Guided Visibility Sampling

Peter Wonka* Michael Wimmer† Kaichi Zhou* Stefan Maierhofer‡ Gerd Hesina‡ Alexander Reshetov§

*Arizona State University †Vienna University of Technology ‡VRVis Research Center §Intel Corporation

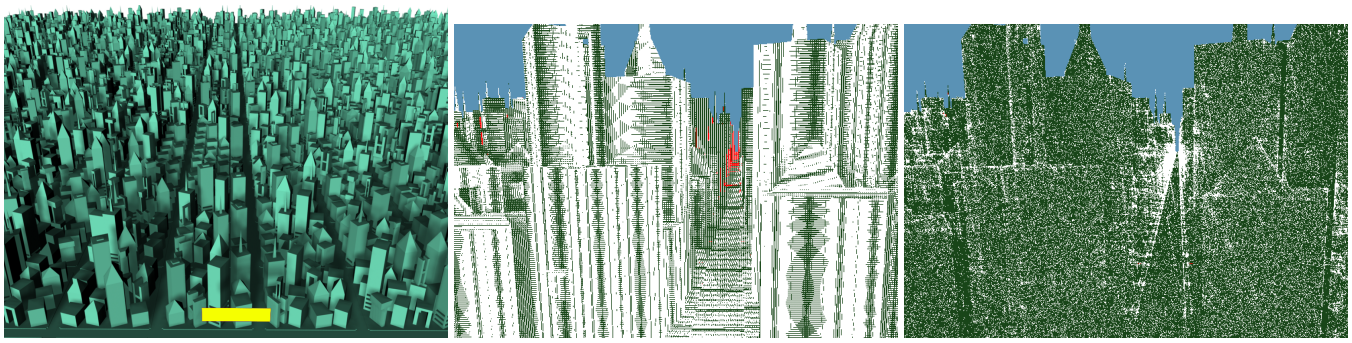

Figure 1: Visualization of sampling strategies (white pixels show a subset of the actual samples, missed geometry is marked red). Left: An urban input scene and a view cell (in yellow) for visibility sampling. Middle: Previous visibility sampling algorithms repeatedly sample the same triangles in the foreground while missing many smaller triangles and distant geometry. Right: Our solution is guided by scene visibility and therefore quickly finds most visible triangles while requiring drastically fewer samples than previous methods.

Abstract

This paper addresses the problem of computing the triangles visible from a region in space. The proposed aggressive visibility solution is based on stochastic ray shooting and can take any triangular model as input. We do not rely on connectivity information, volumetric occluders, or the availability of large occluders, and can therefore process any given input scene. The proposed algorithm is practically memoryless, thereby alleviating the large memory consumption problems prevalent in several previous algorithms. The strategy of our algorithm is to use ray mutations in ray space to cast rays that are likely to sample new triangles. Our algorithm improves the sampling efficiency of previous work by over two orders of magnitude.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Visible line/surface algorithms

Keywords: visibility, occlusion culling, PVS, visibility sampling

1 Introduction

Visibility is a fundamental problem in computer graphics: visibility computations are necessary for occlusion culling, shadow generation, inside-outside classifications, image-based rendering, motion planning, and navigation, to name just a few examples. While visibility from a single viewpoint can be calculated quite easily, many applications require the potentially visible set (PVS) for a region in space, which is, unfortunately, much more complicated. A number of excellent from-region visibility algorithms exist, but most of them are only applicable to a limited range of scenes, require complex computations, and sometimes significant amounts of memory. Therefore, sampling-based solutions have become very popular for practical applications due to their robustness, general applicability, and ease of implementation. In this paper we will improve upon previous sampling-based algorithms by significantly improving the sampling efficiency, i.e., the number of samples required to detect a certain set of visible polygons.

To motivate our design choices, we will look at two key aspects of any visibility algorithm: the behavior of the algorithm in ray space, and the data structure used to store and acquire visibility information.

Figure 2 illustrates the concept of ray space in 2D. Given a view cell, shown as edge parameterized with s, and a scene with objects shown in grey, we can compute visibility by considering all rays from the view cell to a plane behind the scene, parameterized with t. For a 2D scene, this is a 2D set of rays; for a 3D scene this is a 4D set of rays. If this set of rays is sampled densely enough, we will have a good visibility solution.

The inefficiency of a pure regular sampling approach as shown in Figure 2 is that the same surfaces are sampled over and over again (note that the definition of regular depends on the parameterization of ray space!). Therefore, it would be beneficial if we could only sample areas that have not been sampled before. This is shown in Figure 3, where after an initial orthogonal sampling, only few additional rays are needed to find all visible surfaces. In total, little more than a 1D subspace of the 2D ray space needs to be explored in this example. This is due to the spatial coherence of visibility. In this paper, we exploit this coherence: starting from stochastically sampled points, we grow lower-dimensional subspaces of ray space using the newly introduced strategies of *adaptive border sampling* and *reverse sampling*, which are guided by the properties of scene visibility.

*{peter.wonka|kaichi.zhou}@asu.edu, Tempe, AZ 85287-0112
†wimmer@cg.tuwien.ac.at, 1040 Vienna, Austria
‡{sm|hesina}@vrvis.at, 1220 Vienna, Austria
§alexander.reshetov@intel.com, Santa Clara, CA 95054

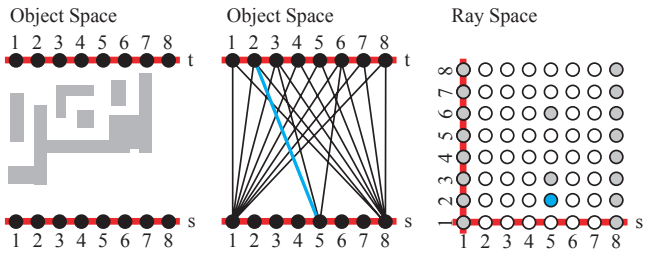

Figure 2: Sampling in object and ray space. Left: a scene with a set of objects. A view cell is shown as a line segment parameterized with s. We are interested in all rays that intersect the view cell and a second line segment parameterized with t. Middle: Shows a subset of the possible rays. One ray is highlighted in blue. Right: A depiction of the discrete ray space. Any ray in the middle figure corresponds to a point in ray space. The blue point corresponds to the blue ray in the middle figure.

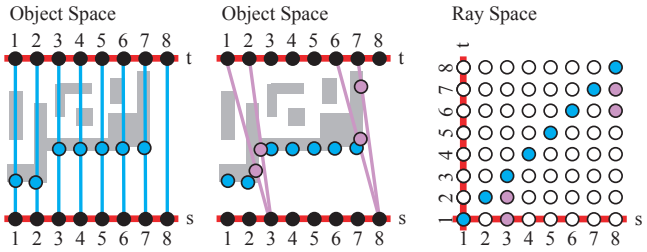

Figure 3: Left: The scene sampled orthogonally. Middle: Additional samples to capture oblique surfaces. Right: The rays used to sample the scene are shown in corresponding colors.

The second key aspect of a visibility algorithm is what data structure is used to store visibility information. The most complete, but also complex, way is to store 4D ray space. For large scenes, this entails prohibitive levels of memory consumption. Conservative algorithms often store the shadow volume, whereas sampling algorithms use the volume of 3D space that has not been sampled yet (the so-called void volume, Figure 4); but these data structures still require several times the memory taken by the scene description itself. Alternatively, the boundary of the void volume (the void surface [Pito 1999]) can be used, which is easy to sample from one point in space, but difficult to manipulate. In this paper, we do not store visibility information beyond the PVS at all, relying on our new reverse sampling approach to penetrate the void surface based on the current sample only.

The key contribution of this paper is an intelligent sampling algorithm that drastically improves the performance of previous sampling approaches by combining random sampling with deterministic exploration phases. The algorithm requires little memory, is simple to implement, accepts any triangular test scene as input, and can be used as a general purpose visibility tool.

2 Overview

2.1 Problem Statement

We consider visibility problems that are posed as follows: As first input we take a three-dimensional scene consisting of a set of triangles, TS. We do not rely on connectivity information, volumetric

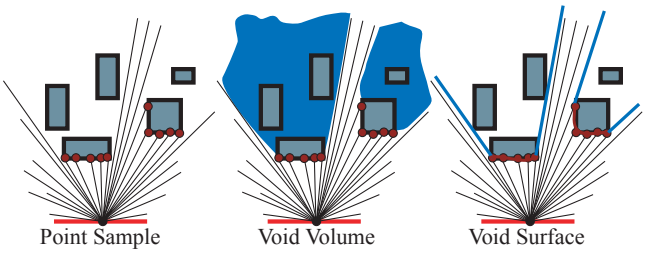

Figure 4: Representing visibility from a single point. Left: independent samples. Middle: the void volume. Right: the void surface.

objects, or large polygons as potential occluders (such a set of triangles is often called triangle soup). As second input we consider a subset of ray space Ω, usually defined by the rays emanating within a 3D polyhedron called *view cell* and intersecting the bounding box of the scene. A ray can be defined by a starting point and a direction. Using TS and Ω, we can define a visibility function $v : \Omega \rightarrow TS$, so that each ray in Ω maps to the triangle in TS that it intersects first.

The exact solution of the visibility problem is the range of this function, $v(\Omega) \subseteq TS$, also called exact visible set EVS. Our algorithm is *aggressive* ([Nirenstein et al. 2002]), i.e., it calculates a potentially visible set $PVS \subseteq EVS$.

Our algorithm can be used to solve the visibility problem in different applications (see Section 5.6). A *usage scenario* to keep in mind for the following exposition is a visibility preprocessing system for real-time rendering: the *view space* (set of possible observer locations) is partitioned into view cells. In a *preprocessing* step, our algorithm is used to calculate and store a PVS for each view cell (note that only its boundary polygons are taken into account, since any ray leaving the view cell can be represented by a ray on the boundary). At runtime, the view cell corresponding to the current observer location is determined, and only the objects in the associated PVS are sent to the graphics hardware, leading to significant savings in rendering time.

2.2 Algorithm Overview

The algorithms in this paper are based on ray shooting and assume the capability to trace a ray x and compute its first intersection with a scene triangle $t \in TS$, i.e., to compute the triangle $t = v(x)$ (fast ray tracers include OpenRT [Wald et al. 2003] and the recently presented MLRTA [Reshetov et al. 2005]).

The idea of a sampling solution is to select a sequence of rays $\mathbf{X} = x_i$, trace the ray and add the triangle $v(x_i)$ to the visibility set PVS.

In this paper, we will address the problem on how to sample *efficiently*, that is how to improve the chances of finding new triangles. We will start with one of the most popular sampling strategies, random sampling (Section 3.1). Then we will show how to use visibility information from previous samples to construct intelligent sampling strategies based on ray mutation to complement random sampling:

Adaptive Border Sampling is an algorithm to quickly find nearby triangles by sampling along the borders of triangles previously found to be visible (Section 3.2).

Reverse Sampling is an algorithm to sample into regions in space that are likely to be near the boundaries of visible and invisible space, i.e., the void surface (Section 3.3).

In Section 3.4, we will show how to combine the different sampling algorithms in order to obtain *guided visibility sampling*, a complete hybrid random and deterministic sampling algorithm. The algorithm is called guided because both sampling strategies are guided by visibility information in the scene (see Section 5.4 for a more detailed discussion).

3 Visibility Sampling

All rays in the scene form a 5D space. A ray x has a starting point x_p (3D) and a direction x_{dir} (2D). A typical visibility query is to give a region R in 3D space and ask what is visible along the rays leaving the region (view cell). While this defines a 5D set of rays, we only need to consider a 4D set of rays in practice; the rays starting at the boundary δR of the viewing region. Additionally, all triangles intersecting R are classified as visible.

3.1 Random Sample Generator

The random (or pseudo-random) sampling algorithm selects a sequence of random samples $X = x_i$ from the scene. The probability distribution for each new sample $p(x_i)$ is independent of all previous samples $x_1,...,x_{n-1}$. The question of sampling uniformity for random sampling has been explored in the context of form-factor computation [Sbert 1993]. We sample the position and ray direction uniformly using the following formulae:

$$u = \xi_1, \quad v = \xi_2, \quad \phi = 2\pi\xi_3, \quad \theta = \arcsin\xi_4,$$

where the ξ_i are independent Halton sequences [Niederreiter 1992], and (u,v) are the normalized coordinates on a view cell face. While random sampling alone suffers from similar inefficiencies as regular sampling (see Section 1), it will be used to seed the more efficient strategies described next.

3.2 Adaptive Border Sampling

This sampling algorithm is a deterministic ray mutation strategy that covers most of the ground work to make our system successful. This strategy leaves the ray starting point x_p on the view cell fixed while covering adjacent triangles in object space, practically constructing a local visibility map [Bittner 2002] from the selected view cell point.

The key idea of this sampling strategy is that it adapts the sampling rate to the geometric detail of the surface (see Figure 5). Therefore, it is unlikely that subpixel triangles are missed, which is a problem for methods that sample objects regularly. The method performs especially well for the most frequent case of a connected mesh, but does not assume or use any connectivity information. The connected regions are discovered in the random sampling step (therefore, scenes with many small disconnected meshes like trees remain a challenge for the approach).

The algorithm proceeds as follows. If a triangle $t = (p_1, p_2, p_3)$ is hit for the first time by a sample ray $x = (x_p, x_{dir})$, we enlarge t by a small amount to obtain an enlarged polygon t' , and adaptively sample along its edges (Figure 5).

For each edge, we use two rays x_l and x_r, and the corresponding samples $hit(x_l)$ and $hit(x_r)$ in world space. If the rays x_l and x_r hit different triangles, we recursively subdivide the edge, up to a

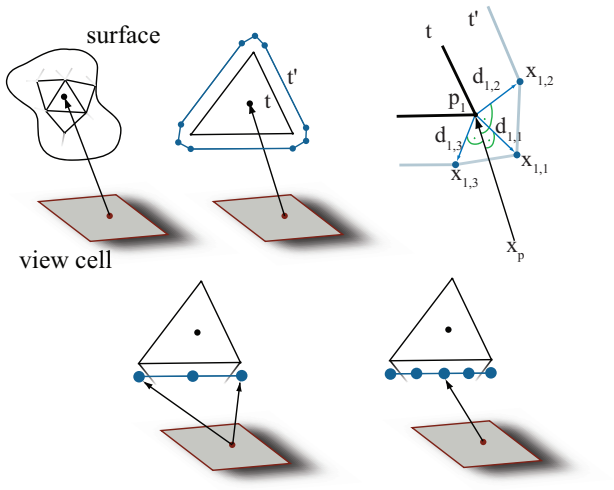

Figure 5: Adaptive border sampling: Top: If we hit a new surface, we sample nearby points on the border polygon t'. Bottom: Adaptive subdivision of an edge of t'.

given threshold. At this point, we also detect depth discontinuities between the new samples and the original sample on the triangle, which is already a part of reverse sampling as described in the next section.

The actual method used for *border enlargement* deserves attention. In order not to miss any adjacent triangles, the border polygon t' should be as tight as possible. On the other hand, if it is too tight, t will be hit again due to the numerical precision of ray shooting. If the enlargement were done in object space, this would happen for near edge-on or very distant triangles. We therefore enlarge t in ray space by rotating rays to the vertices of t to their new positions on t' by a small angle. This is more robust because it depends neither on the distance of the triangle nor on its orientation, but only on the numerical precision of the ray representation. In practice, this means that for each vertex, the new vertices are put on a plane perpendicular to the ray.

The shape of t' is chosen so that the ray space distance to t is fairly constant. This is not possible with only 3 vertices, since sliver triangles would lead to singularities. We therefore chose t' to be a polygon of 9 vertices. For each vertex p_i of t, three vertices $p_{i,j}$ on t' are generated. Two vertices are generated each on a vector $d_{i,j}$ perpendicular to the ray and to one of the adjacent edges, respectively. The third is the midpoint of the other two, pushed away from t along the angle bisector $d_{i,i}$:

$$\begin{aligned}
d_{i,i+1} &= N((p_i - x_p) \times (p_{i+1} - p_i)) \\
d_{i,i-1} &= N((p_i - x_p) \times (p_i - p_{i-1})) \\
d_{i,i} &= N(d_{i,i-1} + d_{i,i+1}) \quad \text{if} \quad d_{i,i-1} \cdot d_{i,i+1} > 0, \quad \text{else:} \\
&\quad N((p_i - x_p) \times d_{i,i-1} + d_{i,i+1} \times (p_i - x_p)) \\
p_{i,j} &= p_i + \varepsilon \cdot |p_i - x_p| \cdot d_{i,j}
\end{aligned}$$

where $N(v)$ is the vector normalization operator. $d_{i,i}$ is chosen to be numerically robust. For backfacing triangles, the $d_{i,j}$ need to be inverted.

Adaptive border sampling efficiently explores connected visible areas of the input model from a single viewpoint along a 1D curve in ray space (see Section 5.4). However, it cannot penetrate into gaps

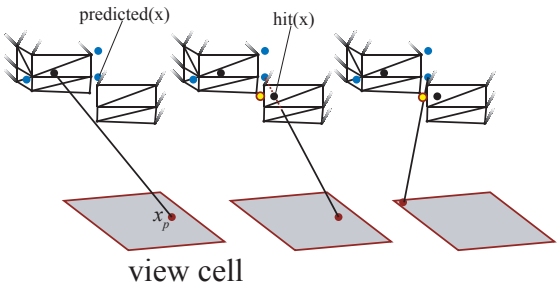

Figure 6: Reverse Sampling: Left: initial hit on triangle t. Middle: the new ray to $predicted(x)$ is blocked by a much closer triangle t'. Right: Reverse sampling mutates the starting point on the view cell so that the ray passes through p_{new} (yellow) and reaches $predicted(x)$.

visible only from other portions of ray space. This is handled by reverse sampling.

3.3 Reverse Sampling

This algorithm is a deterministic mutation strategy that allows penetrating into as yet uncovered regions of space. Note that this cannot be done perfectly: finding the actual void volume is equivalent to the original visibility problem. However, the adaptive sampling process gives good candidate locations for further sampling rays, namely at discontinuity locations. This strategy works by changing the starting point of the ray instead of its direction.

A discontinuity is detected during the adaptive sampling of an edge by comparing the distance of the ray origin to the actual hitpoint $|hit(x) - x_p|$ with the distance to a "predicted" hitpoint $|predicted(x) - x_p|$. The predicted hitpoint is just the intersection of the ray x with the plane of the original triangle t. If the new hitpoint is considerably (Δ) closer, i.e.,

$$|predicted(x) - x_p| - |hit(x) - x_p| > \Delta,$$

the ray is obviously occluded by a closer triangle. Note that we do not check the reverse case (jump from closer to farther triangle) as this will be detected when doing adaptive border sampling for the farther triangle. We calculate a mutated ray from a different view cell position to the predicted hitpoint so that it passes by the occluding triangle. For this, the plane $p = (x_p, hit(x), hit(x_{old}))$ is intersected with the newly found triangle (x_{old} is the previous ray from which x was generated). On the intersecting line, we select a point p_{new} which lies just outside of the new triangle. The mutated ray is now constructed with $x_{new,dir} = predicted(x) - p_{new}$ as direction vector, and $x_{new,p} = intersect(viewcell, line(p_{new}, predicted(x))$ as origin (see Figure 6). If the new ray is not contained in the ray space Ω (i.e., it does not intersect the view cell), however, it is discarded.

The new ray x_{new} is now treated as independent ray, and the triangle it intersects will be added for adaptive border sampling like any other triangle, but this time with the new view cell origin.

For the 2D example in Figure 3, reverse sampling corresponds to a horizontal movement in ray space.

3.4 Combining the Different Sampling Algorithms

The sampling strategies presented so far can be combined into an extremely efficient guided visibility sampling algorithm. Its two main components are a sample generator for exploring the ray space with independent random samples, and a sampling queue for propagating the ray using adaptive border sampling and reverse sampling. The algorithm is described by the following pseudocode:

```
main()
 while not finished
  (xp,xdir) = generate_random_ray()
  handle_ray(x)
  while not queue.empty()
   adaptive_border_sampling(
    queue.dequeue())

handle_ray(x)
 if v(x) notin PVS
  PVS += v(x)
  queue += x

adaptive_border_sampling(x)
 t' = enlarge(v(x), ε)
 for each p in t'
  handle_ray((xp, p-xp))
 for each edge (pl, pr) in t'
  subdiv_edge(pl, pr)

subdiv_edge(pl, pr)
 x = (xp, pl-xp)
 y = (xp, pr-xp)
 check_discontinuity(x)
 check_discontinuity(y)

 if v(x) = v(y) or |hit(x)-hit(y)| < ε
  return
 else
  p = (pl + pr)/2
  handle_ray((xp, p-xp))
  subdiv_edge(pl, p)
  subdiv_edge(p, pr)

check_discontinuity(x)
 if |predicted_hit(x) - xp)| -
    |hit(x) - xp| > thresh)
  xnew = reverse_sampling(x)
  if start(xnew) in view cell
   handle_ray(xnew)
```

3.5 Termination criteria

Depending on the application requirements, there are several options regarding when to stop casting rays for a view cell: a) a fixed criterion, allocating a number of rays or an amount of time for the computation of each view cell, or b) an adaptive criterion, terminating if the number of newly found triangles per a certain number of samples falls below a threshold, or most preferably, c) a combination of both. A typical example for such a criterion is: stop the iteration when not more than 50 new triangles are found for 1M rays, or when a total of 10M rays has been shot, whichever comes first.

4 Results

4.1 Overview

To compare the efficiency of our algorithm to previous work, we use the following algorithms: GVS, our guided visibility sampling algorithm with adaptive border sampling (ABS) and reverse sampling (RS); and RAND, random sampling (in GVS, a value of epsilon of 5e-5 was used for enlarging triangles). We have dedicated separate subsections to the comparisons with NIR, the main other existing visibility sampling method published by Nirenstein and Blake [2004] (mainly because this algorithm has a slightly different goal than GVS); and EXACT, Bittner's [2003] exact visibility algorithm.

The test scenes selected are (see Figure 7 and Table 1): PPLANT, the complete UNC Power Plant model; CITY, a city model of the ancient city of Pompeii generated using the CityEngine [Müller et al. 2006]; CANYON, a dataset of the Grand Canyon; and CUBES, a simple scene of random cubes. The tests were conducted on an Intel Pentium4 3.2GHz with 4GB of main memory. The graphics card for NIR was an NVIDIA GeForce 7900GTX 512MB.

Scene	triangles	size	vc
CANYON	2,242,504	10x5x3 km	140x72 m
CITY	5,646,041	320x312x9 m	15x2.2 m
PPLANT	12,748,510	200x61x81 m	2x1.3 m
CUBES	24,000	100x100x100 m	1.5x1.5 m

Table 1: Statistics for all scenes. vc denotes the average size of view cells used in the model.

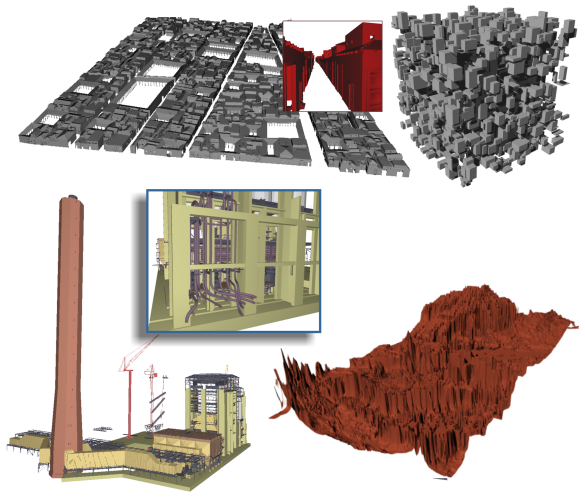

Figure 7: Top left: CITY. Top right: CUBES. Bottom left: PPLANT. Bottom right: CANYON. Inlays: view from a view cell.

For GVS and RAND, we used Intel's multi-level ray tracer (ML-RTA [Reshetov et al. 2005]), which allowed sampling rates between about 800K/s and 1200K/s, with peaks up to several million samples/s. The sampling rate depends on the scene type (not so much on the size—PPLANT had a higher sampling rate than CANYON, for example), and on the coherence of the rays (with random samples and ABS samples being faster depending again on the scene). The overhead of the sampling selection process varied between 5 and 15%, depending on the relative distribution of random, ABS and RS rays.

4.2 Asymptotic behavior

We first analyze the *theoretical properties* of the algorithms in terms of their sampling behavior, i.e., on a *sample-by-sample basis*, since this is the only comparison that does not depend on the individual implementation. Since we do not have an exact visibility algorithm that runs in reasonable time on larger scenes, we can only study their asymptotic behavior on a small number of view cells. Figure 8 provides a detailed analysis of the CANYON scene, graphing the pixel error (calculated by counting the false pixels in a large number of random renderings [Nirenstein and Blake 2004]) and the number of triangles found over the number of samples for GVS and RAND. The top left image shows that GVS converges linearly as long as the deterministic strategies (ABS and RS) can be used for most triangles. The black dot on each view cell curve shows when our termination criteria terminates the PVS search (we used 50 or less new triangles found per 1M samples). It can be seen that this happens in a fairly well converged state already. The graph also shows that the behavior is very similar for all view cells. The length of the linear segment only depends on the final PVS size.

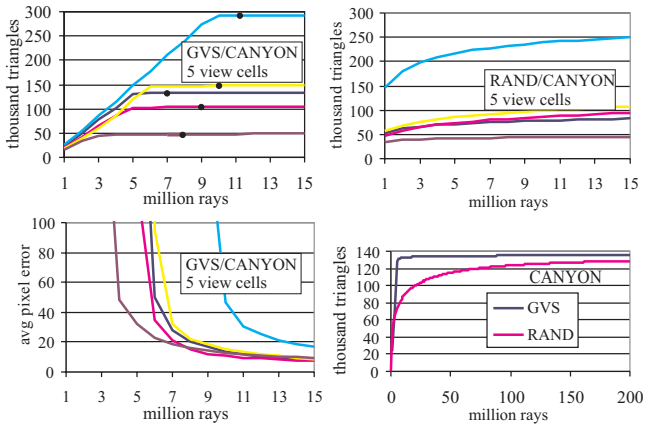

Figure 8: Detailed asymptotic analysis for 5 view cells for the CANYON model (see text for details). The pixel errors are measured on a 1000x1000 screen, equivalent to 10^{-4}%. The plots in the lower right image show the blue view cell from the other images.

The top right figure shows RAND in comparison. The convergence of RAND looks mainly logarithmic and has a very quick falloff after an initial strong phase. It is especially noteworthy that even at 15M samples, when GVS has already long converged, RAND is still 50K triangles behind GVS for most view cells. The bottom right figure analyzes this behavior on an even larger scale for the dark blue view cell from the other graphs. This figure confirms the quick convergence of GVS, and shows that even after 200M samples, RAND is still several thousand triangles behind GVS. It can be concluded that it would take RAND several orders of magnitude longer to find a PVS that GVS can find with about 7M to 8M samples.

Finally, the bottom left figure proves that the PVS size correlates strongly to average pixel error, and that the termination criterion discussed above works well in practice, bringing the average pixel error below 30 pixels on a 1000x1000 screen. Due to the better distribution of initial samples, RAND shows lower average pixel error in the phase where GVS searches mainly deterministically. However, to reach the same pixel errors as provided by GVS in a converged state, RAND has to calculate a similar number of triangles in the PVS, leading to the same observation as before, that similar pixel error requires orders of magnitude more samples than with GVS.

4.3 Practical results

Next, we demonstrate that these findings generalize to a larger number of scenes, and provide a *practical analysis* including running times. Table 2 summarizes our findings.

We used the same convergence criterion of 50 triangles per 1M samples for GVS, and a constant 150M rays for RAND. It can be seen that this results in very similar average and maximum errors for both algorithms. However, the running times differ by more than an order of magnitude, which reflects the good convergence behavior of GVS with respect to RAND shown above. The table also lists results for NIR, which are discussed in the following subsection. In addition to the error we also give the size of the PVS in terms of the whole model size (an EVS was not available in reasonable time). A higher value means a more accurate solution.

	Alg.	Avg.Err.	Max.Err.	time	PVS
CANYON	GVS	35	239	7.9s	6.7%
	RAND	67	828	183s	6.3%
	NIR512	2,191	8,215	6.8s	5.8%
	NIR1024	519	2480	11.6s	6.3%
CITY	GVS	22	230	6.1s	1.1%
	RAND	70	625	69s	0.4%
	NIR512	1,292	8,655	5.4s	0.2%
	NIR1024	631	8,965	8s	0.4%
PPLANT	GVS	23	211	30s	0.8%
	RAND	69	825	129s	0.5%
	NIR512	3,225	17,169	24s	0.4%
	NIR1024	1,549	8,317	25.9s	0.6%

Table 2: Statistics for all scenes averaged over a number of view cells. We used a threshold of 50 or less triangles found per 1M samples to cut off computation for GVS. For RAND, we shot 150M rays for each test. Errors are number of false pixels on a 1000x1000 screen, which corresponds to 10^{-4}%. Results for NIR are given at 512x512 and 1024x1024 resolution. The intrinsic parameters had to be adjusted for each scene to obtain reasonable results (see the comments in Section 4.4). The last column shows the average size of the calculated PVS as a percentage of the whole model.

4.4 Comparison to hardware sampling

Nirenstein and Blake [2004] recently published an interesting adaptive regular sampling algorithm which uses graphics hardware to adaptively sample hemi-cubes on the view cell. It is difficult to directly compare NIR and GVS. On the one hand, they are both based on the same atomic operation—taking a visibility sample. This is because sampling with graphics hardware and with a ray tracer is functionally practically equivalent due to the available sub-pixel precision (usually 12 bit) in current graphics hardware.

The time complexity, however, differs significantly between the two algorithms. The time complexity of ray casting is linear in the number of rays and, due to spatial data structures, logarithmic in the number of objects. In practice, we have also observed a strong dependence on the type of the scene and the implementation of the ray tracer, which makes general predictions on the scalability with respect to scene size very hard.

For graphics hardware, the basic operation is an item-buffer render. Depending on whether a particular view is mostly fill or geometry limited, the resolution of this item buffer has more or less impact on the rendering time. Our implementation of NIR rendered models from multiple vertex buffers stored directly in video memory, which provided triangle throughput near the theoretical maximum on the card we used (between 130 and 190M triangles/s, depending on how many vertices were shared in the model—note that some vertices had to be duplicated to allow item buffer rendering). Only on the CANYON model did we observe a fill rate limitation (9 vs. 12 hemicubes/s for 512 vs. 1024 resolutions), whereas CITY and PPLANT were geometry limited (7 and 2 hemicubes/s).

Efficient acceleration algorithms exist for both architectures, if a certain amount of preprocessing is tolerated. Of particular importance for visibility processing is that the complexity of scenes that can be handled by ray tracing is limited only by the available storage space, as ray casters can work efficiently out of core (e.g., Wald et al. [2004] have demonstrated that a 350 million polygon model can be ray cast at 2-3 frames per second). Furthermore, it should be pointed out that rasterization benefits from hardware acceleration, whereas ray tracing is still run in software. Recent advances in hardware for ray tracing [Woop et al. 2005] promise a huge potential for improving the speed of sampling-based algorithms like GVS even further, once this technology becomes more commonplace.

However, the main difference between the algorithms is the principal goal. NIR aims to increase rendering speed by aggressively culling more objects than are actually occluded, the rationale being that large gains in rendering speed can be obtained if errors in the final image are tolerated. Indeed, NIR consistently underestimates the PVS, as shown in Table 2 (note that even for an error threshold of 0, a significant rest-error is reported for NIR [Nirenstein and Blake 2004]).

While this approach is valuable for applications like quick previewing etc., where a resolution can be fixed, and an average of, for example, 1000 false pixels is tolerable, many applications require a more accurate PVS. This is where GVS excels. The GVS algorithm aims to provide the most accurate PVS possible with a minimum number of samples. Therefore, the performance metric for GVS is not the total percentage of culled objects, but the degree to which the actual PVS can be approximated. Our results show that GVS, using a limited number of samples, consistently finds the largest PVS, resulting in average pixel errors below 0.005%. This is important for any visualization application that relies on visibility preprocessing (especially if antialiasing is used or the output resolution is not fixed in advance), but also for a number of other applications where reliable (and practically exact) visibility is required, e.g., computational geometry, GI, and robotics.

It should be noted for the results in Table 2 that NIR results are derived through a PVS subdivision threshold, which works differently from the method used in GVS and RAND and can therefore not be compared directly. We found that this threshold was very sensitive to the type of the scene and had to be tuned so as not to lead to excessive subdivision or too early termination in each scene separately (for example, in once case the error for the 1024 resolution was significantly worse than for 512, due to premature termination). The reason for NIR's inability to pick up the complete PVS lies both in the regular sampling strategy, which forces a very fine subdivision on the view cell in order to pick up sub-pixel triangles, and in the thresholding for the adaptive subdivision, which can prematurely terminate the subdivision.

4.5 Comparison to exact visibility

We compared our algorithm to EXACT on the CUBES scene, from a view cell of about 1.5x1.5m. EXACT took 19s on a PIV 1.7GHz PC to find 3,743 visible triangles. To find the same number of triangles, GVS required about 3s. For GVS, a screenspace error of 0.001% was already reported after 2s. More interesting, however, is the fact that both GVS and RAND found significantly more visible triangles than EXACT if given enough samples. For example, 3,850 triangles were found after only 15s by GVS. Note that EXACT was used on an "as is" basis—better results could certainly be achieved by tuning numerical thresholds intrinsic to the method. This shows clearly that the accuracy of visibility algorithms, even exact ones, is ultimately limited by numerical issues.

5 Related Work, Discussion and Applications

A large volume of research has been devoted to visibility problems due to their importance in computer graphics, computer vision, robotics and other fields. This section compares various as-

pects of the proposed visibility sampling algorithm to a wider class of from-region visibility algorithms. For a general overview, we can recommend excellent surveys of visibility problems and algorithms [Durand 1999; Cohen-Or et al. 2003].

From-region visibility algorithms are usually classified as exact (potentially visible set PVS = exact visible set EVS), conservative (PVS $\supseteq$ EVS), aggressive (PVS $\subseteq$ EVS), or approximate (PVS $\sim$ EVS).

5.1 Exact Visibility

Exact solutions to compute visibility from a region in space have been rare [Duguet and Drettakis 2002; Durand 1999], but recently, two algorithms have been published [Nirenstein et al. 2002; Bittner 2003] and further improved upon [Haumont et al. 2005; Mora et al. 2005] that are both exact and work for general scenes. While exact algorithms have been the holy grail of the visibility community for a long time, these two algorithms show that the complexity inherent in the visibility problem may be an obstacle to make exact visibility widely applicable. The high running times and high complexity of implementation are critical, and numerical robustness issues can actually make the solution as approximate as a sampling-based strategy (see [Bittner 2003]). We believe that sampling-based methods and exact methods complement each other, as they have different strengths and weaknesses.

5.2 Conservative Visibility

Several authors stress the importance of conservative visibility computations, i.e., never underestimating the visible set. Since this problem is almost as hard as the exact visibility problem, practically all published conservative from-region algorithms simplify the problem by imposing certain restrictions on the scene. Typical restrictions are the limitation to 2.5D visibility [Wonka et al. 2000; Bittner et al. 2001; Koltun et al. 2001], architectural scenes [Airey et al. 1990; Teller and Séquin 1991], the restriction to volumetric occluders [Schaufler et al. 2000], or the restriction to larger occluders close to the view cell [Leyvand et al. 2003; Durand et al. 2000]—this last restriction is implied by the nature of the data structures used to store visibility information. While it can be argued that larger occluders can be synthesized from smaller ones [Andujar et al. 2000], this is not possible in general. The guarantee to include all visible geometry in the PVS may be important for some applications, but ultimately, sampling-based methods can be much more successful:

1. As opposed to the published conservative algorithms, they do not make any assumptions about the scene, allowing them to handle a much larger variety of scenes.
2. Due to their ease of implementation and robustness, non-conservative algorithms are more practical for commercial products such as computer games [Aila and Miettinen 2004], and are already used in this context.
3. Numerical issues often make conservative algorithms non-conservative in practice.

5.3 Aggressive Visibility

Since visibility is such a fundamental problem, general, robust and practical tools are important to complement the specialized algorithms discussed before. These tools are almost universally based on sampling. The two most popular solutions are to randomly select a large number of rays to sample visibility [Schaufler et al. 2000; Airey et al. 1990; Shade et al. 1998], or to first sample the boundary of the view cell with points and then sample visibility from each of these points [Levoy and Hanrahan 1996; Stuerzlinger 1999]. In the context of view planning for laser range scanners, sampling algorithms exist that store the void surface or the void volume to compute the next-best view [Pito 1999]. A similar algorithm was also used for the generation of textured depth meshes [Wilson and Manocha 2003]. Another option is to shoot rays from the scene triangles towards the view cell [Gotsman et al. 1999], which leads to oversampling of ray space for most scenes.

Nirenstein and Blake [2004] were the first to realize the full potential of sampling for visibility computation. They proposed a new approach which uses graphics hardware for sampling. As discussed in Section 4.4, this algorithm aims to reduce the rendering time by culling even visible triangles as long as this does not result in significant rendering error. This is opposed to our algorithm, which always tries to find the best possible approximation of the exact visible set.

5.4 Algorithm Analysis

Ray space analysis. In the introduction in Figure 3, we have argued that it is desirable not to sample the ray space regularly. The right image in this figure shows that only an approximately 1D subspace of rays needs to be considered in this simple 2D example. Our new algorithm samples ray space more intelligently: random sampling places initial seed points in ray space to stochastically search for regions in ray space that have not been explored yet. To continue the example for 2D as in the figure, adaptive border sampling corresponds to a vertical expansion in 2D ray space (since the viewpoint remains fixed) which only proceeds into yet unexplored areas. A particular advantage of the adaptive border sampling method is that the sampling rate is adapted to the geometric complexity of the visible surfaces. Reverse sampling, on the other hand, is a movement in the horizontal direction (since the hitpoint remains fixed) in cases where these movements promise to lead to not yet explored regions.

For the full 3D case, it is instructive to study our algorithm in terms of the *visibility complex* [Durand 1999]. The visibility complex describes a partition of the 4D ray space into 4D regions of rays that hit the same object (note that ray space is strictly 4D because we are only interested in rays starting from the view cell). The 3D boundaries of this partition are called *tangency volume* and consist of rays tangent to scene objects. Samples placed along the object borders therefore correspond to samples near the tangency volume of the object in dual space. Since we keep the viewpoint (2 degrees of freedom) fixed during the deterministic ABS exploration phase, we need to sample a 1D set only. Without ABS, we would ignore the tangency volumes and have to sample the whole 2D subset of ray space defined by the chosen viewpoint.

Reverse sampling, on the other hand, looks for lines tangent to two scene edges. In ray space, these lines are near intersections of two tangency volumes. These intersections are called bitangents and are only 2D. For reverse sampling, the viewpoint is allowed to move along a plane (1D), so in total RS also samples a 1D set. The combined ABS and RS strategies therefore correspond to explorations of the 4D ray space along those 1D curves that are most likely to reveal new objects. This explains the high efficiency of the GVS algorithm.

Another useful interpretation of the ABS sampling strategy in 3D is based on the visibility map [Bittner 2002]. The visibility map is a structure that contains all visible line segments in a given view.

These segments can be characterized mainly as flat and corner (interior edges of a mesh), or shadow (depth discontinuities). The ABS sampling strategy places samples at all edges of the visibility map (without explicitly constructing it). Samples on interior edges of a mesh serve to find connected sets of a mesh (trivially adjacent regions in the visibility complex). Samples at the shadow edges serve to discover depth discontinuities, where objects are partly occluded by other objects. Shadow edges are where the RS sampling strategy is used to refine the sampling (by finding the bitangents in the visibility complex).

Accuracy. The term conservative (or even exact) visibility is actually quite misleading. Most algorithms, though conservative in theory, are not conservative in practice due to numerical robustness problems. This is especially true for algorithms relying on graphics hardware. Furthermore, complex algorithms are prone to implementation problems. Due to the much improved sampling efficiency, the magnitude of error introduced by our algorithm is comparable to that of other error sources. Such errors are usually tolerated for conservative algorithms (see Section 4). Other algorithms that are often used in conjunction with visibility processing, like level-of-detail algorithms or shadow mapping, are an additional source of errors.

Scene complexity. One distinguishing feature of our sampling-based algorithm is that it can handle arbitrary types of scenes with high overall and visual complexity. It does not rely on occluder synthesis, and depends mostly on the size of the visible set, not on the total scene complexity.

5.5 Limitations and Future Work

Although guided visibility sampling generally finds the major part of the PVS very quickly, the fact that it is stochastic on the one hand and guided by the visibility in the scene on the other hand makes the final accuracy dependent on the structure of the scene. Therefore, we cannot give any hard guarantees for the pixel error of the calculated PVS. Also, the ability to explore connected ray space subsets in the far distance is limited by the numerical precision of the ray direction vector. For ABS, this means that triangles that have a solid angle of less than double precision accuracy when seen from the ray origin will most likely be missed.

The worst case of scene complexity is in scenes that consist of a large set of small disconnected triangles, such as forest scenes or synthetic scenes of random triangles. The visibility of such scenes is so complex that even sampling-based solutions will either have high error or take a long time to compute. Still, it is important to point out that sampling-based algorithms are the only ones that are able to even process these scenes.

In this respect, an avenue of future work is to incorporate geometric LOD into the sampling framework, similar to the vLOD system proposed by Chhugani et al. [2005]. Geometric LODs could potentially increase the speed of the ray tracer, and make intersection computations more robust because small triangles in the distance get replaced by larger ones. However, robust geometric LOD is not available for all scenes, and integrating LODs into ray tracers is a current topic of research. Furthermore, the error metric used to create the LODs impacts the accuracy of the visibility algorithm and therefore the usable output resolutions.

5.6 Applications

One important strength of sampling-based methods is their ease of application. We will discuss a number of application scenarios for our algorithm.

Visibility preprocessing for real-time rendering and games. This is the scenario already described in the overview, and one of the most important applications for GVS. For example, the scenes of current computer games are becoming increasingly general, so that special purpose algorithms (cells and portals, and 2.5D solutions) cannot be used anymore, while exact algorithms are difficult to implement and error-prone. GVS can be used in all stages of game development: During level design, the number of rays can be limited so that a coarse solution can be provided almost instantaneously. For the final production, the PVS can be calculated with high accuracy. It is very important to create a PVS that is as close to the EVS as possible and not dependent on a particular output resolution, since the resolution the application will be run at is not known in advance. In addition, antialiasing methods (supersampling and multisampling) use information from subpixel triangles, so that the virtual resolution is even higher. Note that although scenes in computer games are inherently dynamic, the major part of the scene is still static, so huge gains in rendering speeds can be obtained. Furthermore, GVS works on arbitrary polyhedral view cells, so that the view space can be chosen freely.

Online and networked visibility. As shown in the results, a reasonable approximation to the EVS with low pixel error can be found in a second or less. Therefore, GVS can be used for online visibility culling by running it on a separate processor or over the network, as described in the Instant Visibility system [Wonka et al. 2001]. In this case, transmitting the PVS on a per-object basis will improve results because it suffices for one triangle of an object to be found by GVS in order to classify the whole object as visible. Furthermore, a small modification to GVS makes the algorithm better suitable to progressive evaluation: instead of interleaving ABS and random samples from the beginning, create a certain number (e.g., 1M) of random samples in a startup phase, and only then use those to seed the ABS rays. This will give a better distribution of samples in the initial phase of the algorithm, since ABS systematically "flood fills" the PVS around its seed point, and it takes some time until all image regions have been reached.

Impostor generation. In many scenes, visibility culling is not sufficient to guarantee a high frame rate everywhere in the model. Therefore, image-based methods can be used to replace complex scene parts by so-called impostors. However, since impostors trade rendering speed against memory consumption, it is important to find the exact visible parts of the scene to avoid wasting impostor memory on invisible geometry [Jeschke et al. 2005]. GVS is ideally suited for this purpose since it provides accurate per-triangle visibility information, so that only those object parts that are actually visible need to be stored in an impostor.

Visibility as decision basis. Many practical applications require accurate visibility information as part of a decision making process. Examples include visibility analysis in urban planning (does the new skyscraper impact old town?), military applications (line of sight culling, tactical battlefield management [McDermott and Gelsey 1987]), telecommunications (visibility of emitters), robotics and many more. GVS is advantageous for these problems because it is general purpose and does not have any parameters to tweak, and does not depend on any special properties of the scene.

6 Conclusion

We have presented a visibility sampling algorithm to compute a full 3D visibility solution from a region in space. The proposed algorithm improves the efficiency of previous sampling strategies by over two orders of magnitude, thereby allowing visibility solutions with negligible error to be computed in reasonable time. The proposed algorithm works on arbitrary so-called polygon soups and does not require any memory beyond that used by the ray caster. Due to the new sampling strategies employed in the algorithm, its accuracy is competitive even with exact and conservative approaches, while it is also extremely simple to implement.

We have provided evidence that Guided Visibility Sampling closes an important gap in visibility research. It combines the speed and ease of implementation of sampling-based and special-purpose conservative algorithms with most of the accuracy of exact solutions. Thus, GVS can be used as a general purpose visibility tool.

Acknowledgements

We thank Jiri Bittner for fruitful discussions. This research was also supported by the EU in the scope of the GameTools project (IST-2-004363), and by the NGA, grant no. HM1582-05-1-2004.

References

AILA, T., AND MIETTINEN, V. 2004. dPVS: An occlusion culling system for massive dynamic environments. *IEEE Computer Graphics & Applications 24*, 2.

AIREY, J. M., ROHLF, J. H., AND BROOKS, JR., F. P. 1990. Towards image realism with interactive update rates in complex virtual building environments. In *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, vol. 24, 41–50.

ANDUJAR, C., SAONA, C., AND NAVAZO, I. 2000. Lod visibility culling and occluder synthesis. *Computer Aided Design 32*, 13, 773–783.

BITTNER, J., WONKA, P., AND WIMMER, M. 2001. Visibility preprocessing for urban scenes using line space subdivision. In *Proc. of Pacific Graphics 2001*, 276–284.

BITTNER, J. 2002. Efficient construction of visibility maps using approximate occlusion sweep. In *SCCG '02: Proceedings of the 18th spring conference on Computer graphics*, 167–175.

BITTNER, J. 2003. *Hierarchical Techniques for Visibility Computations*. PhD thesis, Czech Technical University in Prague.

CHHUGANI, J., PURNOMO, B., KRISHNAN, S., COHEN, J., VENKATASUBRAMANIAN, S., AND JOHNSON, D. S. 2005. vLOD: High-fidelity walkthrough of large virtual environments. *IEEE Trans. on Visualization and Computer Graphics 11*, 1, 35–47.

COHEN-OR, D., CHRYSANTHOU, Y. L., SILVA, C. T., AND DURAND, F. 2003. A survey of visibility for walkthrough applications. *IEEE Trans. on Visualization and Computer Graphics 9*, 3, 412–431.

DUGUET, F., AND DRETTAKIS, G. 2002. Robust epsilon visibility. In *Proc. ACM SIGGRAPH 2002*, 567–575.

DURAND, F., DRETTAKIS, G., THOLLOT, J., AND PUECH, C. 2000. Conservative visibility preprocessing using extended projections. In *Proc. ACM SIGGRAPH 2000*, 239–248.

DURAND, F. 1999. *3D Visibility: Analytical Study and Applications*. PhD thesis, Universite Joseph Fourier, Grenoble, France.

GOTSMAN, C., SUDARSKY, O., AND FAYMAN, J. 1999. Optimized occlusion culling using five-dimensional subdivision. *Computers and Graphics 5*, 23, 645–654.

HAUMONT, D., MÄKINEN, O., AND NIRENSTEIN, S. 2005. A low dimensional framework for exact polygon-to-polygon occlusion queries. In *Proc. Eurographics Symposium on Rendering*, 211–222.

JESCHKE, S., WIMMER, M., SCHUMANN, H., AND PURGATHOFER, W. 2005. Automatic impostor placement for guaranteed frame rates and low memory requirements. In *Proc. of ACM SIGGRAPH Symp. on Interactive 3D Graphics and Games*, 103–110.

KOLTUN, V., CHRYSANTHOU, Y., AND COHEN-OR, C.-O. 2001. Hardware-accelerated from-region visibility using a dual ray space. In *Rendering Techniques 2001*, 205–216.

LEVOY, M., AND HANRAHAN, P. 1996. Light field rendering. In *Proc. ACM SIGGRAPH 96*, 31–42.

LEYVAND, T., SORKINE, O., AND COHEN-OR, D. 2003. Ray space factorization for from-region visibility. *ACM Transactions on Graphics 22*, 3, 595–604.

MCDERMOTT, D., AND GELSEY, A. 1987. Terrain analysis for tactical situation assessment. In *Proceedings Spatial Reasoning and Multi-Sensor Fusion*, 420–429.

MORA, F., AVENEAU, L., AND MÉRIAUX, M. 2005. Coherent and exact polygon-to-polygon visibility. In *Proceedings of Winter School on Computer Graphics 2005*, 87–94.

MÜLLER, P., WONKA, P., HÄGLER, S., ULMER, A., AND GOOL, L. V. 2006. Procedural modeling of buildings. *ACM Transactions on Graphics 25*, 3.

NIEDERREITER, H. 1992. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM Philadelphia.

NIRENSTEIN, S., AND BLAKE, E. 2004. Hardware accelerated visibility preprocessing using adaptive sampling. In *Rendering Techniques 2004*, 207–216.

NIRENSTEIN, S., BLAKE, E., AND GAIN, J. 2002. Exact from-region visibility culling. In *Rendering Techniques 2002*, 191–202.

PITO, R. 1999. A solution to the next best view problem for automated surface acquisition. *IEEE Trans. Pattern Anal. Mach. Intell. 21*, 10, 1016–1030.

RESHETOV, A., SOUPIKOV, A., AND HURLEY, J. 2005. Multi-level ray tracing algorithm. *ACM Trans. on Graphics 24*, 3, 1176–1185.

SBERT, M. 1993. An integral geometry method for fast form factor computation. *Computer Graphics Forum 12*, 3, C409–C420.

SCHAUFLER, G., DORSEY, J., DECORET, X., AND SILLION, F. 2000. Conservative volumetric visibility with occluder fusion. In *Proc. ACM SIGGRAPH 2000*, 229–238.

SHADE, J., GORTLER, S., WEI HE, L., AND SZELISKI, R. 1998. Layered depth images. In *Proc. ACM SIGGRAPH 98*, 231–242.

STUERZLINGER, W. 1999. Imaging all visible surfaces. In *Proc. Graphics Interface 1999*, 115–122.

TELLER, S. J., AND SÉQUIN, C. H. 1991. Visibility preprocessing for interactive walkthroughs. *Computer Graphics (Proc. ACM SIGGRAPH 91) 25*, 61–69.

WALD, I., PURCELL, T. J., SCHMITTLER, J., BENTHIN, C., AND SLUSALLEK, P. 2003. Realtime ray tracing and its use for interactive global illumination. In *Eurographics State of the Art Reports*.

WALD, I., DIETRICH, A., AND SLUSALLEK, P. 2004. An interactive out-of-core rendering framework for visualizing massively complex models. In *Rendering Techniques 2004*, 81–92.

WILSON, A., AND MANOCHA, D. 2003. Simplifying complex environments using incremental textured depth meshes. *ACM Transactions on Graphics 22*, 3, 678–688.

WONKA, P., WIMMER, M., AND SCHMALSTIEG, D. 2000. Visibility preprocessing with occluder fusion for urban walkthroughs. In *Rendering Techniques 2000*, 71–82.

WONKA, P., WIMMER, M., AND SILLION, F. 2001. Instant visibility. *Computer Graphics Forum 20*, 3, 411–421.

WOOP, S., SCHMITTLER, J., AND SLUSALLEK, P. 2005. RPU: a programmable ray processing unit for realtime ray tracing. *ACM Transactions on Graphics 24*, 3, 434–444.