A General Algorithm for Output-Sensitive Visibility Preprocessing

Samuli Laine*
Helsinki University of Technology / TML
Hybrid Graphics, Ltd.

Abstract

Occlusion culling based on precomputed visibility information is a standard method for accelerating the rendering in real-time graphics applications. In this paper we present a new general algorithm that performs the visibility precomputation for a group of viewcells in an output-sensitive fashion. This is achieved by exploiting the directional coherence of visibility between adjacent viewcells. The algorithm is independent of the underlying from-region visibility solver and is therefore applicable to exact, conservative and aggressive visibility solvers in both 2D and 3D.

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

Keywords: visibility, occlusion culling, PVS

1 Introduction

Visibility preprocessing is a powerful and commonly used occlusion culling method for accelerating the rendering in real-time graphics applications such as computer games. Preprocessed visibility data can be used for culling the objects that are hidden by stationary occluders with practically zero overhead, thereby making the rendering process highly output-sensitive. However, the precomputation of the visibility data using most current methods is an input-sensitive process, and it may become a significant burden in the production pipeline for large virtual environments.

The precomputed visibility data typically consists of visibility relations between *viewcells* and objects. During rendering, the viewcell that contains the viewpoint is identified, and only the objects that are visible from the viewcell are drawn. In other words, the visible set (VS) of the viewcell is used as a potentially visible set (PVS) for the viewpoint. In virtual environments that exhibit strong occlusion, the visible set of a viewcell contains only a small fraction of the objects on the average. If the amount of occlusion is small, the usefulness of a precomputed visibility solution is diminished as the average size of the visible set approaches the total number of objects.

The purpose of our algorithm is to perform the computation of the visible sets for a group of viewcells in an output-sensitive fashion, meaning that the computational complexity of the process depends only on the output, not on the input. This requires that the number of cell-object visibility tests is proportional to the number of visible cell-object relations in the output; a property that is achieved by exploiting the coherence of visibility between adjacent viewcells. The algorithm is independent of the underlying from-region visibility algorithm and can be used on top of any conservative, aggressive or exact algorithm.

Unlike other methods that exploit the coherence between viewcells, our algorithm is not hierarchical, and it operates directly on the final set of viewcells. Therefore, the visibility queries are made from smaller regions than in hierarchical methods, which benefits many conservative visibility algorithms.

The rest of this paper is organized as follows. The related work is briefly reviewed in Section 2. In Section 3, we introduce the concept of a directed visible set and establish a strong relationship between the directed visible sets of adjacent viewcells. Section 4 shows how this enables us to schedule the visibility computation so that the visibility information can be propagated efficiently. In section 5 we discuss the implementation of the algorithm with an aggressive rasterization-based visibility solver and an exact visibility solver. Discussion and experimental results are given in sections 6 and 7. Finally, conclusions and future work are given in Section 8.

2 Related Work

Visibility is one of the most researched fields in computer graphics, and the amount of published research is overwhelming. Comprehensive surveys are given by Cohen-Or et al. [2003] and Bittner and Wonka [2003].

Computing the visible sets for a set of viewcells requires solving *from-region* visibility between the viewcell and the objects. This is in contrast to a *from-point* visibility solution that only considers the visibility of the objects from the current viewpoint.

Conservative Algorithms A conservative visibility algorithm never judges a visible object to be hidden, but may do the opposite, placing an actually hidden object into the visible set. This results in sub-optimal rendering performance. The quality of a conservative algorithm is largely defined by how much it overestimates the visible set.

Teller and Séquin [1991] present an algorithm for evaluating the visibility between arbitrarily shaped cells that are connected by portals. A natural cell-and-portal decomposition can be found mainly for indoor scenes because it requires that the pathways for visibility, i.e. the portals between cells, are relatively rare and can be explicitly identified.

When the objects in the environment are used as occluders, often a small subset of all objects is sufficient to represent most of the occlusion. Coorg and Teller [1997] precompute a set of good occluders for each viewcell and cull the objects in the scene hierarchically from the current viewpoint. The occluders must be convex, and occluder fusion occurs only when the combined silhouette of multiple occluders is convex. A similar approach is taken by Cohen-Or et al. [1998], who precompute the set of strong occluders as well as the set of visible objects for each viewcell.

*e-mail: samuli@tml.hut.fi

Bittner et al. [1998] precompute a set of potential occluders for each viewcell. At run-time, the occluders are selected dynamically from this set and their aggregate occlusion is computed using an occlusion tree that is built according to the current viewpoint. Koltun et al. [2000] observe that the occluders need not be actual objects in the scene, but may be constructed artificially to represent the aggregate occlusion of multiple objects. These *virtual occluders* are precomputed for each viewcell and used for culling objects at runtime. Bernardini et al. [2000] present a method for constructing simplified view-dependent occluders for complex objects.

Leyvand et al. [2003] present a spectacularly fast hardware-accelerated conservative method for computing a reasonably tight from-region PVS in 2.5D+ε scenes, where horizontal and vertical coherence of occluders is assumed. A method based on voxelization of watertight objects is given by Schaufler et al. [2000]. In practice, the requirement of watertight objects severely restricts the applicability of the method. Durand et al. [2000] give a conservative method for performing visibility precomputation from a viewcell based on extended projections. Their preprocessing algorithm evaluates the visibility to each initial viewcell separately, and subdivides a viewcell recursively if the number of visible objects is too large.

A method for parallelizing rendering and PVS computation from a small region around the current viewpoint is presented by Wonka et al. [2001]. A recent method by Bittner et al. [2004] maintains the PVS of a moving viewpoint using hardware occlusion queries. The algorithm benefits from spatial and temporal coherence of visibility. The performance is reported to be slightly inferior when compared to a precomputed viewcell-based PVS. An advantage of the method is that it is able to handle dynamic occluders.

Aggressive Algorithms An aggressive visibility algorithm produces a visible set that may lack some of the actually visible objects but never contains hidden objects. This results in optimal rendering performance, but gives rise to possible artifacts due to missing objects.

Gotsman et al. [1999] perform statistical visibility sampling by ray casting. Their method constructs a five-dimensional BSP tree that bounds the possible ray origins and directions. Subdividing the BSP nodes corresponds to either subdividing the 3D cell that bounds the origin of the ray or constraining the directional bounds of the ray, depending on the split axis. A recent aggressive sampling method is presented by Nirenstein and Blake [2004], where hardware-accelerated rasterization is used in place of ray casts. The viewcells are constructed during the PVS computation by subdividing a three-dimensional kD-tree whose root node encloses the entire scene.

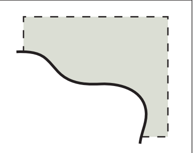
Exact Algorithms An exact visibility algorithm always determines the visibility of an object correctly, never omitting a visible object or placing a hidden object in the visible set. The rendering performance is optimal and no artifacts can occur. Unfortunately, exact visibility algorithms are computationally much more expensive than conservative or aggressive algorithms.

Exact from-region visibility algorithms have been presented for 2D, 2.5D and 3D cases. A thorough treatise on the subject is given by Bittner [2002]. Nirenstein et al. [2002] apply exact from-region visibility in the context of PVS precomputation. Their method utilizes virtual occluders that are built during the processing but does not exploit the coherence of visibility between viewcells.

Exploiting Coherence Of the methods listed above, only four [Gotsman et al. 1999; Nirenstein and Blake 2004; Cohen-Or et al. 1998; Durand et al. 2000] benefit from the coherence of occlusion between viewcells. In these, the visibility information is built in top-down fashion by subdividing a viewcell and computing the visible sets of the children by taking into account only the objects that are visible to the parent. The subdivision process can begin with a single viewcell covering the whole environment [Gotsman et al. 1999; Nirenstein and Blake 2004] or with a grid of initial viewcells [Cohen-Or et al. 1998; Durand et al. 2000].

Bittner et al. [2004] and Wonka et al. [2001] exploit the coherence of occlusion in a different way while computing the PVS according to the current viewpoint. In the algorithm of Bittner et al. [2004], a slowly moving viewpoint leads to most of the visibility data from previous frame remaining valid. In the approach of Wonka et al. [2001] the PVS is computed from a region that surrounds the current viewpoint. This ensures that the PVS remains valid for a predictable amount of time, assuming that the movement speed of the viewpoint is bounded.

3 Directed Visible Sets

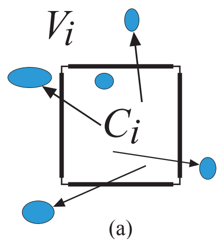

A viewcell.

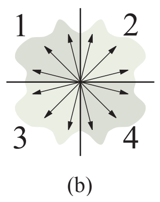
In this section we define the concept of a *directed visible set* and establish a neighborhood relationship between the directed visible sets of adjacent viewcells. This relationship is used in the derivation of the algorithm in Section 4. Throughout the paper, we assume the part of the surface of any viewcell that can be seen through lies on the surface of an axis-aligned box, as illustrated in the inset figure. In the figure, the interior of the viewcell is shown as gray, the thick line represents an opaque surface and the dashed line corresponds to the unobstructed surface of the viewcell. From now on, when speaking of the surface of the viewcell, we refer to this unobstructed part of the surface.

The *visible set* of a viewcell is the set of objects that are visible from some point inside the viewcell. Given a viewcell C_i, the corresponding visible set V_i is equal to the union of the set of objects intersected by C_i and the set of objects that are visible from the surface of C_i. This is because every *sightline* that originates from the interior of C_i must either terminate at an object inside C_i or pass through the surface of C_i. We can therefore reduce the problem of determining V_i into determining the set of objects intersected by C_i and determining the visible set of the surface of C_i.

Let us consider the division of the *direction space* of the sightlines into separate partitions. In a *dim*-dimensional space, we divide the direction space of the sightlines into 2^{dim} partitions so that in each partition the signs of the components of the direction vector are fixed. This gives four quadrants in 2D and eight octants in 3D. The approach is similar to the one of Gotsman et al. [1999], except that we always perform the direction space division exactly into 2^{dim} directional partitions, and no further subdivision is done. We note that the directional partitions are non-overlapping and that their union equals the entire direction space.

The directed visible set V_{i_d} of cell C_i for a single directional partition, denoted d, is now defined as the set of objects that are visible from some point inside C_i when only the sightlines whose direction vectors satisfy the sign constraints of d are allowed. As with unconstrained direction vectors, this equals the union of the set of objects intersected by C_i and the set of objects that are visible from the surface of C_i, but now with directionally constrained sightlines.





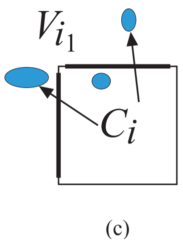


Figure 1: Direction space division and directed visible sets. (a) Visible set V_i of cell C_i consists of the marked elliptical objects, the arrows representing possible sightlines. (b) Directional partitioning in 2D yields four quadrants that are here numbered from 1 to 4. (c) Directed visible set V_{i_1} of C_i corresponding to quadrant 1. Note that the directionally constrained sightlines can penetrate only two faces of C_i.

Furthermore, only the parts of the surface of C_i that face towards the directions specified by d can contribute. Figure 1 illustrates the direction space division and a directed visible set in comparison to a traditional visible set.

Finally, we observe that the union of directed visible sets V_{i_d} gives the visible set V_i of cell C_i, since the union of the directional partitions equals the entire direction space. Consequently, we can solve the directed visible sets separately and combine them to arrive at the final visible set V_i.

3.1 Neighborhood Relationship

When a sightline exits viewcell C_i through any of its faces, it simultaneously enters a neighboring viewcell if there is one. Thus, if C_i is surrounded by neighbors from all sides, it follows that all sightlines must traverse through the neighboring cells. Now, assume that a sightline S originates from the interior of cell C_i, penetrates the surface of C_i at point P, and finally terminates at object O. Object O is now visible from any point along sightline S. Consequently, object O must belong into the visible set of all the cells intersected by S, and particularly into the visible set of the immediate neighbor of C_i at P.

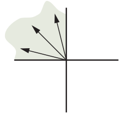
From this, we obtain a neighborhood relationship for the visible sets. Assuming that cell C_i is completely surrounded by neighboring viewcells, we denote the set of neighbors of C_i by N_i, and the set of objects intersected by C_i by O_i. It now follows that

$$V_i \subseteq O_i \cup \bigcup_{j \in N_i} V_j. \tag{1}$$

This can be extended to directed visible sets as well. We observe that for each directional partition d it is possible to prune the neighborhood set N_i because only a subset of neighbors of C_i can be entered by directionally constrained sightlines originating from the interior of C_i. Denoting the *directed neighborhood set* of cell C_i by N_{i_d}, we obtain a similar relationship.

$$V_{i_d} \subseteq O_i \cup \bigcup_{j \in N_{i_d}} V_{j_d}. \tag{2}$$

Thus, when computing the directed visible set V_{i_d} of cell C_i, we need to consider only the objects in the set formed by the right hand side of Equation 2. This set gives us a conservative estimate of V_{i_d}, and consequently also a conservative estimate of the set of occluders that may affect V_{i_d}.



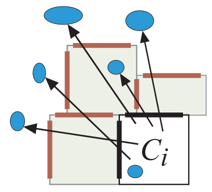


Figure 2: Directed neighborhood relationship. When the directions of the sightlines are constrained to the quadrant shown in (a), the directed visible set of C_i in (b) is a subset of the directed visible sets of the three neighbors shown plus the objects intersected by C_i. Furthermore, the directed visible set of a single face of C_i is bounded by the directed visible sets of the neighbors of C_i behind that particular face.

In practice, the relationship can be made even tighter by considering the directed visible sets of the separate faces of C_i, namely by observing that the directed visible set of a face of C_i is a subset of the union of the directed visible sets of the neighbors behind that particular face. The directed neighborhood relationship is illustrated in Figure 2.

4 Optimal Scheduling of Computation

In this section we show how the direction space partitioning allows us to schedule the computation of the visible sets so that the sets V_{j_d} on the right hand side of Equation 2 are always available when determining the set V_{i_d} on the left hand side.

4.1 Directed Dependence Graph

From Equation 2 we see that obtaining a conservative estimate for V_{i_d} requires that $V_{j_d},\ j \in N_{i_d}$ are already known. This gives us a set of dependence relations, from which we construct a dependence graph for directional partition d, as illustrated in Figure 3. In the dependence graph we have a node for each cell, and an edge from cell C_i to cell C_j if and only if $j \in N_{i_d}$, meaning that C_i depends on C_j. Determining the edges of the dependence graph is trivial, since N_{i_d} contains the neighboring cells of C_i that are behind the faces of C_i defined by the directional partition d.

The dependence graph for a directional partition d contains no cycles if the set of viewcells has been constructed by a recursive axis-aligned splitting process, after which some of the cells may have been removed. This can be seen by considering the modifications made to the dependence graph caused by splitting. When a node is split, the newly formed dependence edges connected to the children retain their direction as illustrated in Figure 4, and no cycles external to the node can emerge. Furthermore, since the split is axis-aligned, both children cannot depend on each other. This ensures that the resulting set of viewcells cannot have cyclic dependence graphs.

It is worth noticing that it is possible to construct viewcell configurations that generate cyclic dependence graphs, as illustrated in Figure 5. These kind of configurations are not producible by a recursive splitting process, and apparently occur only in 3D. In these cases, it is always possible to split the nodes that contribute to the cycles until all cycles are broken. However, from now on we concentrate only on viewcell configurations without dependence cycles, such as those produced by recursive splitting.

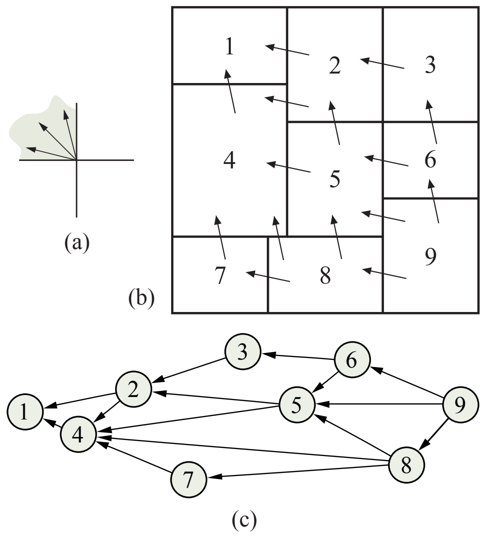


Figure 3: A set of viewcells and the corresponding dependence graph. (a) The directional partition for which the dependencies are computed. (b) A set of viewcells with dependencies drawn as arrows. (c) The corresponding dependence graph ordered so that all dependencies go from right to left. Processing the cells in left-to-right order ensures that the directed visible sets V_{j_d} on the right hand side of Equation 2 are known when computing the corresponding V_{i_d} on the left hand side.

After the dependence graph has been constructed, the computation can be optimally scheduled by sorting the graph so that all dependence edges go from future to past. Such an ordering is illustrated in Figure 3c. The sorting can be done in linear time with respect to the number of nodes and edges in the graph. Furthermore, it is not necessary to sort the graph prior to the actual computation, since the graph can easily be traversed in a sorted order.

4.2 The Multipass Algorithm

We are now equipped with the tools needed to derive the algorithm for computing the visible sets of all viewcells. The complete visibility solution algorithm loops through all 2^{dim} directional partitions, and for each partition, constructs the directed dependence graph and traverses it in a sorted order. In practice, maintaining a dependence counter for each cell is sufficient for traversing the dependence graph in the correct order.

Our *multipass algorithm* for visibility precomputation is given in Algorithm 1. We now examine the algorithm in some detail. At line 3, the dependence counter for each cell is initialized according to the number of neighbors it has to the direction of current d. If there are no neighbors, the cell is added to the set S of cells that are ready to be processed. During the execution of a single directional pass, set S contains the cells C_i whose dependence counter dep_i is zero. The loop at lines 6–13 runs until all cells have been processed. At line 7, a cell C_i with $dep_i = 0$ is selected from set S. Since $dep_i = 0$, all neighbors of C_i to the direction of d, i.e., all cells that cell C_i depends upon have already been processed. Therefore, in the call to COMPUTE-DIRECTED-VS at line 8, all sets V_{j_d} on the right hand side of Equation 2 have been solved. At lines 9–12 the dependence counters of the cells that depend on cell C_i are decremented, and the cells whose dependence counter reaches zero are added to S. Finally, after all directional passes are made, the directed visible sets are combined to yield the final visible sets at line 15.

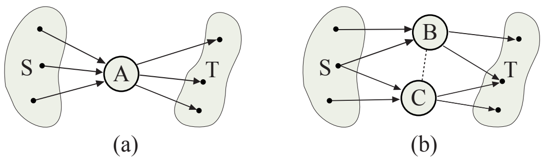


Figure 4: A set of viewcells produced by a recursive splitting process cannot have cyclic dependence graphs. To see this, we consider splitting the nodes while updating the dependence graph. (a) Cell A has a set of cells S that depend on it, and a set of cells T it depends upon. If the graph is initially acyclic, sets S and T must be disjoint. (b) Node A is split into a pair of nodes B and C, which inherit some or all dependence relations of A. Since B and C are spatial subsets of A, a dependence edge may either remain or disappear for a child, but cannot change its direction. Regardless of whether B depends on C or vice versa, no cycles can emerge.

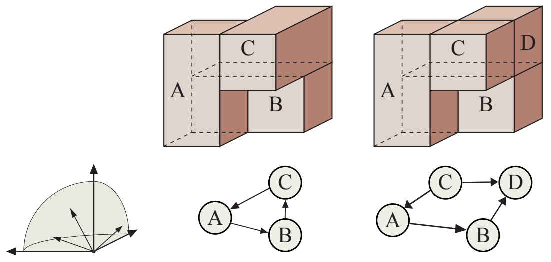


Figure 5: A particularly wicked configuration of viewcells in 3D may generate a cyclic dependence graph. Left: a configuration that generates a cycle and the corresponding dependence graph. Right: splitting cell C results in an acyclic dependence graph. The directional partition for which the graphs are drawn is shown in the lower left.

5 Implementing the Algorithm

In this section we consider some possible implementations of subroutine COMPUTE-DIRECTED-VS that is called at line 8 of Algorithm 1. The purpose of this subroutine is to compute the directed visible set V_{i_d} for cell C_i, given the directional partition d and assuming that the directed visible sets V_{j_d}, $j \in N_{i_d}$ are available. We focus entirely on the 3D case ($dim = 3$), but the methods discussed are equally applicable to 2D.

The visibility solver in our context refers to the algorithm that determines the set of objects that is visible from the surface of viewcell C_i with directionally constrained sightlines. In the following, we consider how the directional constraints for the sightlines can be enforced in the exact visibility solvers presented by Bittner [2002] and Nirenstein and Blake [2004], as well as in the aggressive rasterization-based solver of Nirenstein et al. [2004].

It should be noted that it is not mandatory to enforce the directional constraints in the visibility solver in order to use the multipass algorithm, since the visibility solver may be conservative. Even in this case, the multipass algorithm remains output-sensitive. The only drawback is that the directed visible sets on the right hand side of Equation 2 would be overly conservative and thus the computation would be slower.

```
COMPUTE-VS($C, dim$)
1  for each $d \in [1, 2^{dim}]$ do
2      for each $C_i$ do
3          $dep_i \leftarrow$ number of neighbors of $C_i$ in direction of $d$
4          if $dep_i = 0$ then add $C_i$ into $S$
5      end for
6      while $S$ is nonempty do
7          pick any $C_i$ from $S$ and remove it from $S$
8          $V_{i_d} \leftarrow$ COMPUTE-DIRECTED-VS($C_i, d, dim$)
9          for each neighbor $C_j$ of $C_i$ in direction reverse to $d$ do
10             $dep_j \leftarrow dep_j - 1$
11             if $dep_j = 0$ then add $C_j$ into $S$
12         end for
13     end while
14 end for
15 for each $C_i$ do $V_i \leftarrow \bigcup_{d \in [1, 2^{dim}]} V_{i_d}$
```

Algorithm 1: The multipass visible set computation algorithm in pseudocode. Detailed explanation of the algorithm is given in Section 4.2.

5.1 Exact Visibility Solver

The exact visibility solver algorithm of Bittner [2002] is based on a six-dimensional dual representation of the line space in 3D. The same approach is taken by Nirenstein et al. [2002]. The six-dimensional dual space is called the *Plücker space*. A thorough treatise on the subject is beyond the scope of this paper, and we refer the interested reader to the comprehensive presentation by Bittner [2002].

The six *Plücker coordinates* of a directed line l are denoted $\pi_0 \dots \pi_5$. The Plücker coordinates of a line from point u to point v are given by the following formulas (from [Bittner 2002]).

$$\begin{array}{lll} \pi_0 = v_x - u_x & \pi_1 = v_y - u_y & \pi_2 = v_z - u_z \\ \pi_3 = u_y v_z - u_z v_y & \pi_4 = u_z v_x - u_x v_z & \pi_5 = u_x v_y - u_y v_x \end{array} \quad (3)$$

A simple permutation of the Plücker coordinates of a line defines a six-dimensional *Plücker hyperplane* with components $\omega_0 \dots \omega_5$. Taking the sign of the dot product between a dual-space representation of line l_1 and the Plücker hyperplane of line l_2 tells whether line l_1 passes line l_2 in a clockwise or a counter-clockwise manner. Now, the set of lines that pass through a planar polygon corresponds to a convex six-dimensional polytope that is the intersection of the half-spaces defined by the Plücker hyperplanes of the edges of the polygon. The exact visibility algorithms of Bittner [2002] and Nirenstein et al. [2002] are based on constructing the six-dimensional polytope that represents the possible lines between two polygons, and recursively removing the parts that correspond to lines passing through occluder polygons. If the polytope vanishes[1], the polygons cannot see each other.

Our directional constraints for the sightlines fix the signs of the components of direction vector of the sightline. These constraints are easily converted into Plücker hyperplanes. From Equation 3 we see that coordinates $\pi_0 \dots \pi_2$ are equal to the components of a three-dimensional vector from point u to v. Now, a constraint on the sign of x component of the direction vector of the sightline is realized in six-dimensional dual space by a hyperplane corresponding to a line with nonzero π_0 and zero $\pi_1 \dots \pi_5$. Similarly, for y and z components, two more hyperplanes are needed. Therefore, enforcing the directional constraints for the sightlines only requires adding three additional Plücker hyperplanes into the initial six-dimensional polytope that defines the lines between the two polygons between which the visibility is being solved.

[1]Actually, it is only required that the remaining parts of the polytope do not intersect a second-order surface called the *Grassman manifold* in the Plücker space.

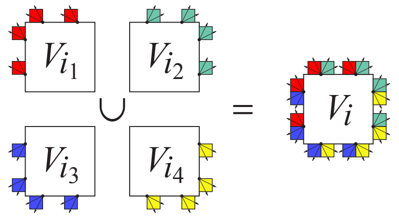


Figure 6: Two-dimensional illustration of hemicubical sampling. The directional constraints of the sightlines are enforced by rasterizing only one fourth of a hemicube (one half of the hemisquare in 2D). Combining the results from multiple directional partitions yields the same result as rendering full hemicubes, with exactly the same number of rasterized fragments in total. The numbering of the directional partitions is the same as in Figure 1.

5.2 Sampling Visibility Solver

Next we consider the point sampling-based visibility solver presented by Nirenstein and Blake [2004]. In this method, the visible set of a viewcell is determined by sampling the visibility from multiple points located on the surface of the viewcell. The sampling is performed by rasterizing hemicubical maps centered at the sampling points.

With this algorithm, the directional constraints for the sightlines can be enforced by limiting the view frusta of the images that are rasterized from a sampling point. Instead of rasterizing a complete hemicube it suffices to rasterize only a fourth of the hemicube that corresponds to the octant specified by the current directional partition, as illustrated in Figure 6. When eight directional passes are made, the number of rasterized fragments is equal to sampling the same number of whole hemicubes once. The cost of additional geometry processing can be practically removed by per-object view-frustum culling. With the point-based sampling method, we can utilize the directed visible sets of the neighboring cells very efficiently by rasterizing only the objects in the directed visible set of a single neighbor cell; instead of taking the union in Equation 2, we can directly use the V_{j_d} of the neighbor at the sampling point.

6 Discussion

In this section we contrast our multipass algorithm for exploiting the coherence of visibility with the previously presented methods. In addition, we discuss the limitations of the algorithm and give a proof of the output-sensitivity of the algorithm.

6.1 Comparison to Related Methods

Of the methods surveyed in Section 2, four [Gotsman et al. 1999; Nirenstein and Blake 2004; Cohen-Or et al. 1998; Durand et al. 2000] exploit the coherence of visibility between viewcells by solving the sets of visible objects hierarchically. This has several drawbacks. If the visibility algorithm performs little or no occluder fusion, the occlusion power of single occluders becomes very small for large viewcells. In addition, the first splits in the hierarchy are generally expensive, since the visible set to be refined is large.

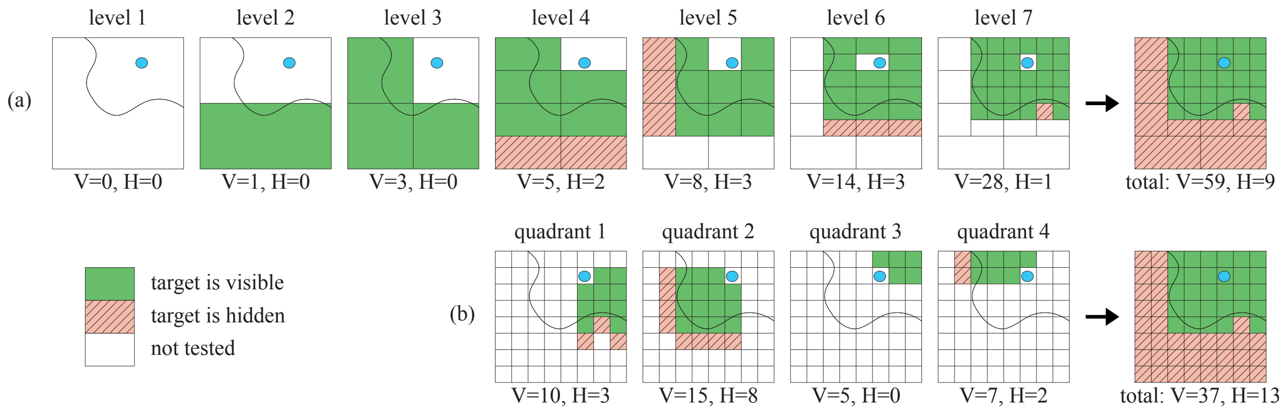


Figure 7: A comparison of a visibility computation algorithm that processes viewcells hierarchically [Cohen-Or et al. 1998; Durand et al. 2000] and our multipass algorithm. Exact visibility queries are assumed. The ellipse represents a target for a visibility query, and may be an object or a node in a bounding volume hierarchy for objects. The curved line is an occluder. For each hierarchy level or directional pass, the number of queries returning visible (V) and hidden (H) as a result is shown. (a) The hierarchical algorithm does not evaluate the visibility of the target for the children of a node in the viewcell hierarchy when the target is hidden from the parent node. However, many unnecessary visible queries are made. (b) Our multipass algorithm operates directly on the finest-level viewcells. The number of visible queries is smaller than in the hierarchical algorithm. More hidden queries are made, but they are made from significantly smaller regions than in the previous cases, which would benefit conservative visibility solvers.

Figure 7 compares the number of visible and hidden occlusion queries made by a hierarchical algorithm and our multipass algorithm. The hierarchical algorithm is used by Cohen-Or et al. [1998] and Durand et al. [2000]. Note that here we refer to using a hierarchical technique for the viewcells, as opposed to placing the occludees into a hierarchy. Indeed, the only feature we focus on is how the algorithms exploit the coherence of visibility between viewcells. The target for the visibility query may be a single object or a node in an object hierarchy.

In the simple 2D scene of Figure 7, our algorithm performs fewer visible queries and somewhat more hidden queries. However, it must be noted that the hidden queries are made from much smaller regions than in the hierarchical method. If a conservative visibility solver was used in the example, the hierarchical methods could have behaved worse, since many conservative visibility solvers overestimate the visibility more for large regions than small regions.

In most hierarchical algorithms the progressive top-down refinement of the visibility data requires multiple evaluations of the same visibility relationships. For example, if an object is visible from every viewcell, its visibility is separately proven for every node in the viewcell hierarchy. The superset simplification method of Nirenstein and Blake [2004] avoids this multiple evaluation by caching the visibility information of distinct sampling points so that they can be re-used when a viewcell is split. The first splits, however, remain relatively expensive, and especially if the entire scene cannot be accommodated in main memory, the sampling of the first levels of the hierarchy may become prohibitively expensive. With high sample counts, the cache may become very large even when compressed, which may also limit the applicability of the cache-based method. In addition, the method can only be used with point-based sampling.

Our multipass algorithm is not hierarchical and thus needs no superset simplification algorithm. Furthermore, if used in conjunction with a point-based sampling method, it always provides the sampler a tighter visible set to refine than the superset simplification algorithm of Nirenstein and Blake [2004]. The difference in the visible set to be refined is illustrated in Figure 8.

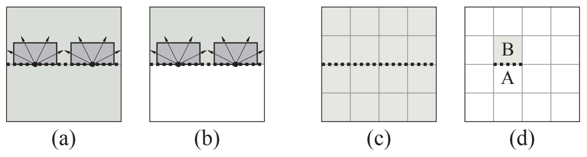


Figure 8: Comparison to the cache-based superset simplification method of Nirenstein and Blake [2004]. (a) When a viewcell is split, the superset simplification method of Nirenstein and Blake [2004] performs visibility sampling for the splitting plane using the visible set of the whole viewcell (shown in gray). (b) In the same situation, our algorithm needs to consider only the visible set of the half of the cell when sampling from the plane. The difference is even larger if the viewcell is eventually split further. (c) Assuming that a viewcell is eventually split into 4×4 viewcells, the superset simplification method of Nirenstein and Blake [2004] considers the visible set of the whole original cell for the first split (dotted line). (d) With our algorithm, the visibility sampling for the shown portion of the dotted line needs to consider only the visible set of cell B when determining the visible set of cell A. No caching of the results of distinct sampling points is required.

6.2 Limitations

Our algorithm requires that the final set of viewcells is determined prior to the visibility computation. This is in contrast to the hierarchical methods that are able to terminate the splitting of the viewcells adaptively based on the evaluated visible sets. However, it is possible to first determine the set of viewcells using e.g. the sampling-based method of Nirenstein and Blake [2004] with very coarse sampling, and then perform the final visibility computation with our algorithm.

If the viewcells do not span the entire environment, there may be sightlines that pass through the boundary of the entire *viewcell compound*, as illustrated in Figure 9. This may occur with e.g. a hole in a wall if view cells are placed on one side only. When solving the directed visible sets from the boundary of the viewcell compound,

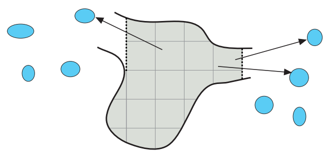


Figure 9: In this simplified 2D environment there are sightlines between the viewcells (gray squares) and objects (ellipses) that do not lie completely within the set of viewcells. Solving the directed visible sets from the boundary of the viewcell compound (dotted lines) requires that all objects are taken into account, since there is no neighbor information to benefit from.

no neighbor information is available, and therefore all objects in the environment must be taken into account. The problem becomes severe only if the boundary of the viewcell compound is relatively large. In this case, it may become beneficial to evaluate the visibility from the boundary with a hierarchical algorithm, while using the multipass algorithm for the viewcell faces inside the compound.

6.3 Output-sensitivity

We now show that the multipass algorithm is output-sensitive under certain conditions. First, we assume that all sightlines are contained inside the viewcell compound. Second, the proof relies on the assumption that the number of cells that may directly depend on any viewcell is bounded by some constant K. In a regular grid, for example, $K = dim$. Finally, we require that the running time of a visibility query between a single viewcell and a set of objects depends only on the number of objects in the query, the same objects acting both as occluders and occludees.

To prove that the multipass algorithm is output-sensitive, we may focus on a single directional pass, since if processing a single directional partition is output-sensitive, so is processing a constant number of partitions. From now on, we consider only a single pass and define the size of the output as the number of visible cell-object relations within the directional partition d of the pass.

Consider a situation where the directed visible set V_{i_d} has been determined for viewcell C_i. No assumptions about the conservativeness or exactness of the visibility solver used for determining this set are needed. Now assume that V_{i_d} contains n objects, which causes an increase of n to the size of the output. To prove the output-sensitivity of the algorithm, we now show that the increase in the computational workload for the remaining cells is at most proportional to n.

Let us consider a cell C_j that is one of cells that depend directly on C_i. When we are to solve the directed visible set of cell C_j, we must take into account the objects that are in the directed visible sets of the cells it depends on. Let us denote the union of these sets S. Now, if the directed visible set V_{i_d} of C_i turned out to contain n objects, this can cause an increase of at most n in the size of S. Observing that there are at most K cells that depend on C_i, we see that the total increase in the sizes of all sets S that are affected by C_i is at most Kn. Therefore the total increase in workload caused by having n visible objects in C_i is at most proportional to n. This completes the proof.

7 Experimental Results

We compared our method against a simple brute-force algorithm and the cache-based viewcell hierarchy algorithm of Nirenstein and Blake [2004] in a set of visibility preprocessing tasks. For solving the visibility, rasterization was used with uniform sampling density and no sample positioning heuristics. Since none of the algorithms solves the visibility from same area multiple times, all algorithms took exactly the same number of samples in a given test case.

The most important figures in the results are the average sizes of the *refine sets* that the algorithms give to the visibility solver as the conservative approximations of the visible sets. Before from-point rasterization, the refine sets are pruned by object-level view frustum culling, and the refine set sizes in the results are measured after view frustum culling done per sampling point. In order to get comparable results, the hemicubes of the two comparison algorithms were rasterized in multiple parts as is done by the multipass algorithm. The additional cost of rasterizing the hemicubes in parts is limited to performing geometry transformations multiple times for objects on the boundary of the view frustum.

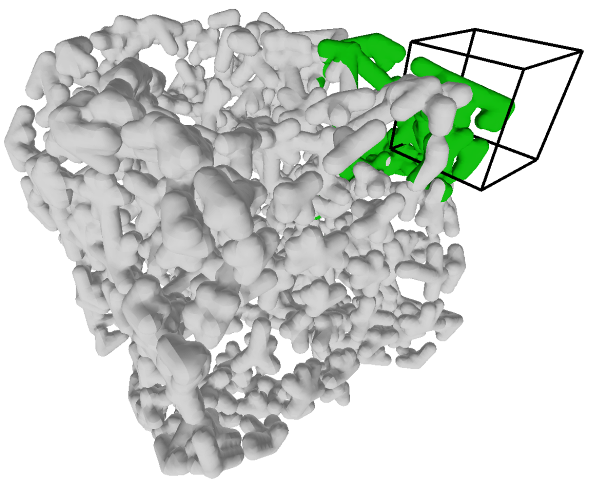
Evaluating the output-sensitivity of the algorithms requires that the amount of visibility can be adjusted while keeping all other parameters as constant as possible. To accomplish this, we used four procedurally generated test scenes, two of which are shown in Figure 10. The scenes are split into objects by simple spatial subdivision. All scenes have $\sim$350K triangles and $\sim$1800 objects. The visibility precomputation was performed for three viewcell resolutions, 2^3, 4^3 and 8^3 viewcells. In total, twelve test runs were thus made for each of the three algorithms.

The tests were run on Pentium Mobile 1.6GHz with ATI Mobility Radeon 9200. The results are shown in Table 1. It can be seen that the multipass algorithm performs better than the comparison algorithms both in terms of refine set sizes and rasterization times. Figure 11 shows the improvement in the sizes of the refine sets compared against the cache-based viewcell hierarchy algorithm of Nirenstein and Blake [2004]. It is seen that the relative efficiency of the multipass algorithm increases when the PVS size is reduced either due to added occlusion or increased viewcell resolution.

8 Conclusions and Future Work

We have presented a new general algorithm for precomputing static visibility into a set of viewcells in an output-sensitive fashion. In contrast to the previous algorithms that exploit the coherence of visibility, our algorithm is not hierarchical but operates directly on the finest-grain viewcells. This is beneficial when using conservative from-region visibility solvers that perform limited occluder fusion and consequently tend to overestimate the visibility from large regions. We have shown that using our algorithm requires only small changes into existing exact from-region visibility solvers to enforce the directional constraints for the sightlines.

A chief advantage of our algorithm, when compared to hierarchical algorithms, is that the full set of objects in the environment does not need to be considered at any point of visibility precomputation, provided that the viewcells span the entire environment. This is especially important for very large environments that cannot be accommodated in main memory. In addition, multiple evaluation of the visibility relationships is completely avoided.



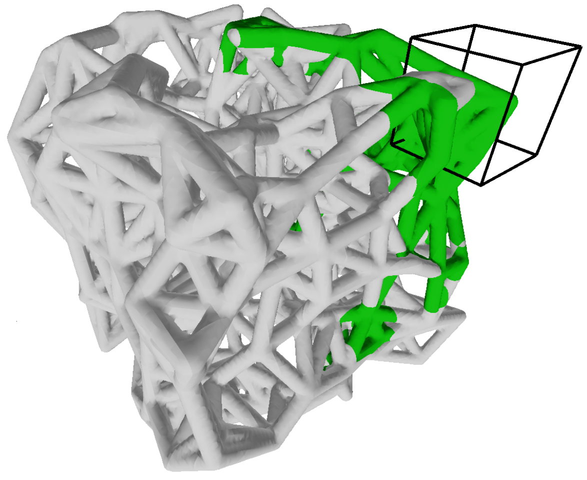


Figure 10: Four procedurally generated test scenes were used for performance measurements. The tunnel network is the same for all scenes, but the number of open tunnels is varied in order to control the amount of visibility. The model on the left (TUNNELS-10) has 10% of the tunnels opened, and consequently the visible set (dark green) of the viewcell shown is small. The model on the right (TUNNELS-100) has 100% of the tunnels open, which makes the visible set larger. In addition to these models, two intermediate variants with 40% and 70% of open tunnels were used (TUNNELS-40 and TUNNELS-70, respectively). All models have ∼350K triangles and ∼1800 objects.

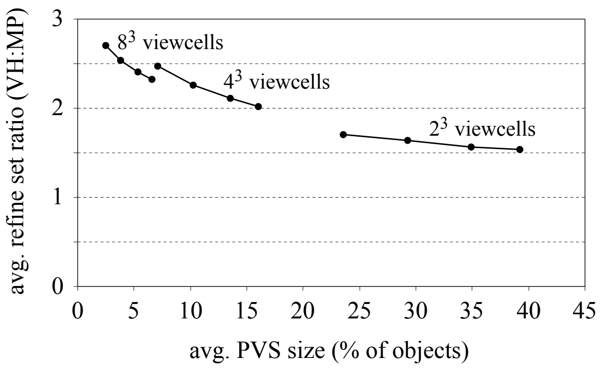


Figure 11: Ratio between the average sizes of the refine sets produced by the cache-based viewcell hierarchy algorithm of Nirenstein and Blake [2004] (VH) and the multipass algorithm (MP). On the horizontal axis is the average size of the PVS. Each four-point series corresponds to the results for the four test scenes with the same viewcell resolution. In each series, the relative efficiency of the multipass algorithm increases as the amount of visibility diminishes due to added occlusion. The relative efficiency of the multipass algorithm increases also when the size of the PVS is reduced by increasing the viewcell resolution.

8.1 Future Work

Efficient processing of the boundary of the viewcell compound may require hierarchical or other techniques if the boundary is large. An algorithm that combines the multipass algorithm with other techniques for dealing with the boundary remains to be developed.

The multipass algorithm requires that the set of viewcells is known a priori. However, it might be possible to develop a hybrid algorithm that constructs the viewcells adaptively based on the visibility, while retaining the output-sensitivity of the multipass algorithm.

Acknowledgements

The author would like to thank Timo Aila and Lauri Savioja for their helpful comments. This work has been partially funded by the National Technology Agency of Finland, Bitboys, Hybrid Graphics, Nokia and Remedy Entertainment.

References

BERNARDINI, F., EL-SANA, J., AND KLOSOWSKI, J. T. 2000. Directional discretized occluders for accelerated occlusion culling. *Computer Graphics Forum (Eurographics '00) 19*, 3, 507–516.

BITTNER, J., AND WONKA, P. 2003. Visibility in computer graphics. *Environment and Planning B: Planning and Design 30*, 5, 729–756.

BITTNER, J., HAVRAN, V., AND SLAVIK, P. 1998. Hierarchical visibility culling with occlusion trees. In *Proceedings of Computer Graphics International '98*, 207–219.

BITTNER, J., WIMMER, M., PIRINGER, H., AND PURGATHOFER, W. 2004. Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum (Eurographics '04) 23*, 3, 615–624.

BITTNER, J. 2002. *Hierarchical Techniques for Visibility Computations*. PhD thesis, Czech Technical University in Prague, Department of Computer Science and Engineering.

COHEN-OR, D., FIBICH, G., HALPERIN, D., AND ZADICARIO, E. 1998. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum (Eurographics '98) 17*, 3, 243–254.

COHEN-OR, D., CHRYSANTHOU, Y., SILVA, C. T., AND DURAND, F. 2003. A survey of visibility for walkthrough ap-

	scene	viewcells = $2 \times 2 \times 2$			viewcells = $4 \times 4 \times 4$			viewcells = $8 \times 8 \times 8$		
		BF	**VH**	**MP**	**BF**	**VH**	**MP**	**BF**	**VH**	**MP**
avg. refine set size (% of objects)	TUNNELS-10	6.12	4.32	2.53	6.01	2.24	0.91	6.02	1.27	0.47
	TUNNELS-40	6.14	4.55	2.78	6.04	2.51	1.11	6.04	1.50	0.59
	TUNNELS-70	6.22	4.74	3.04	6.08	2.76	1.31	6.10	1.70	0.71
	TUNNELS-100	6.31	4.94	3.21	6.09	2.94	1.46	6.15	1.85	0.80
total rasterization time (seconds)	TUNNELS-10	146.2	73.6	40.7	467.2	123.4	50.6	1143.9	175.3	72.9
	TUNNELS-40	165.9	87.8	50.0	540.7	159.4	66.9	1327.3	231.8	92.8
	TUNNELS-70	192.0	106.8	61.6	616.0	200.9	84.8	1536.3	303.3	117.4
	TUNNELS-100	208.3	119.4	69.3	675.3	234.7	100.5	1731.9	366.0	140.1
avg. PVS size (% of objects)	TUNNELS-10	23.60			7.11			2.48		
	TUNNELS-40	29.23			10.28			3.78		
	TUNNELS-70	34.93			13.56			5.33		
	TUNNELS-100	39.25			16.03			6.58		

Table 1: Experimental results obtained by running three algorithms in the four test scenes with three different viewcell resolutions. The sizes of the refine sets are computed after view frustum culling. Frame buffer readbacks are not included in the rasterization time — this would merely incur a constant cost because every algorithm takes exactly the same number of samples. The brute-force algorithm (BF) processes every viewcell separately, and consequently the refine sets are approximately equally sized regardless of the amount of visibility. The cache-based viewcell hierarchy algorithm of Nirenstein and Blake [2004] (VH) performs significantly better, especially with higher viewcell resolutions. Our multipass algorithm (MP) has yet smaller refine sets in every test case. The reduction in the sizes of the refine sets is reflected in the total rasterization times. For reference, the PVS sizes are also shown for all test scenes and viewcell resolutions.

plications. *IEEE Transactions on Visualization and Computer Graphics 9*, 3, 412–431.

COORG, S., AND TELLER, S. 1997. Real-time occlusion culling for models with large occluders. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, ACM Press, 83–90.

DURAND, F., DRETTAKIS, G., THOLLOT, J., AND PUECH, C. 2000. Conservative visibility preprocessing using extended projections. *Proceedings of ACM SIGGRAPH 2000*.

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

KOLTUN, V., CHRYSANTHOU, Y., AND COHEN-OR, D. 2000. Virtual occluders: An efficient intermediate PVS representation. In *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, 59–70.

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

NIRENSTEIN, S., AND BLAKE, E. 2004. Hardware accelerated aggressive visibility preprocessing using adaptive sampling. In *Rendering Techniques 2004: Proceedings of the 15th Eurographics Symposium on Rendering*, 207–216.

NIRENSTEIN, S., BLAKE, E., AND GAIN, J. 2002. Exact from-region visibility culling. In *Rendering Techniques 2002: Proceedings of the 13th Eurographics Workshop on Rendering*, 199–210.

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

TELLER, S. J., AND SÉQUIN, C. H. 1991. Visibility preprocessing for interactive walkthroughs. *Computer Graphics (Proceedings of ACM SIGGRAPH '91) 25*, 4, 61–70.

WONKA, P., WIMMER, M., AND SILLION, F. 2001. Instant visibility. *Computer Graphics Forum (Eurographics '01) 20*, 3.