying the nodes using sampling introduces approximation errors. When an insufficient number of samples is used, a node can be erroneously classified as completely blocked or completely uninhibited. However, the error can be made arbitrarily small by using more sampling rays.

The four-dimensional octree can be seen as a hierarchy of three-dimensional shafts spanned between two faces of $\mathcal{B}_O$ (see Figure 2).

3.2. Rendering

During the rendering phase we first test whether a shadow ray intersects the bounding box of the object. When the shadow ray misses the bounding box, the object is certainly missed. When the shadow ray intersects the bounding box, we determine which faces of the bounding box are pierced and the barycentric coordinates of its intersection points. Next, we descent recursively in the hierarchy of nodes spanned between intersected faces until we reach the leaf node containing the shadow ray. We can again distinguish between three cases:

1. the node is completely blocked by the object: the visibility of the shadow ray evaluates to 0;
2. the node is completely uninhibited by the object: the visibility of the shadow ray for the object evaluates to 1;
3. the node is unclassified: in this case we need to resort to a commonly used acceleration structure (e.g. Kd-tree or bounding volume hierarchy) to evaluate the visibility.

When the ray completely pierces the bounding box, we only need to resort to a commonly used acceleration structure when a node is unclassified. In the other cases no intersection tests or traversal through such a hierarchy are required.

When a ray segment starts and/or ends within the bounding box (see Figure 2), it is possible that the ray segment is contained in a completely blocked node, while it does not hit the object. Therefore, we resort to a commonly used acceleration structure (e.g. bounding volume hierarchy) to evaluate the visibility of these rays.

3.3. Implementation Details

We implemented our octree as a pointer-less octree [Gar82], [Sam90], using 32 bits per node. Instead of using one pointer per child, an interior node only stores the pointer to the first of its children which are allocated continuously in memory. By using a smart encoding of the nodes in our octree, we can represent an interior nodes with sixteen leaf nodes as

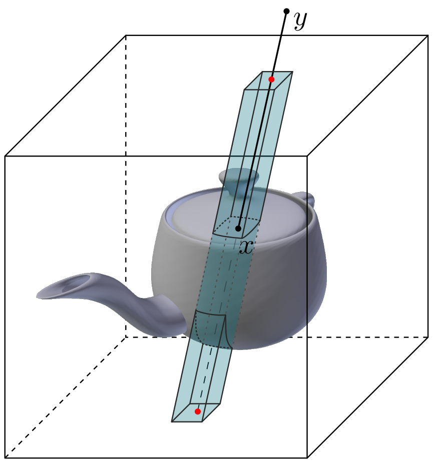


Figure 2: *The ray segment $\overline{xy}$ does not hit the object even though is located in a shaft which is completely blocked by the object. Therefore, when a ray segment starts and/or ends inside the bounding box of an object when can only classify rays which are laying inside an empty shaft.*

one *compressed interior node* of 32 bits in our octree. Further compression can be achieved by reducing the octree to a directed acyclic graph [KSA13]. Redundant subtrees are removed from our octree by allowing interior nodes to share pointers to the identical subtrees.

Table 1 shows the amount of compression achieved by reducing our pointer-less octree to a directed acyclic graph and using *compressed interior nodes*.

	Uncompressed	**Compressed**
Killeroo	1036 MiB	107 MiB
Sierpinski	1786 MiB	146 MiB
Hairball	4539 MiB	284 MiB
Forest	11165 MiB	794 MiB

Table 1: *Comparison of the compressed and uncompressed octree data structure for the test scenes (see Figure 3) at a recursion depth of 7.*

4. Results

We evaluated the performance of our algorithm for several scenes of increasing geometrical complexity (see Figure 3) and compared it to a traditional bounding volume hierarchy. To test the limits of our algorithm, we used various maximum subdivision depths and various amounts of ray samples to classify a node. Table 2 summarises the rendering time, amount of shadow ray intersections, memory consumption and the error for our tests scenes. Each image is rendered using 16 samples per pixel and the visibility of each light source is evaluated using 128 shadow rays.

4.1. Performance

For scenes with simple visibility events (Killeroo scene) and simple geometry (Sierpinski pyramid scene), we can see that our algorithm is able to outperform the bounding volume hierarchy by a slight margin. While the number of intersection tests is reduced up to 38%, the rendering time is only slightly reduced. This is due to the fact that a traditional acceleration structure is very efficient in finding an intersection in these scenes.

For scenes with complex visibility and complex geometry (Hairball and Forest scene), our algorithm is able to reduce rendering time up to 30% and the amount of intersections up to 51%. The bounding volume hierarchy requires a lot of traversal steps and intersection tests to find an intersection with the strands of the hairball and the leaves of the trees (see Figure 3). This is due to the fact that a ray traversed through the bounding volume hierarchy frequently misses the small geometry. In contrast, our algorithm can avoid these expensive traversals and intersection tests for many shadow rays by performing a simple look-up.

4.2. Recursion Depth

We expect more unclassified nodes in our data structure to be classified as completely blocked or uninhibited when the maximum subdivision depth is increased, because the rays contained in a node become more coherent with each subdivision. Our experiments verify this behaviour (see Table 2), since increasing the maximum subdivision depth of our algorithm results in reduced rendering times and intersection counts. However, the memory and build time increase for higher subdivision steps since more nodes need to be classified and stored in memory.

4.3. Error

When an insufficient number of rays is used to classify the nodes, a node which is partially blocked by an object can potentially be misclassified as completely blocked or completely uninhibited. As shown in Figure 4, this has a detrimental effect on the image quality. Fortunately, the error can be made arbitrarily small by sampling a node with more rays. However, this will result in increasingly larger build times (see Table 2)

Using less samples per node also results in reduced rendering time, build time and intersection counts. This is due to a combination of factors. First, fewer nodes have to be subdivided due to the misclassification, resulting in lower build times. Second, because less nodes are unclassified, we will have to resort fewer times to the bounding volume hierarchy, reducing the amount of intersections and the render time.

4.4. Build Time

The time required to build our acceleration structure, to a sufficient depth and with a small enough error, can be large. However, since our acceleration structure is created locally for objects in the scene, we can build it once for an object and store it on disk as an attribute to the object and reuse it during subsequent renders.

5. Conclusions

We presented a new acceleration structure to accelerate visibility queries by classifying large regions of the four-dimensional ray space of an object as either blocked or uninhibited.

Our results show that there exists trade-off between the performance and memory consumption of our algorithm. Increasing the recursion depth of our tree results in larger reductions in rendering time and intersection tests. However, a high recursion depth results in longer build times and higher memory consumption. While our algorithm only slightly outperforms a traditional acceleration structures for scenes with simple geometry, our algorithm is able to reduce the render time and amount of intersections significantly for complex scenes.

References

[AK87] ARVO J., KIRK D.: Fast ray tracing by ray classification. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1987), SIGGRAPH '87, ACM, pp. 55–64. URL: http://doi.acm.org/10.1145/37401.37409, doi:10.1145/37401.37409. 2

[APB87] ARNALDI B., PRIOL T., BOUATOUCH K.: A new space subdivision method for ray tracing CSG modelled scenes. *The Visual Computer 3*, 2 (1987), 98–108. URL: http://dx.doi.org/10.1007/BF02153666, doi:10.1007/BF02153666. 2

[FCK*88] FRANKLIN W. R., CHANDRASEKHAR N., KANKANHALLI M., SESHAN M., AKMAN V.: Efficiency of uniform grids for intersection detection on serial and parallel machines. In *New Trends in Computer Graphics*, Magnenat-Thalmann N., Thalmann D., (Eds.). Springer Berlin Heidelberg, 1988, pp. 288–297. URL: http://dx.doi.org/10.1007/978-3-642-83492-9_25, doi:10.1007/978-3-642-83492-9_25. 2

[Gar82] GARGANTINI I.: Linear octrees for fast processing of three-dimensional objects. *Computer Graphics and Image Processing 20*, 4 (1982), 365 – 374. URL: http://www.sciencedirect.com/science/article/pii/0146664X82900582, doi:http://dx.doi.org/10.1016/0146-664X(82)90058-2. 3

[KKCS98] KWON B., KIM D. S., CHWA K.-Y., SHIN S. Y.: Memory-efficient ray classification for visibility operations. *IEEE Transactions on Visualization and Computer Graphics 4*, 3 (Jul 1998), 193–201. doi:10.1109/2945.722294. 2

[KSA13] KÄMPE V., SINTORN E., ASSARSSON U.: High resolution sparse voxel dags. *ACM Trans. Graph. 32*, 4 (July 2013), 101:1–101:13. URL: http://doi.acm.org/10.1145/2461912.2462024, doi:10.1145/2461912.2462024. 4

[LD08] LAGAE A., DUTRÉ P.: Compact, fast and robust grids for ray tracing. *Computer Graphics Forum (Proceedings of the 19th Eurographics Symposium on Rendering) 27*, 4 (June 2008), 1235–1244. URL: http://www3.interscience.wiley.com/journal/121404245/abstract, doi:10.1111/j.1467-8659.2008.01262.x. 2

[MKYS07] MORTENSEN J., KHANNA P., YU I., SLATER M.: A visibility field for ray tracing. In *Computer Graphics, Imaging and Visualisation, 2007. CGIV '07* (Aug 2007), pp. 54–61. doi:10.1109/CGIV.2007.14. 2

[Sam90] SAMET H.: *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990. 3

[Wal07] WALD I.: On fast construction of sah-based bounding volume hierarchies. In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing* (Washington, DC, USA, 2007), Ray Tracing '07, IEEE Computer Society, pp. 33–40. URL: http://dx.doi.org/10.1109/RT.2007.4342588, doi:10.1109/RT.2007.4342588. 2

[WH06] WALD I., HAVRAN V.: On building fast kd-trees for ray tracing, and on doing that in o(n log n). In *IEEE Symposium on Interactive Ray Tracing 2006* (Sept 2006), pp. 61–69. doi:10.1109/RT.2006.280216. 2

[WMG*09] WALD I., MARK W. R., GÜNTHER J., BOULOS S., IZE T., HUNT W., PARKER S. G., SHIRLEY P.: State of the art in ray tracing animated scenes. *Computer Graphics Forum 28*, 6 (2009), 1691–1722. URL: http://dx.doi.org/10.1111/j.1467-8659.2008.01313.x, doi:10.1111/j.1467-8659.2008.01313.x. 1

	Acceleration structure			**Killeroo**	**Sierpinski**	**Hairball**	**Forest**
Total render time	BVH			44.4s	109.0s	122.5s	418.1s
	VC		depth 5	42.3s	104.4s	77.9s	281.0s
	VC	16 samples per node	depth 6	42.1s	100.4s	76.3s	263.4s
	VC		depth 7	41.7s	100.7s	75.9s	250.3s
	VC		depth 5	42.6s	106.4s	84.9s	337.6s
	VC	81 samples per node	depth 6	42.4s	103.9s	82.6s	319.6s
	VC		depth 7	42.0s	101.6s	80.8s	302.2s
	VC		depth 5	42.7s	106.6s	86.2s	367.2s
	VC	256 samples per node	depth 6	42.2s	103.5s	84.2s	348.1s
	VC		depth 7	42.0s	102.3s	82.9s	330.7s
Number of Shadow ray intersections	BVH			303.80M	1212.40M	2790.33M	4602.29M
	VC		depth 5	182.4M	998M	1190M	2722M
	VC	16 samples per node	depth 6	171.9M	955M	1133M	2444M
	VC		depth 7	164.6M	919M	1083M	2237M
	VC		depth 5	192.1M	1072M	1434M	3477M
	VC	81 samples per node	depth 6	180.8M	1015M	1371M	3163M
	VC		depth 7	171.8M	959M	1296M	2888M
	VC		depth 5	196.0M	1098M	1542M	3854M
	VC	256 samples per node	depth 6	183.1M	1036M	1474M	3551M
	VC		depth 7	173.6M	972M	1382M	3255M
Memory of the acceleration structure	BVH			8.0 MiB	4.8 MiB	175.8 MiB	400.4 MiB
	VC		depth 5	1.3 MiB	1.0 MiB	1.2 MiB	3.7 MiB
	VC	16 samples per node	depth 6	10.8 MiB	10.0 MiB	13.6 MiB	42.1 MiB
	VC		depth 7	83.1 MiB	97.4 MiB	179.9 MiB	486.9 MiB
	VC		depth 5	1.5 MiB	1.2 MiB	1.4 MiB	4.7 MiB
	VC	81 samples per node	depth 6	13.0 MiB	13.0 MiB	16.8 MiB	57.7 MiB
	VC		depth 7	102.3 MiB	134.0 MiB	254.4 MiB	709.3 MiB
	VC		depth 5	1.6 MiB	1.3 MiB	1.5 MiB	5.1 MiB
	VC	256 samples per node	depth 6	13.8 MiB	13.9 MiB	18.2 MiB	63.0 MiB
	VC		depth 7	107.2 MiB	145.7 MiB	283.6 MiB	793.4 MiB
Build time of the acceleration structure	BVH			0.1s	0.1s	1.4s	3.2s
	VC		depth 5	3.6s	3.3s	12.1s	32.7s
	VC	16 samples per node	depth 6	27.2s	24.6s	125.2s	300.5s
	VC		depth 7	195.9s	189.3s	1464.1s	2695.4s
	VC		depth 5	16.5s	11.4s	45.5s	141.7s
	VC	81 samples per node	depth 6	111.6s	83.7s	433.7s	1224.1s
	VC		depth 7	812.9s	719.7s	5029.3s	10670.0s
	VC		depth 5	52.5s	27.7s	120.7s	376.9s
	VC	256 samples per node	depth 6	320.0s	206.1s	1030.0s	3070.9s
	VC		depth 7	2351.8s	1978.1s	11559.1s	26392.2s
Classification error	VC		depth 5	0.770%	1.315%	1.082%	1.103%
	VC	16 samples per node	depth 6	0.759%	1.291%	1.147%	1.137%
	VC		depth 7	0.753%	1.249%	1.180%	1.162%
	VC		depth 5	0.0551%	0.096%	0.142%	0.163%
	VC	81 samples per node	depth 6	0.0548%	0.103%	0.155%	0.173%
	VC		depth 7	0.0549%	0.106%	0.160%	0.181%
	VC		depth 5	0.0062%	0.016%	0.058%	0.031%
	VC	256 samples per node	depth 6	0.0068%	0.019%	0.060%	0.035%
	VC		depth 7	0.0071%	0.023%	0.060%	0.038%

Table 2: *Performance of our Visibility Classification algorithm (VC) compared to a Bounding Volume Hierarchy (BVH).*

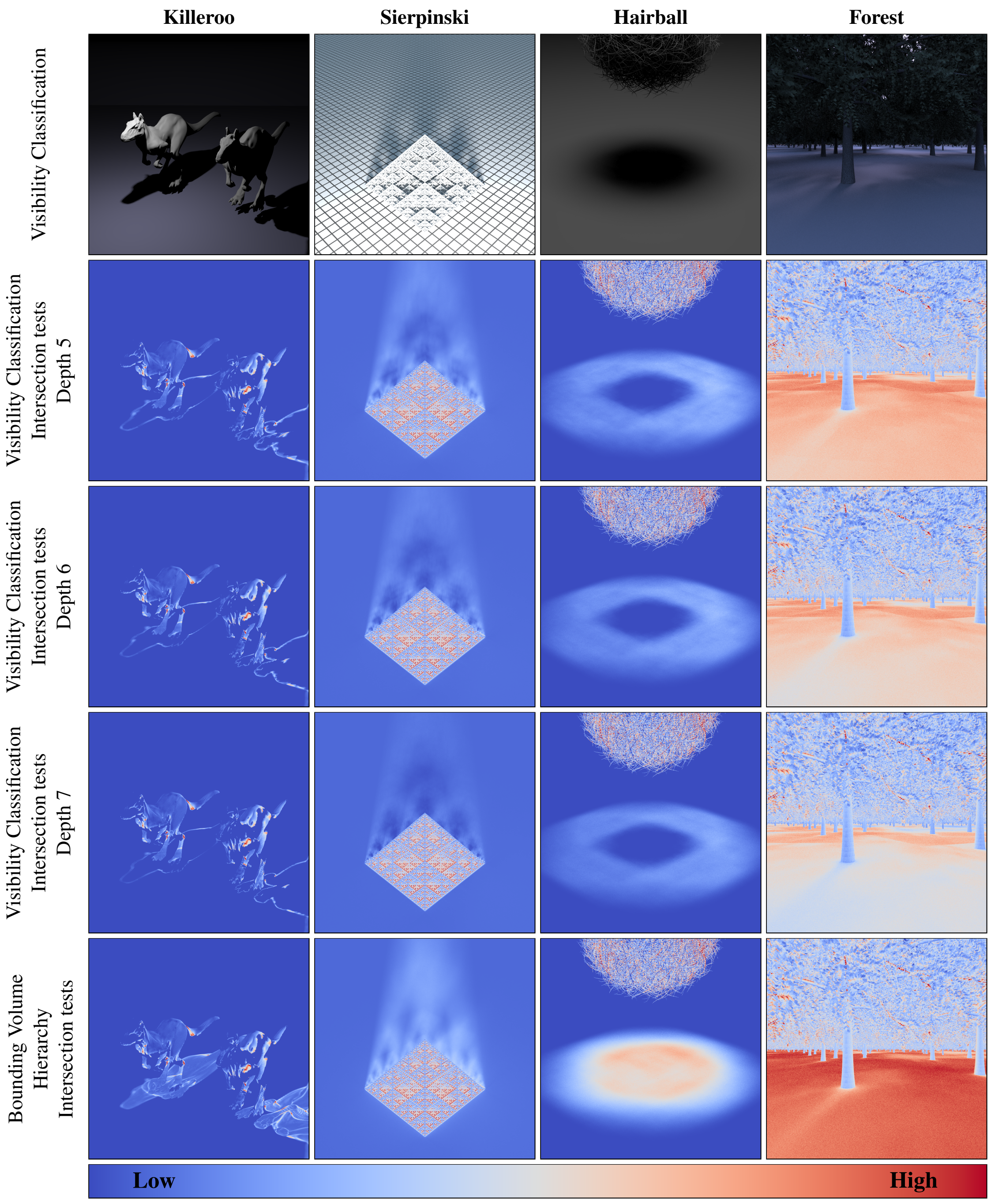
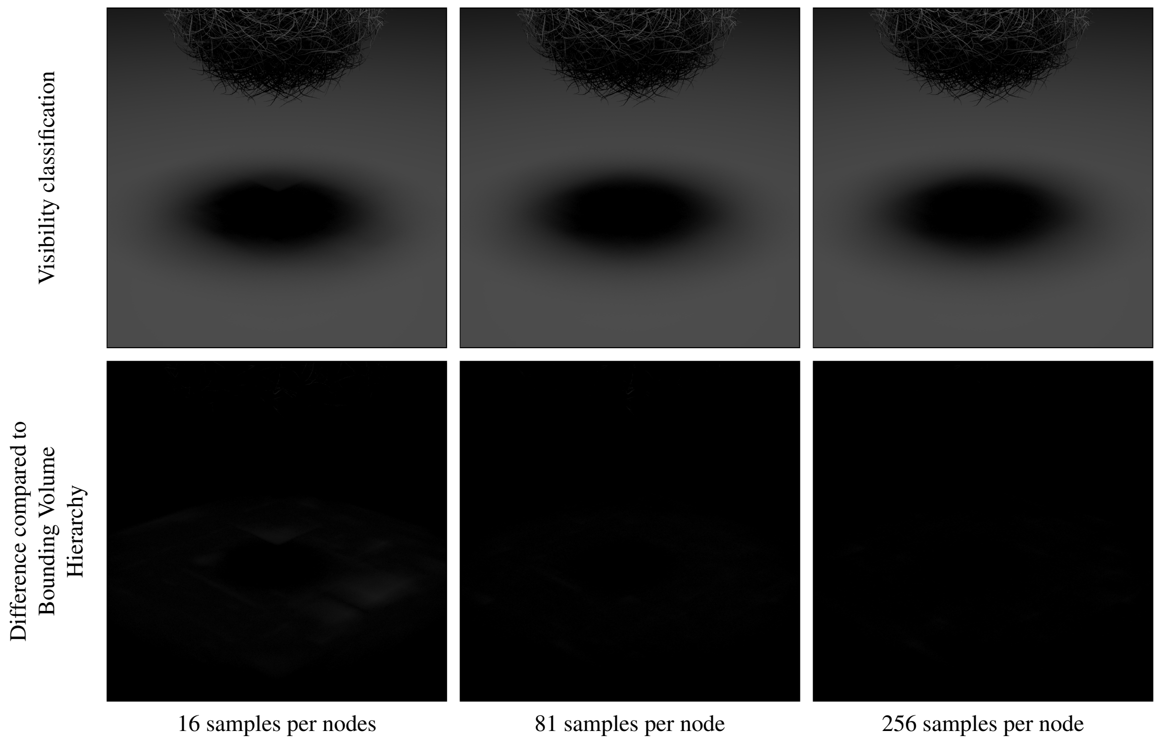

Killeroo
Sierpinski
Hairball
Forest
Visibility Classification
Visibility Classification Intersection tests Depth 5
Visibility Classification Intersection tests Depth 6
Visibility Classification Intersection tests Depth 7
Bounding Volume Hierarchy Intersection tests
Low
High

Figure 3: *Comparison of visibility classification against a bounding volume hierarchy. Every image is rendered using 16 samples per pixel and 128 shadow rays for every light source. The recursion depth of our acceleration structure varies from 5 to 7. The top row shows the test scenes rendered with our visibility classification algorithm. The second, third and fourth row show the amount of intersection tests required to evaluate the visibility in each pixel using our visibility classification. The last row shows this amount for the bounding volume hierarchy. The false-color images show that our algorithm outperforms the bounding volume hierarchy and that amount of intersection tests decreases with every additional recursion step in our algorithm.*



Visibility classification
Difference compared to Bounding Volume Hierarchy
16 samples per nodes
81 samples per node
256 samples per node

Figure 4: *Effect of the classification error on the image quality. Top row shows our visibility classification algorithm rendered with an increasing amount of samples per node. The bottom row shows the difference image with the bounding volume hierarchy. The error decreases as more samples are used to classify a node.*