A Survey of Visibility for Walkthrough Applications

Daniel Cohen-Or* Tel Aviv University

Yiorgos Chrysanthou† University Of Cyprus

Cláudio T. Silva‡ AT&T Labs

Frédo Durand§ MIT - LCS

Abstract

Visibility algorithms for walkthrough and related applications have grown into a significant area, spurred by the growth in the complexity of models and the need for highly interactive ways of navigating them.

In this survey we review the fundamental issues in visibility and conduct an overview of the visibility culling techniques developed in the last decade. The taxonomy we use distinguishes between point-based and from-region methods. Point-based methods are further subdivided into object- and image-precision techniques, while from-region approaches can take advantage of the cell-and-portal structure of architectural environments, or handle generic scenes.

1 Introduction

Visibility determination has been a fundamental problem in computer graphics since the very beginning of the field [5, 83]. A variety of hidden surface removal (HSR) algorithms were developed in the 1970s to address the fundamental problem of determining the visible portions of the scene primitives in the image. The basic problem is now believed to be mostly solved, and for interactive applications, the HSR algorithm of choice is usually the Z-buffer [15].

Visibility algorithms have recently regained attention because the ever increasing size of 3D datasets makes them impossible to display in real time with classical approaches. Pioneering work addressing this issue includes Jones [52], Clark [19] and Meagher [63]. Recently, Airey et al. [2], Teller and Séquin [84,87], and Greene et al. [46] built upon these seminal papers and revisited visibility to speed up image generation. In this survey, we review recent visibility algorithms for the acceleration of walkthrough applications.

Visibility culling aims at quickly rejecting invisible geometry before actual hidden-surface removal is performed. This can be accomplished by only drawing the visible set, that is, the subset of primitives which may contribute to at least one pixel of the screen. Much of visibility-culling research focuses on algorithms

*School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel, dcor@tau.ac.il

†Department of Computer Science, University Of Cyprus, 75 Kallipoleos Street, P.O. Box. 20537, CY-1678 Nicosia, Cyprus, yiorgos@ucy.ac.cy

‡AT&T Labs-Research, 180 Park Ave., Room D265, Florham Park, NJ 07932, csilva@research.att.com

§Massachusetts Institute of Technology, Laboratory for Computer Science, NE43-255, Cambridge, MA 02139, fredo@gfx.lcs.mit.edu

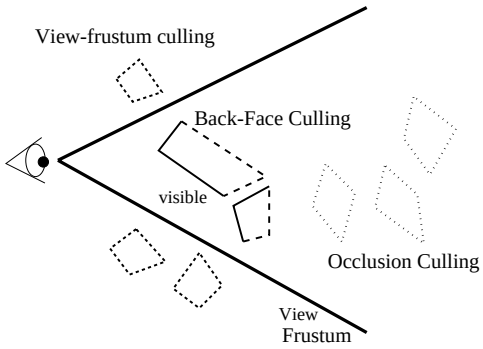


Figure 1: Three types of visibility culling techniques: (i) view-frustum culling, (ii) back-face culling and (iii) occlusion culling.

for computing (hopefully tight) estimations of the visible set. A Z-buffer algorithm is then usually used to obtain correct images.

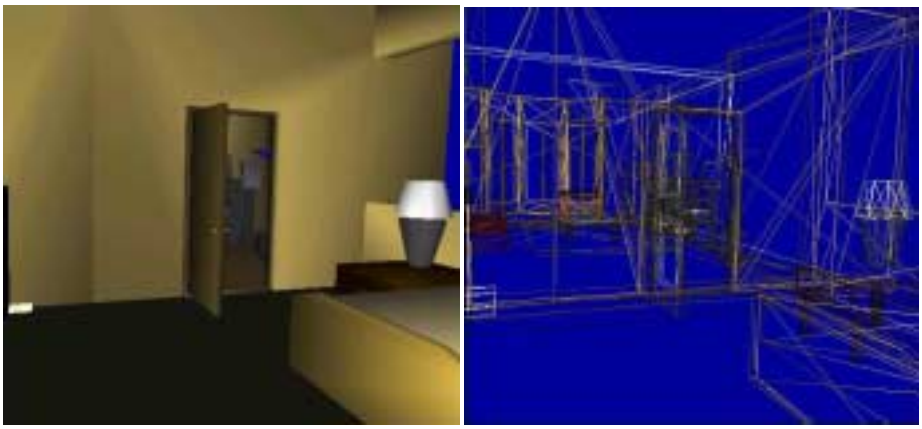
Visibility culling starts with two classical strategies: back-face and view-frustum culling [35]. Back-face culling algorithms avoid rendering geometry that faces away from the viewer, while viewing-frustum culling algorithms avoid rendering geometry that is outside the viewing frustum. Efficient hierarchical techniques have been developed [19, 39, 59], as well as other optimizations [6, 80]

In addition, *occlusion culling* techniques aim at avoiding rendering primitives that are occluded by some other part of the scene. This technique is global as it involves interrelationship among polygons and is thus far more complex than backface and view-frustum culling. The three kinds of culling can be seen in Figure 1. In this survey, we will focus on occlusion culling and its recent developments.

Since testing each individual polygon for occlusion is too slow, almost all the algorithms that we will describe here, place a hierarchy on the scene, with the lowest level usually being the bounding boxes of individual objects, and perform the occlusion test top-down on that hierarchy, as described by Garlick et al. [39] for view-frustum culling.

A very important concept is *conservative visibility* [1, 87]. The *conservative visibility set* is the set that includes at least all of the visible set plus maybe some additional invisible objects. In it, we may classify an occluded object as visible, but may never classify a visible object as occluded. Such estimation needs to be as close to the visible set as possible, and still be "easy" to compute. By constructing conservative estimates of the visible set, we can define a *potentially visible set* (PVS) [1,87] which includes all the visible objects, plus a (hopefully small) number of occluded objects, for that whole region of space. The PVS can be defined with respect to a single viewpoint or with respect to a region of space.

It is important to point out the differences between occlusion culling and HSR. Unlike occlusion culling methods, HSR algorithms invest computational efforts in identifying the exact portions of a visible polygon. Occlusion culling algorithms need to merely identify which polygons are *not* visible, without the need to deal with the expensive sub-polygon level. Moreover, occlusion culling can (hopefully conservatively)

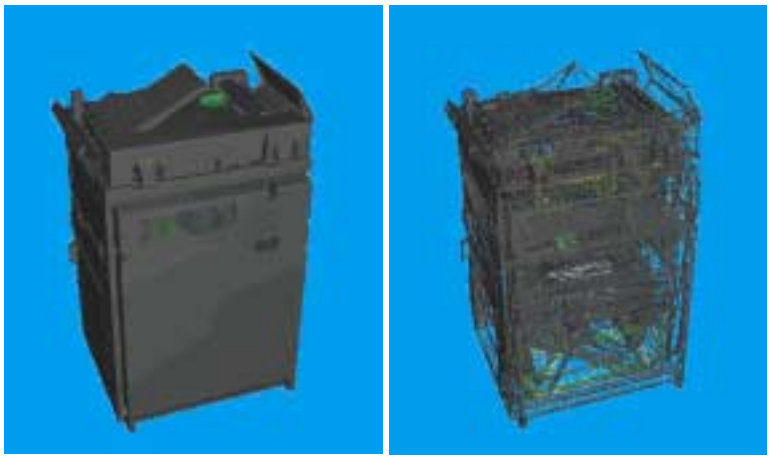


(a) (b)

Figure 2: With indoor scenes often only a very small part of the geometry is visible from any given viewpoint. In (b) the hidden part of the scene is drawn. Courtesy of Craig Gotsman, Technion.

over-estimate the set of visible objects, since classical HSR will eventually discard the invisible primitives. However, the distinction is not a clear cut since in some cases, an HSR algorithm may include an occlusion culling process as an integral part of the algorithm (for example, ray casting), and since some methods integrate occlusion-culling and HSR at the same level.

What makes visibility an interesting problem is that for large scenes, the number of visible fragments is



(a) (b)

Figure 3: A copying machine; only a fraction of the geometry is visible from the outside. But behind the closed shell might be hiding a great deal of geometry (b). Courtesy of Craig Gotsman, Technion.

(a) (b)

Figure 4: A small change in the viewing position can cause large changes in the visibility.

often much smaller than the total size of the input. For example, in a typical urban scene, one can see only a very small portion of the entire model, assuming the viewpoint is situated at or near the ground. Such scenes are said to be *densely occluded*, in the sense that from any given viewpoint, only a small fraction of the scene is visible [2,84]. Other examples include indoor scenes, where the walls of a room occlude most of the scene, and in fact, from any viewpoint inside the room, one may only see the details of that room or those visible through the *portals*, see Figure 2. A different example is a copying machine, shown in Figure 3, where from the outside one can only see its external parts. Although intuitive, this information is not available as part of the model representation, and only a non-trivial algorithm can determine it automatically. (Note that one of its doors might be open.)

The goal of visibility culling is to bring the cost of rendering a large scene down to the complexity of the visible portion of the scene, and mostly independent of the overall size [19,44]. Ideally visibility techniques should be *output sensitive*; the running time should be proportional to the size of the visible set. In open environments, the visible set can be quite large, but several complex scenes are part of "densely occluded environments" [87], where the visible set is only a small subset of the overall environment complexity.

Visibility is not an easy problem, since a small change in the viewpoint might cause large changes in the visibility. An example of this can be seen in Figure 4. The *aspect graph*, described in Section 3, and the *visibility complex* (described in [30]) shed light on the complex characteristics of visibility.

1.1 Related work and surveys

Visibility for walkthroughs is related to many other interesting visibility problems. In particular, in shadow algorithms, the parts of the model that are not visible from the light source correspond to the shadow. So occlusion culling and shadow algorithms have a lot in common and in many ways are conceptually similar, e.g. [18, 96]. It is interesting to note that conservative occlusion culling strategies have not been as widely used in shadow algorithms.

Other related problems include the art gallery problem [68] and its applications to image-based rendering

[34, 81], and global lighting simulation, e.g. [40, 48].

Some other visibility surveys which overlap in content with ours already exist. Durand [30] has written a more comprehensive visibility survey. For those interested in the computational geometry literature, see [27–29]. None of these works focuses on 3D real-time rendering.

Zhang's thesis [97] contains a short survey of computer graphics visibility work, and Moller and Haines [64, Chapter 7] cover several aspects of visibility culling, but with less detail than the previous work, and also without covering much of the recent work.

2 Classification

2.1 A taxonomy of occlusion culling techniques

The organization of the present survey is based on the following taxonomy:

- *Point vs. region.*

  Our major distinction is whether the algorithm performs computations with respect to the location of the current viewpoint only, or performs bulk computations that are valid anywhere in a given region of space. One strength of the from-region visibility set is that it is valid for a number of frames, and thus its cost is amortized over time. More importantly, from-region visibility also has predicting capabilities, which is crucial for network applications or for disk-to-memory pre-fetching. Using the visibility information from adjacent cells, the geometry can be prefetched as it is about to be visible. However, from-region algorithms usually require a long preprocessing, significant storage cost, and do not handle moving objects as well as point-based methods.

- *Image precision vs. object precision.*

  For point-based methods, we will use the classical distinction between object and image-precision [83]. Object precision methods use the raw objects in their visibility computations. Image precision methods on the other hand operate on the discrete representation of the objects when broken into fragments during the rasterization process. The distinction between object and image precision is however not defined as clearly for occlusion culling as for hidden surface removal [83], since most of these methods are anyway conservative, which means that the precision is never exact.

- *Cell-and-portal vs. generic scenes.*

  The last criterion is particularly relevant to from-region visibility. Some algorithms exploit the characteristics of architectural interiors and other similar datasets. These scenes are naturally organized into cells (rooms) that are linked by portals (doors, windows). A visibility method can then be carried out by observing that other cells are visible only through portals. The cell-and-portal approach can also be seen as the dual of the occluder approach: cell-and-portal techniques start with an empty PVS and add objects visible through series of portals, while other approaches first assume that all objects are visible and then eliminate objects found hidden.

2.2 Additional criteria

In addition to these three taxonomy axes, there are various important criteria to take into account when reviewing a visibility algorithm: The various methods are summarized with respect to some of these criteria in Figure 19.

- *Conservative vs. approximate.*

  Most techniques described in this paper are conservative, that is, they overestimate the visible set. Only a few approximate the visible set, but are not guaranteed of finding all the visible polygons. Two classes of approximation techniques can be distinguished: sampling and aggressive strategies. The former use either random or structured sampling (ray-casting or sample views) to estimate the visible set, and hope that they will not miss visible objects. They trade conservativeness for speed and simplicity of implementation. On the other hand, aggressive strategies are based on methods that can be conservative, but choose to lose that property in order to have a tighter approximation of the visible set. They choose to declare some objects as invisible although there is a slight chance that they are not, often based on their expected contribution on the image.

- *Tightness of approximation.*

  Since most algorithms are conservative, it would be interesting to study the degree of over-estimation. Unfortunately, few of the papers we review discuss the ratio between the size of their potentially visible set and the size of the visible set (the authors of these lines being no exception). We hope that the current survey will encourage subsequent articles to provide this measure.

- *All vs. subset of occluders.*

  Some methods treat the occlusions caused by all objects in the scene, while others require the selection of a subset of typically big objects as occluders. The size of this subset can also vary.

- *Convex vs. generic occluders*

  Convexity of the occluders can be required by some methods (typically for object-precision methods).

- *Individual vs. fused occluders.*

  Given three primitives, A, B, and C, it might happen that neither A nor B occlude C, but together they do occlude C. Some occlusion-culling algorithms are able to perform such an *occluder-fusion*, while others are only able to exploit single primitive occlusion. Cell and portal methods are a special case, since they consider the dual of occluders, openings, implicitly fusing together multiple walls. Occluder fusion used to be mostly restricted to image-precision point-based methods. However, from-region visibility methods performing occluder fusion have recently been developed.

- *2D vs. 3D.*

  Some methods are restricted to 2D floorplans or to 2.5D (height fields), while others handle 3D scenes.

- *Special hardware requirements.*

  Several of the techniques described can take further advantage of hardware assistance besides the final Z-buffer pass, either for its precomputation or during the actual rendering.

- *Need of precomputation.*

  Most from-region methods precompute and store the visibility information, but some point-based techniques also require a preprocessing of the data (e.g. for occluder selection).

- *Treatment of dynamic scenes.*

  A few of the algorithms in the literature are able to handle dynamic scenes. One of the main difficulties is handling changes to object hierarchies that most visibility algorithms use [82]. Also if the visibility algorithm uses preprocessing (as discussed above), this information has to be updated. Since from-region methods usually pre-compute a PVS, it is very hard for them to treat dynamic scenes. In particular, all the information has to be recomputed if occluders move. Moving occludees can however be handled by bounding their motion using motion volumes [31].

2.3 Organization of the survey

The organization of this survey is based on the above taxonomy, and on chronological reasons. Before reviewing occlusion-culling techniques, we first present the *aspect graph* data-structure, which will provide a reference on exact visibility computations and analytical visibility properties.

We then introduce from-region methods exploiting the cell-and-portal structure of architectural scenes in Section 4. Section 5 describes point-based occlusion culling working at object-precision, while Section 6 is dedicated to point-based methods using image-precision. Section 7 reviews the more recent class of from-region visibility techniques working with arbitrary scenes.

3 Exact visibility and the aspect graph

When dealing with visibility, it is useful to consider an important data structure developed in computer vision and called the *aspect graph* [32]. As we will see, the aspect graph encodes analytically all the information necessary for efficient display. The principle is to subdivide the viewing space into cells where the view is qualitatively invariant. These cells are separated by *visual events* that are the locus of changes in visibility. Thus, for each such cell, the visible set is constant and changes only when the viewpoint crosses a visual event.

Let us define the aspect graph more formally and look at the two isomorphic graphs in Figure 5. They are projections of a 3D object; however, we treat them as 2D entities. The *Image Structure Graph* (ISG) is the labeled planar graph defined by the view of a polyhedral object. Then two different views of an object have the same *aspect* if and only if their corresponding ISGs are isomorphic. Now we can partition the view space into maximal connected regions in which the viewpoints have the same aspect. This partition is the

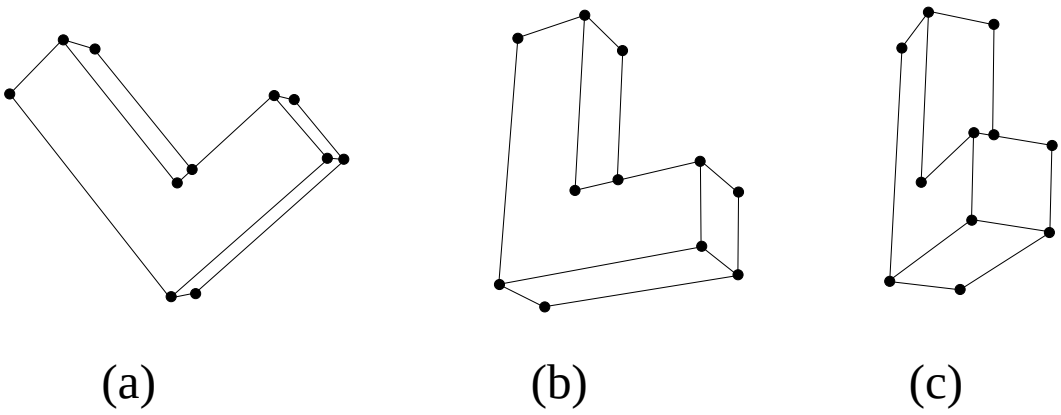


Figure 5: Two different view directions of an object have the same *aspect* if and only if the corresponding Image Structure Graphs are isomorphic. Note that (a) and (b) have the same aspect, which is different to (c).

VSP - *the visibility space partition*, where the boundary of a VSP region is called a *visual event* as it marks a qualitative change in visibility (see Figure 6).

The term, aspect graph, refers to the graph created by assigning a vertex to each region of the VSP, where the edges connecting adjacent regions correspond to visual events.

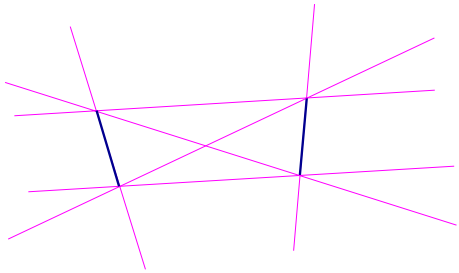


Figure 6: 2 polygons - 12 aspect regions.

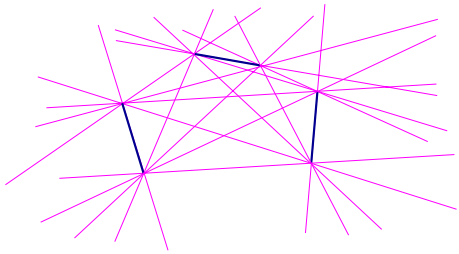


Figure 7: 3 polygons - “many” aspect regions.

Figures 6 and 7 show a visibility space partition in 2D, which is created by just two and three segments (the 2D counterparts of polygons), respectively. One can observe that the number of aspect regions is already

large, and in fact, can be shown to grow quite rapidly with the number of segments.

The worst complexity of aspect graphs is quite high, and in three dimensions, can be as large as $O(n^9)$ (because triples of scene edges can give rise to visual events; and triples of visual events can cause vertices in the visibility space partition). For a typical number of segments (say tens of thousands), in terms of space and time it turns out that computing the aspect graph is computationally impractical. Plantinga [71] proposes an early conservative visibility algorithm based on his aspect graph work. Unfortunately, this algorithm has, to our knowledge, not been implemented.

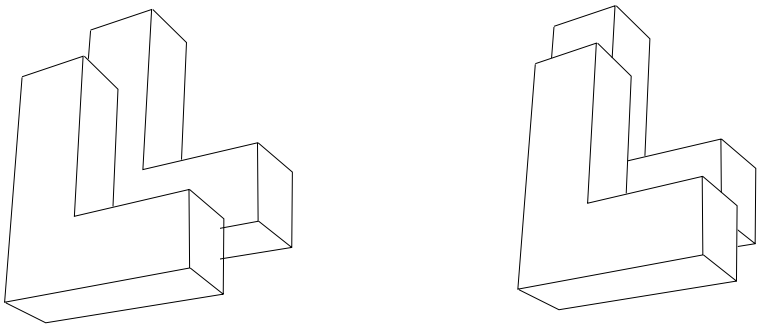


Figure 8: Different aspect regions can have equal sets of visible polygons.

However, as can be seen in Figure 8, different aspect regions can have equal sets of visible polygons. This means that there are far fewer different regions of different visibility sets than different aspects.

Looking once again at the aspect partition of the two segments in Figure 9, we can treat one as an occluder and the other as the occludee, defining their endpoint connecting lines as *supporting lines* and *separating lines*. These lines partition the space into three regions: (i) the region from which no portion of the occludee is visible, (ii) the region from which only some portion of the occludee is visible, and (iii) the region from which the occluder does not occlude any part of the occludee [24].

In 3D, one has to consider supporting and separating planes generated by vertex/edge pairs (EV visual events). Visual events in 3D scenes can also be generated by the interaction of three edges (EEE), which corresponds to non-planar quadric surfaces.

The 3D visibility complex [30] is another way of describing and studying the visibility of 3D space by a dual space of 3D lines, in which all the visibility events and their adjacencies are described. It is more compact than the aspect graph, but its size is still $O(n^4)$.

4 Cell-and-portal from-region visibility

We now review from-region visibility for architectural environments. The work of Airey et al. [1, 2] and Teller and Séquin [87] started the trend towards advanced occlusion culling techniques, and developed much of the foundation for recent work. Some of the key concepts introduced were the notion of "potentially visible sets" from a region of space, "conservative visibility", and "densely occluded environments".

All these works are based on the characteristics of architectural scenes. Namely, that they are naturally

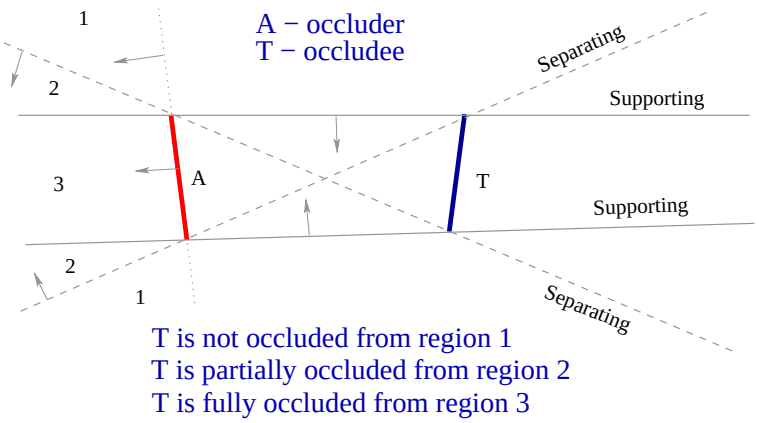


Figure 9: Supporting and separating planes.

subdivided into cells, and that visibility occurs through openings, which they call portals. The potentially visible set is computed for each cell, and used during run-time walkthrough.

The work of Airey et al. [1, 2] proposes two different techniques for computing the PVS. A conservative technique which for each portal, computes whether it can see a given polygon in the model. Their formulation of the algorithm leads to an $O(n^3)$ algorithm. Also in Airely's Ph.D. thesis, he describes several variations of a "sampling" approach, which roughly uses "ray shooting" queries to determine visible geometry in some user-specified precision, and does not actually guarantee conservativeness in the computation of the set of potentially visible primitives.

The work of Teller is quite comprehensive, and covers different aspects of 2D and 3D visibility computations [84, 86, 87]. First, we briefly describe his work on 2D visibility computations. During preprocessing, the model is first subdivided into convex cells using a BSP tree. The main opaque surfaces, such as the walls, are used for defining the partitions and thus the boundaries of the cells. Smaller detailed scene elements are considered 'non-occluding' and are ignored in this step. Non-opaque portals, such as doors, are identified on cell boundaries, and used to form an adjacency graph connecting the cells of the subdivision. See the example in Figure 10. The thick black lines are the walls which are used for partitioning into cells and the light grey lines are the portals. On the left, the adjacency graph shows which cells are directly connected through the portals.

The cell-to-cell visibility is determined by testing if sightlines exist that connect some point in one cell to some point in another. Actually, it is clear that if a line exists from one cell to another, it has to go through a portal and thus we only need to determine if the portals are visible between them. For each cell, the adjacency graph is utilized to generate portal sequences which are then 'stabbed' with the sightline. For example, the tree on the right of Figure 10 shows the cells that are visible from cell A. The cells that are reached by the sightlines contain the potentially visible set (PVS) for any given cell and is stored with it.

During an interactive walkthrough the cell-to-cell visibility can be further dynamically culled using the view volume of the observer, producing a superset of the visible scene data, the eye-to-cell visibility [38]. A cell is visible if all of the following are true: it is in the view volume, all cells along the stab tree are

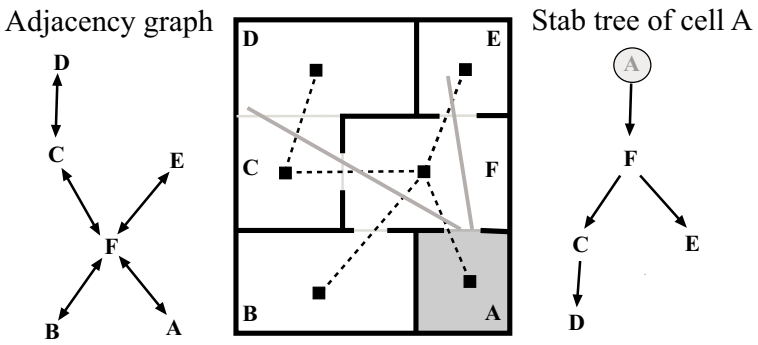


Figure 10: Cells and portals: the adjacency graph and stab tree

in the view volume, all portals along the stab tree are in the view volume, and a sightline within the view volume exists through the portals. Although we might decide to apply only some of these tests, the geometry contained in each visible cell is nevertheless passed down the graphics pipeline for rendering.

In [84, 86], Teller describes techniques for extending his original 2D framework to 3D environments, using a parameterization of line space. Doing so exactly, requires substantial mathematical sophistication beyond what is necessary for the 2D case, and beyond the scope of this survey to describe. We refer the interested reader to his seminal Ph.D. thesis [84].

However, Teller also proposes computing a conservative approximation of the visible region [84, 85] through arbitrary portals in 3D. As each portal is added to the sequence, the separating planes bounding the visibility region are updated. These separating planes between the portals correspond to visual events. For each edge of the sequence of portals, only the extremal separating plane is considered. It is a conservative approximation because some complex non-planar visual events are not considered.

More recently, Jimenez et al. [51] proposed an alternative method to compute conservative visibility through a set of portals. Like Teller's exact technique [84, 86], they perform computations in line space.

5 Point-based object-precision methods

In this section, we review several algorithms which primarily perform visibility computations in object precision. They rely on the identification of big occluders or cells and portals. They have the advantage that all computations can be performed by the CPU, alleviating the need for communication with the graphics hardware that image-precision methods (surveyed in the next section) usually exhibit. However, large occluders or portals are not always easy to find, and these methods are usually less effective at performing occluder fusion.

5.1 Cells and portals

Luebke and Georges [62] propose a point-based cell-and-portal technique, based on an earlier idea of Jones [52] and on the from-region methods discussed in the previous section [1, 84]. Instead of precomputing

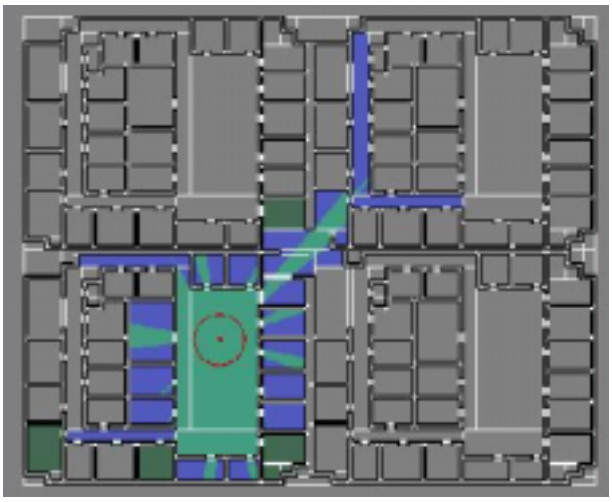


Figure 11: Results from [87] showing the potentially visible set from a given cell. Courtesy of Seth Teller, UC, Berkeley.

for each cell a set of potentially visible geometry, Luebke and Georges perform an on-the-fly recursive depth-first traversal of the cells using screen-space projections of the portals which overestimate the portal sequences, and perform conservative occlusion culling.

Their algorithm works as follows. First, the cell which contains the viewer is rendered, and its portals which are inside the view frustum identified. Clearly, any remaining visible geometry has to lie inside the projection of those portals. The algorithm overestimates the portals by using the axial 2D bounding box of the projected vertices of each portal. Then, the same procedure is repeated for the cells adjacent to the portals. At each step, the new portals are clipped against the pre-existing portals, leading to smaller and smaller visible "windows", until no visible portal remains.

This technique is simple and quite effective, and the source code (an SGI Performer library) is available for download from Luebke's web page. [1]

Hong et al. [49] use an image-based portal technique similar to the work of Luebke and Georges [62] to be able to fly through a virtual human colon in real-time. The colon is partitioned into cells at preprocessing and these are used to accelerate the occlusion with the help of a Z-buffer at run-time.

5.2 Large convex occluders

The work of Coorg and Teller [24, 25] computes the occlusion caused by a subset of large convex occluders. Occluder fusion is not really handled, but they take advantage of temporal coherence by tracking a subset of visual events, which allows them to check the visibility status of occludees only when it actually changes.

They characterize the occlusion of a convex occludee by a single convex occluder using the separating and supporting planes between them (see Figure 8), and the relative position of the viewer with respect to

[1] Pfportals can be obtained at http://pfPortals.cs.virginia.edu.

those planes. The basic idea is that if an observer is between the supporting planes, and behind one of the objects, then it is not possible to see the other object; such is the case if an observer is in region 3 in Figure 8.

In [24], a technique is proposed which incrementally computes a (small) set of visual events where visibility change happens, effectively tracking visibility events among objects as the user moves and the visual relationships among objects change. In effect, the algorithm is implicitly constructing a linearized portion of the aspect graph as the user moves. In principle, a very large number of visual events would need to be considered, but in their paper Coorg and Teller show how to drastically limit the number of visual relationships by only considering a subset of the silhouette edges and vertices of the relevant primitives. The efficiency of the algorithm is further improved by using object hierarchies (based on octrees) to handle the potential quadratic complexity computational increase. The dynamic tracking of the visual events is costly and (seems to be) hard to implement.

In [25], Coorg and Teller propose an improved algorithm. Instead of keeping a large number of continuous visibility events, as the user moves, their algorithm uses a view-dependent set of occluders (which are determined in a preprocessing step – see below), which are used to cull the rest of the scene.

For efficiency, the scene is inserted into an object hierarchy, and the occluders are used to determine which portions of the hierarchy can be pruned, and not rendered. The actual visibility computation is performed hierarchically, and the visibility tests are performed between the pre-selected convex occluders and the bounding boxes of the hierarchy. Their occlusion test is efficient, and uses a conservative estimate of the separating and supporting planes by only considering edges of the occluder and vertices of the occludee (i.e. bounding box).

Their algorithm uses a limited form of occlusion fusion in which it is able to fuse connected occluders whose silhouette with respect to the viewpoint is convex. In any case, their technique is most suitable for use in the presence of large occluders in the scene. In their preprocessing, they select objects with a large projected area in image space to be occluders. To do this, they propose a simple metric based on approximating the solid angle subtended by an object:

$$\frac{-A(\vec{N}\cdot\vec{V})}{||\vec{D}||^2},$$

where A is the area of the occluder, $\vec{N}$ the normal, $\vec{V}$ the viewing direction, and $\vec{D}$ the vector from the viewpoint to the center of the occluder.

5.3 Culling using shadow frusta

A similar way to look at occlusion relationships, is to use the fact that a viewer cannot see the occludee if it is inside the shadow generated by the occluder. Hudson et al. [50] propose an approach based on dynamically choosing a set of occluders, and computing their shadow frusta, which is used for culling the bounding boxes of a hierarchy of objects.

The way Hudson et al. determine which parts of the hierarchy are occluded is different to that of Coorg and Teller. For each of the n best occluders that fall within the view frustum, the authors build a shadow frustum using the viewpoint as the apex and passing through the occluder silhouette. The scene hierarchy

is tested top-down against each of these shadow frusta. If a node of the hierarchy is found to be totally enclosed by one of the frusta then it is occluded and hence discarded (for this frame). If it is found not to intersect any of them then it totally visible and all the objects below it are rendered. If however it partially overlaps even one of them then its children need to be further tested. Interference detection techniques are used for speeding up the tests.

In practice, Hudson et al. precompute and store information for occluder selection. Besides the Coorg and Teller solid-angle heuristic, they also propose taking into account the depth complexity and coherence of the occluders. They use a spatial partition of the scene, and for each cell, identifying the occluders that will be used anytime the viewpoint is inside that cell, and store them for later use.

5.4 BSP tree culling

The method described in Hudson et al. [50] can be improved using BSP trees. Bittner et al. [13] combine the shadow frusta of the occluders into an *occlusion tree*. This is done in a very similar way to the Shadow Volume BSP tree (SVBSP) of Chin and Feiner [16]. The tree starts as a single *lit* (visible) leaf and occluders are inserted, in turn, into it. If an occluder reaches a *lit* leaf then it augments the tree with its shadow frustum; if it reaches a *shadowed* (invisible) leaf then it is just ignored since it means it already lies in an occluded region. Once the tree is built, the scene hierarchy can be compared with it. The cube representing the top of the scene hierarchy is inserted into the tree. If it is found to be fully visible or fully occluded then we stop and act appropriately, otherwise its children are compared with the occlusion tree recursively. This method has an advantage over Hudson et al. [50] since instead of comparing the scene with each of the N shadow frusta, it is compared with one tree with an $O(\log N)$ expected depth (potentially $O(N)$), while taking in into account occluder fusion.

The above technique is conservative; an alternative *exact* method was proposed much earlier by Naylor [67]. That involved a merging of the occlusion tree with the BSP tree representing the scene geometry.

6 Point-based image-precision techniques

As the name suggests image-precision algorithms perform the culling at the discrete representation of the image. The key feature in these algorithms is that during rendering of the scene the image gets filled up and subsequent objects can be culled away quickly by the already-filled parts of the images. Since they operate on a discrete array of finite resolution they also tend to be simpler to implement and more robust than the object-precision ones, which can sometimes suffer numerical precision problems.

Approximate solutions can also be produced by some of the image-precision algorithms by classifying them as occluded geometry parts which are visible through an insignificant pixel count. This invariably results in an increase in running speed.

When the scenes are composed of many small primitives without well-defined large occluders then performing the culling in image-precision becomes more attractive. The projections of many small and individually insignificant occluders can be accumulated on the image using standard graphics rasterizing hardware, to cover a significant portion of the image which can then be used for culling. Another advantage

of these methods is that the occluders do not have to be polyhedral; any object that can be rasterized can be used.

6.1 Ray casting

One of the simplest forms of an image synthesis algorithm is known as ray casting. Here the image is generated by determining which object of the scene is visible from each pixel. By casting a ray that emanates from the eye and passes through a pixel toward the scene, the closest object it intersects determines the content of the pixel. One of the nice properties of this simple concept is that it never renders an occluded part of the scene. The downside of using ray casting as an image synthesis technique is its high complexity. A naive implementation requires each ray to apply an intersection calculation with each of the objects in the scene. However, when acceleration methods are used the rendering is in a back-to-front order that performs a natural occlusion. Thus, it can be extremely fast [7, 8, 21, 22, 70]. In fact, Wald, Slusallek, and colleagues [89–91] recently developed an extremely fast ray tracer which is able to render very large scenes, such as the 12.5 million triangle Power Plant model from the University of North Carolina interactively by using only a handful (actually, seven) of PCs at video resolutions (640 by 480).

6.2 Hierarchical Z-buffer

The Hierarchical Z-buffer (HZB) [46, 47] is an extension of the popular HSR method, the Z-buffer. It builds upon the technique by Meagher [63] for efficient display of octrees. It uses two hierarchies: an octree in object-precision and a Z-pyramid in image-precision. The Z-pyramid is a layered buffer with a different resolution at each level. At the finest level it is just the content of the Z-buffer; each coarser level is created by halving the resolution in each dimension and each element holding the furthest Z-value in the corresponding 2×2 window of the finer level below. This is done all the way to the top, where there is just one value corresponding to the furthest Z-value in the buffer. During scan-conversion of the primitives, if the contents of the Z-buffer change then the new Z-values are propagated up the pyramid to the coarser levels.

In [46] the scene is arranged into an octree which is traversed top-down front-to-back and each node is tested for occlusion. If at any point a node is found to be occluded then it is skipped; otherwise its children are recursively tested. Any primitives associated with a non-occluded leaf node are rendered and the Z-pyramid is updated. To determine whether a node is visible, each of its faces is tested hierarchically against the Z-pyramid. Starting from the coarsest level, the nearest Z-value of the face is compared with the value in the Z-pyramid. If the face is found to be further away then it is occluded; otherwise it recursively descends down to finer levels until its visibility can be determined.

To allow for real-time performance, a modification of the hardware Z-buffer is suggested that allows for much of the culling processing to be done in the hardware. The process can also be somewhat accelerated through the use of temporal coherence, by first rendering the geometry that was visible from the previous frame and building the Z-pyramid from its Z-buffer.

In [42, 43] Greene presents several optimizations to the original HZB. An extended account of this work is presented in [44]. In this later work, Greene proposes a variation of the original technique suitable for

hardware implementation, that is shown to be very efficient with respect to the bandwidth necessary to update the Z-buffer.

In [45], Greene shows a simple technique for supporting non-conservative culling with a HZB. The basic idea is to change the propagation scheme for the Z-buffer bounds, that is, instead of propagating the farthest z value through the pyramid, Greene proposes to propagate the *e*th-to-the-farthest Z-value, where *e* is a user-defined parameter.

6.3 Hierarchical occlusion map

The hierarchical occlusion map method [97] is similar in principle to the HZB, though it was designed to work with current graphics hardware. In order to do this, it decouples the visibility test into an overlap test (do the occluders overlap the occludee in screen space?) and a depth test (are the occluder closer?). It also supports approximate visibility culling; objects that are visible through only a few pixels can be culled using an opacity threshold.

The occlusion is arranged hierarchically in a structure called the *Hierarchical Occlusion Map* (HOM) and the bounding volume hierarchy of the scene is tested against it. However, unlike the HZB, the HOM stores only opacity information while the distance of the occluders (Z-values) is stored separately. The algorithm then needs to independently test objects for overlap with occluded regions of the HOM and for depth.

During preprocessing, a database of potential occluders is assembled. Then at run-time, for each frame, the algorithm performs two steps: construction of the HOM and occlusion culling of the scene geometry using the HOM.

To build the HOM, a large set of occluders is selected from the occluder database and rendered into the frame-buffer. At this point only occupancy information is required; therefore texturing, lighting and Z-buffering are all turned off. The occluders are rendered as pure white on a black background. The result is read from the buffer and forms the highest resolution in the occlusion map hierarchy. The coarser levels are created by averaging squares of 2×2 pixels to form a map which has half the resolution on each dimension. Texturing hardware can provide some acceleration of the averaging if the size of the map is large enough to warrant the set-up cost of the hardware. As we proceed to coarser levels the pixels are not just black or white (occluded or visible) but can be shades of grey. The intensity of a pixel at such a level shows the opacity of the corresponding region.

An object is tested for occlusion by first projecting its bounding box onto the screen and finding the level in the hierarchy where the pixels have approximately the same size as the extent of the projected box. If the box overlaps pixels of the HOM which are not opaque, it means that the box cannot be culled. If the pixels are opaque (or have opacity above the specified threshold when approximate visibility is enabled) then the object is projected on a region of the image that is covered. In this case a depth test is needed to determine whether the object is behind the occluders.

In paper [97] a number of methods are proposed for testing the depth of the objects against that of the occluders. The simplest test makes use of a plane placed behind all the occluders; any object that passes the opacity test is compared with this. Although this is fast and simple it can be over-conservative. An

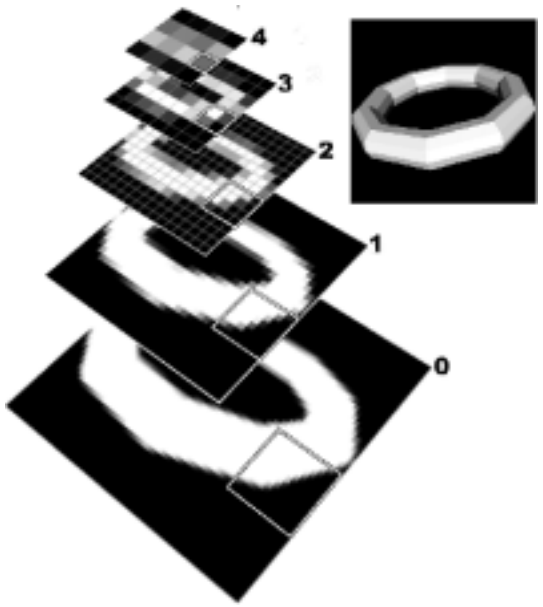


Figure 12: A hierarchy of occlusion maps created by recursively averaging blocks of pixels. Courtesy of Hansong Zhang, UNC.

alternative is the *depth estimation buffer* where the screen space is partitioned into a set of regions and a separate plane is used for each region of the partition.

For efficiency reasons, the finest level of the occlusion map is usually coarser than the image. This could result in sampling artifacts along silhouettes, and thus in non-conservative results. However, the authors report that this usually does not occur in practice.

6.4 Directional discretized occluders

The directional discretized occluders (DDOs) approach is similar to the HZB and HOM methods in that it also uses both object- and image-space hierarchies. Bernardini et al. [12] introduce a method to generate efficient occluders for a given viewpoint. These occluders are then used to recursively cull octree nodes during rendering similarly to HZB and HOM.

In the preprocessing stage, the input model is approximated with an octree and simple, view-dependent polygonal occluders are computed to replace the complex input geometry in subsequent visibility queries. Each face of every cell of the octree is regarded as a potential occluder and the solid angles spanning each of the two halfspaces on the two sides of the face are partitioned into regions. For each region, a flag indicates whether that face is a valid occluder for any viewpoint contained in that region. Each square, axis-aligned face is a view-dependent polygonal occluder that can be used in place of the original geometry in subsequent visibility queries.

Figure 13 is a two-dimensional illustration of the DDO approach. The grid is a discretization of the space surrounding the scene; it represents our octree nodes. The input geometry, *A* and *B*, is shown using

dashed lines. For the purpose of occlusion culling, the geometry *A* can be replaced by a simpler object (shown using thick solid lines) which is a subset of the grid edges, that is, the octree faces. The two figures show the same scene from different viewpoints and view directions. Note that the subset of grid edges that can act as occluders (in place of geometry *A*) changes as the viewpoint changes.

The preprocessing stage is expensive, and may take in the order of hours for models containing hundreds of thousands of polygons. However, the computed occluders are all axis-aligned squares, a fact that can be exploited to design efficient data structures for visibility queries. The memory overhead of the method is only six bitmasks per octree node. Culling methods which need to pre-select large occluders, (e.g. Coorg and Teller [25]), or which pre-render occluders to compute occlusion maps, (e.g. Zhang et al. [98]), could benefit from the DDO preprocessing step to reduce the overhead of visibility tests.

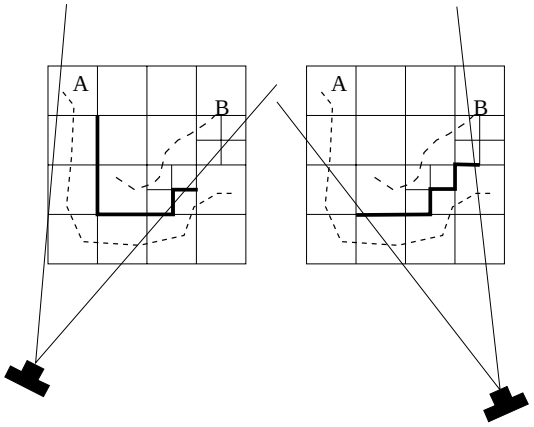


Figure 13: Illustration of the DDO approach. The input geometry, *A* and *B*, is drawn as dashed lines. The valid occluders for the two viewpoints are shown as thick solid lines. Courtesy of James Klosowski, IBM.

6.5 OpenGL-assisted occlusion culling

Bartz et al. in [10,11] describe a different image-precision culling method. The hierarchical representation of the scene is tested against the occluded part of the image, which resembles the HZB and the HOM. However, in contrast to these methods, there is no hierarchical representation of the occlusion, rather OpenGL calls are used to do the testing.

To test for occlusion, a separate buffer, the *virtual occlusion buffer*, is associated with the frame-buffer to detect the possible contribution of any object to the frame-buffer. This is implemented with a stencil buffer. The bounding boxes of the scene are hierarchically sent down the graphics pipeline. As they are rasterized, the corresponding pixels are set in the virtual occlusion buffer whenever the Z-buffer test succeeds. The frame-buffer and the Z-buffer remain unaltered throughout this process, since the rendering of the bounding boxes is used just to query the hardware for visibility information.

The virtual occlusion buffer is then read and any bounding box that has a footprint in it is considered to be (at least partially) visible and the primitives within it can be rendered. Since the operation of reading the

virtual occlusion buffer can be very expensive, it was proposed to sample it by reading only spans from it. The sampling inevitably makes the algorithm a non-conservative test.

The performance of the algorithm depends on the relative cost of reading the virtual occlusion buffer. With common advanced graphics boards the set-up for reading the buffer is a significant portion of the overall time, reducing the usefulness of the method.

6.6 Hardware-assisted occlusion culling

Hardware vendors have started adopting occlusion-culling features into their designs. These are based on a feedback loop to the hardware which is able to check whether any change is made to the Z-buffer when scan-converting a given primitive. Using this hardware feature can avoid rendering a very complex set model by first checking whether it is potentially visible, for example, by checking whether an enclosing primitive (e.g., a bounding box or an enclosing k-dop [53]) is visible, and only rendering the actual object if the simpler enclosing object is indeed visible. Other vendors provide a similar functionality by simply adding instrumentation capabilities to the hardware which is able to count the fragments which pass the depth test (e.g., [75–77]).

Severson [76] estimates that performing an occlusion-query with a bounding box of an object using feedbacks from the graphics card is equivalent to rendering about 190 25-pixel triangles. This indicates that a naive approach where objects are constantly checked for occlusion might actually hurt performance, and not achieve the full potential of the graphics board. In fact, it is possible to slow down the fx6 considerably if one is unlucky enough to project the polygons in a back-to-front order (because none of the primitives would be occluded).

Researchers [9, 55] have explored ways to minimize such visibility queries. In general, by rendering primitives in a front-to-back order, one can potentially minimize the number of necessary hardware queries. Bartz et al. [9] studies the use of k-DOPs [53] for speeding up occlusion-culling queries in such architectures.

In their recent offerings, vendors have improved the occlusion-culling features by performing several occlusion culling queries in parallel [26], or lowering the memory bandwidth required for updating the Z-values [65] (which they claim is the largest user of bandwidth on their cards with texture fetching). There are also reports on the partial implementation of the hierarchical Z-buffer of Greene et al. [46] in hardware.

6.7 Approximate volumetric visibility

One approach to approximate visibility is based on using a volumetric representation, that is, instead of performing geometric visibility computations, one can compute a volume which has intrinsic properties related to the “density” of geometry in the environment, and approximate the visibility between regions by computing the volume opacity between regions. This approach was first proposed by Sillion [78] in the context of speeding up visibility computations for a radiosity system, and extended in [79] into a multi-resolution framework. Volumetric visibility was independently developed by Klosowski and Silva [54, 56] in their PLP system (see below), where it is used to roughly estimate the order of projection of the geometry. The Prioritized-Layered Projection (PLP) algorithm [54,56], is an approximate occlusion-culling technique.

Rather than performing an expensive conservative visibility determination, PLP is an aggressive culling algorithm that estimates the visible primitives for a given viewpoint, and only renders those primitives that it determines to be most likely visible, up to a user-specified budget.

Consequently, PLP is suitable for generating partially correct images for use in a time-critical rendering system. PLP works by initially creating a partition of the space occupied by the geometric primitives. Each cell in the partition is then assigned, during the rendering loop, a probabilistic value indicating how likely it is that the cell is visible, given the current viewpoint, view direction, and geometry in the neighboring cells. The intuitive idea behind the algorithm is that a cell containing much geometry is likely to occlude the cells behind it. At each point of the algorithm, PLP maintains a priority queue, also called the *front*, which determines which cell is most likely to be visible and therefore projected next by the algorithm. As cells are projected, the geometry associated with those cells is rendered, until the algorithm runs out of time or reaches its limit of rendered primitives. At the same time, the neighboring cells of the rendered cell are inserted into the front with appropriate probabilistic values. PLP performs effective visibility estimation by scheduling the projection of cells as they are inserted in the front.

In [55], Klosowski and Silva extend their work into a conservative technique by using image-precision techniques. The new algorithm provides an efficient way of finding the remaining visible primitives by adding a second phase to PLP which uses image-precision techniques for determining the visibility status of the remaining geometry. Another contribution of that work is to show how to efficiently implement such image-precision visibility queries using currently available OpenGL hardware and extensions.

El-Sana et al. [33] present an approach that integrates occlusion culling within the view-dependent rendering framework. View-dependent rendering provides the ability to change the level of detail over the surface seamlessly and smoothly in real-time. The exclusive use of view-parameters to perform level-of-detail selection causes even occluded regions to be rendered with a high level of detail. The authors overcome this drawback by integrating occlusion culling into the level selection mechanism. Because computing exact visibility is expensive and it is currently not possible to perform this computation in real time, they use an approximate visibility estimation technique instead.

6.8 Occluder shadow footprints

Many 3D scenes have in fact only two and a half dimensions. Such a scene is called a *terrain* or *height field*, i.e., a function $z = f(x,y)$. Wonka and Schmalstieg [92] exploit this characteristic to compute occlusions with respect to a point using a Z-buffer with a top parallel view of a scene.

Consider the situation depicted in Figure 14 (side view). They call the part of the scene hidden by the occluder from the viewpoint the *occluder shadow* (as if the viewpoint were a light source). This occluder shadow is delimited by wedges. The projection of such a wedge on the floor is called the footprint, and an occludee is hidden by the occluder if it lies on the shadow footprint and if it is below the edge.

The Z-buffer is used to scan-convert and store the height of the shadow footprints, using an orthographic top view (see Figure 14). An object is hidden if its projection from above is on a shadow footprint and if it is *below* the shadow wedges i.e., if it is occluded by the footprints in the top view.

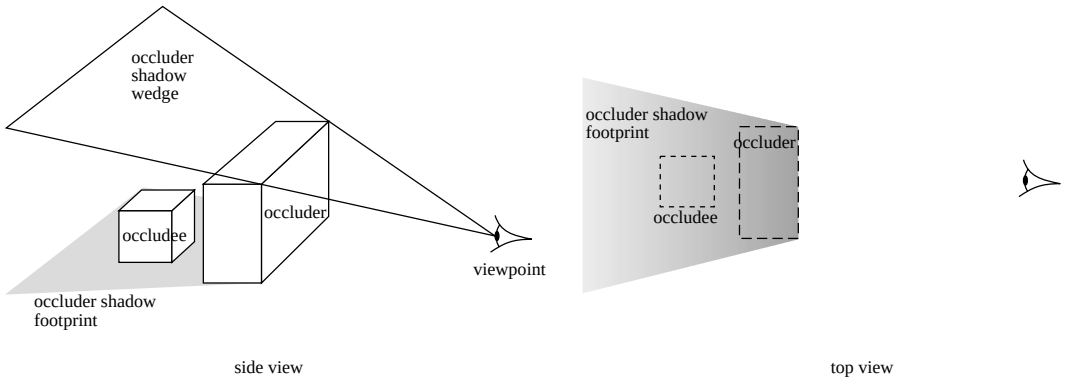


Figure 14: Occluder shadow footprints. A projection from above is used to detect occlusion. Objects are hidden if they are below the occluder shadows. The footprints (with height) of the occluded regions are rasterized using a Z-buffer. Depth is represented as grey levels. Note the gradient in the footprint due to the slope of the wedge.

6.9 Discussion

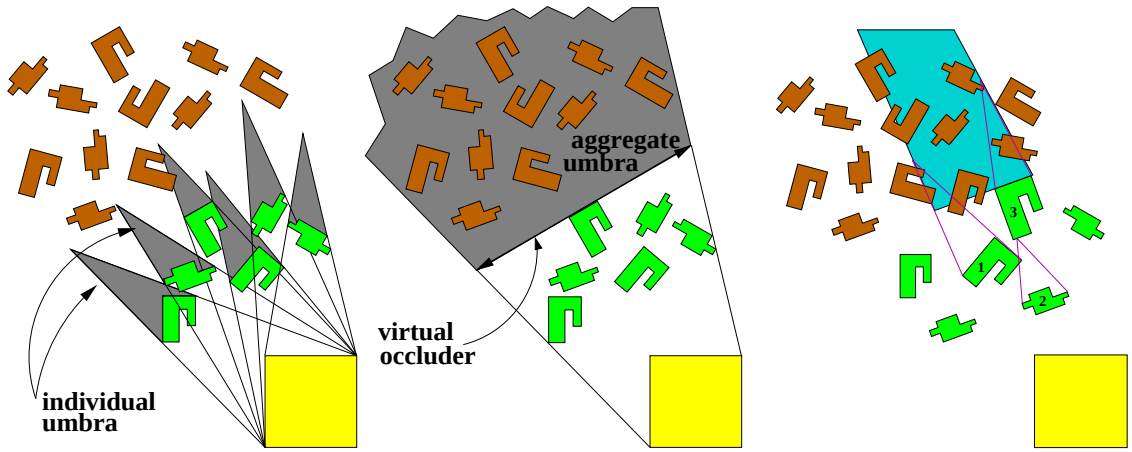
One drawback common to most of the techniques described in this section is that they rely on being able to read information from the graphics hardware. Unfortunately, in most current architectures, using any sort of feedback from the graphics hardware is quite slow and places a limit on the achievable frame rate of such techniques. As Bartz et al. [10] show, these methods are usually only effective when the scene complexity is above a large threshold.

An often overlooked problem is that of latency and rendering pipeline. Indeed, most of these methods assume that the drawing and culling stages are synchronous, that is, the graphics hardware and the application work simultaneously on the same frame. Unfortunately, this is not necessarily the case in modern real-time APIs [72], where three pipeline stages can be run in parallel on three different frames: application scene management, cull and draw. In this case, reading from the Z-buffer for culling would use the information from the previous frame.

7 Generic from-region visibility

In a typical visibility culling algorithm the occlusion is tested from a point [25, 50]. Thus, these algorithms are applied in each frame during the interactive walkthrough. A promising alternative is to find the PVS from a region or view cell, rather than from a point. The computation cost of the PVS from a view cell would then be amortized over all the frames generated from the given view cell. As aforementioned, the predictive capabilities of from-region methods are crucial for pre-fetching.

Effective methods have been developed for indoor scenes [36, 87], but for general scenes, the computation of the visibility set from a region is more involved than from a point. Sampling the visibility from a number of view points within the region [41] yields an approximated PVS, which may then cause unacceptable flickering artifacts during the walkthrough. Conservative methods were introduced in [20, 73] which



(a) (b) (c)

Figure 15: (a) The union of the umbrae of the individual objects is insignificant. (b) But their aggregate umbra is large and can be represented by a single virtual occluder. (c) The individual umbrae (with respect to the yellow view cell) of objects 1, 2 and 3 do not intersect, but yet their occlusion can be aggregated into a larger umbra.)

are based on the occlusion of individual large convex objects.

In these methods a given object or collection of objects is culled away if and only if they are fully occluded by a single convex occluder. It was shown that a convex occluder is effective only if it is larger than the view cell [66]. However, this condition is rarely met in real applications. For example, the objects in Figure 15 are smaller than the view cell, and their umbrae (with respect to the view cell) are rather small. Their union does not occlude a significant portion of the scene (see in (a)), while their aggregate umbra is large (see in (b)).

Recently, new techniques were developed in which the visibility culling from a region is based on the combined occlusion of a collection of objects (occluder fusion). The collection or cluster of objects that contributes to the aggregate occlusion has to be neither connected nor convex. The effective from-region culling of these techniques is significantly larger than previous from-region visibility methods. Below, five techniques are described followed by a discussion.

7.1 Conservative volumetric visibility with occluder fusion

Schaufler et al. [74] introduce a conservative technique for the computation of view cell visibility. The method operates on a discrete representation of space and uses the opaque interior of objects as occluders. This choice of occluders facilitates their extension into adjacent opaque regions of space, in essence, maximizing their size and impact.

The method efficiently detects and represents the regions of space hidden by occluders and is the first

to use the property that occluders can also be extended into empty space provided this space itself is occluded from the view cell. This is proved to be effective for computing the occlusion by a set of occluders, successfully realizing occluder fusion.

Initially, the boundary of objects is rasterized into the discretization of space and the interior of these boundaries is filled with opaque voxels. For each view cell, the occlusion detection algorithm iterates over these opaque voxels, and groups them with adjacent opaque voxels into effective blockers. Subsequently, a shaft is constructed around the view cell and the blocker to delimit the region of space hidden by the blocker. The corresponding voxels are marked as occluded. As regions of space have already been found to be hidden from the view cell, extension of blockers into neighboring voxels can also proceed into these hidden regions realizing occluder fusion with all the occluders which caused this region to be hidden.

As an optimization, opaque voxels are used in the order from large to small and from front to back. Occluded opaque voxels are not considered further as blockers.

To recover the visibility status of objects in the original scene description, the space they occupy is looked up in the spatial data structure and, if all the voxels intersected by the object are classified as hidden, the object is guaranteed to be hidden as well.

The authors present specialized versions for the cases of 2D and 2.5D visibility, and motivate the ease of extension to 3D: because only two convex objects at a time are considered in the visibility classification (the view cell and the occluder), the usual difficulties of extending visibility algorithms from 2D to 3D, caused by triple-edge events, are avoided. Example applications described in the paper include visibility preprocessing for real-time walkthroughs and reduction in the number of shadow rays required by a ray-tracer (see [74] for details).

7.2 Conservative visibility preprocessing using extended projections

Durand et al. [31] (see also [61]) present an extension of point-based image-precision methods such as the Hierarchical Occlusion Maps [98] or the Hierarchical Z-buffer [46] to volumetric visibility from a view cell, in the context of preprocessing PVS computation. Occluders and occludees are projected onto a plane, and an occludee is declared hidden if its projection is completely covered by the cumulative projection of occluders (and if it lies behind). The projection is however more involved in the case of volumetric visibility: to ensure conservativeness, the *Extended Projection* of an occluder underestimates its projection from any point in the viewcell, while the extended projection of an occludee is an overestimation (see Figure 16(a)). A discrete (but conservative) pixel-based representation of extended projections is used, called an *extended depth map*. Extended projections of multiple occluders aggregate, allowing occluder-fusion. For convex viewcells, the extended projection of a convex occluder is the intersection of its projections from the vertices of the cell. This can be computed efficiently using the graphics hardware (stencil buffer) and a conservative rasterization. Concave occluders intersecting the projection plane are sliced (see [31] for details).

A single set of six projection planes can be used, as demonstrated by an example involving a city database. The position of the projection plane is however crucial for the effectiveness of extended projections. This is why a reprojection operator was developed for hard-to-treat cases. It permits a group of

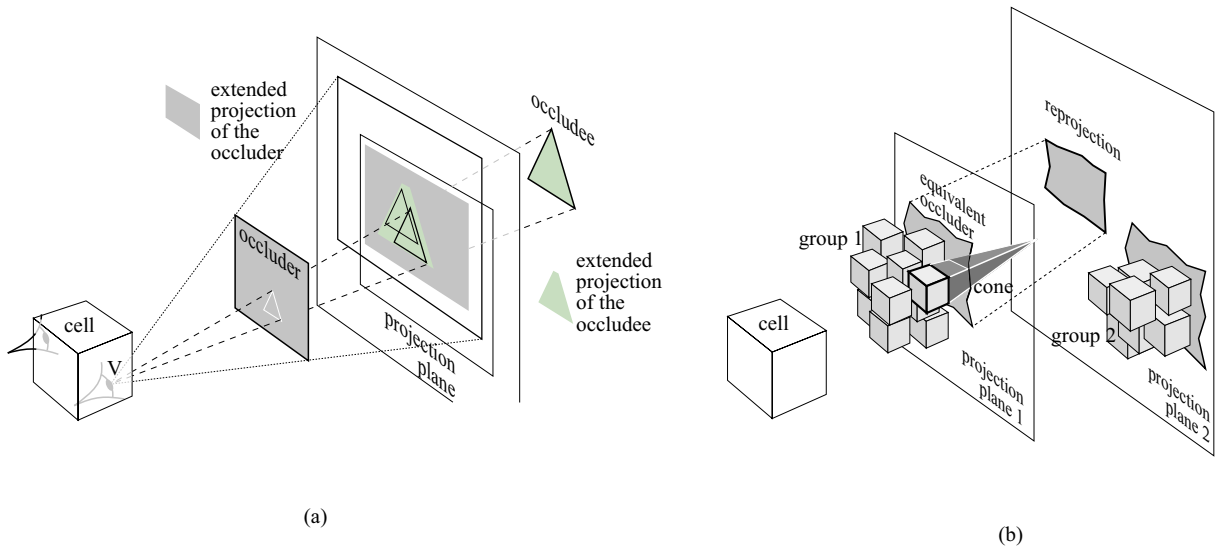


Figure 16: (a) Principle of Extended Projections. The Extended Projection of the occluder is the intersection of its projections from all the points in the viewing cell, while the extended projection of the occludee is the union of its projections. (b) If plane 2 is used for projection, the occlusion of group 1 is not taken into account. The shadow cone of the cube shows that its Extended Projection would be void, since it vanishes in front of the plane. The same constraint applies for group 2 and plane 1. We thus project group 1 onto plane 1, then reproject this aggregate projection onto plane 2.

occluders to be projected onto one plane where they aggregate, and then reprojects this aggregated representation onto a new projection plane (see Figure 16(b)). This reprojection is used to define an occlusion-sweep where the scene is swept by parallel planes leaving the cell. The cumulative occlusion obtained on the current plane is reprojected onto the next plane as well as new occluders. This allows the handling of very different cases such as the occlusion caused by leaves in a forest.

7.3 Virtual occluders

Koltun et al. [57] introduce the notion of from-region virtual occluders, and propose a 2.5D implementation. Given a scene and a view cell, a virtual occluder is a view-dependent (simple) convex object, which is guaranteed to be fully occluded from any given point within the view cell and which serves as an effective occluder from the given view cell. Virtual occluders compactly represent the aggregate occlusion for a given cell. The introduction of such view-dependent virtual occluders enables applying an effective from-region culling technique and efficiently computing a potential visibility set from a cell. The paper presents an object-precision technique that synthesizes such virtual occluders by aggregating the visibility of a set of individual occluders. It is shown that only a small set of virtual occluders is required to compute the PVS efficiently on-the-fly during the real-time walkthrough.

In the preprocessing stage several objects are identified as seed objects. For each seed object, a cluster of nearby objects is constructed so that a single virtual occluder faithfully represents the occlusion of this

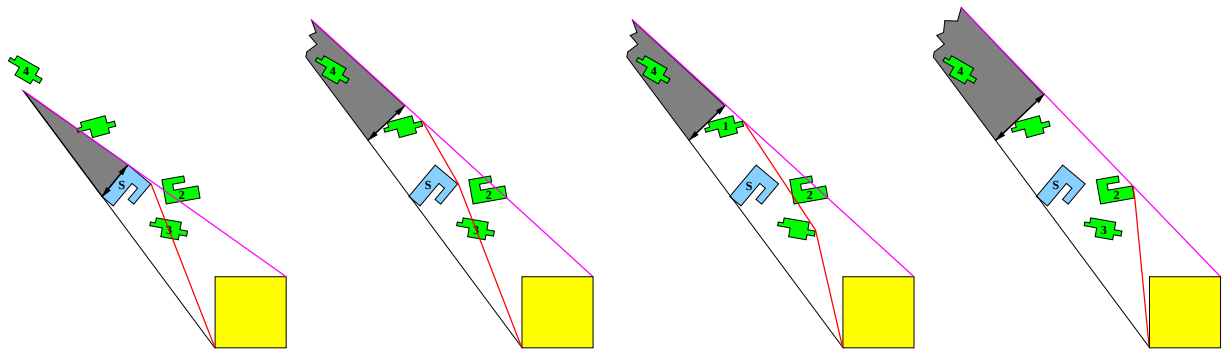


Figure 17: Growing the virtual occluders by intersecting objects with the active separating and supporting lines.

cluster of objects. At first, the cluster is defined to include only the seed object. Then, iteratively, at each step, more objects which satisfy a geometric criterion are added to the cluster of occluders, thus augmenting the aggregate umbra of the cluster. The virtual occluder is placed just behind the furthest object in the cluster, and is completely contained in the aggregate umbra of the cluster (see Figures 15 and 17).

One virtual occluder is stored at each step of the iteration. As a result, at the end of the process, there is a large and highly redundant group of virtual occluders. This group can be well represented by a small subset of the most effective virtual occluders.

In the real-time rendering stage, the PVS of a view cell is computed just before the walkthrough enters the view cell. It is done by hierarchically testing the scene-graph nodes against the virtual occluders. Since only a very small number of them are used, this test is extremely fast.

The 3D problem is solved by a 2.5D implementation, which proves to be effective for most typical scenes, such as urban and architectural walkthroughs. The 2.5D implementation performs a series of slices in the height dimension, and uses the 2D algorithm to construct 2D virtual occluders in each slice. These occluders are then extended to 3D by giving them the height of their respective slices.

7.4 Occluder fusion for urban walkthroughs

Wonka et al. [93] present an approach based on the observation that it is possible to compute a conservative approximation of the umbra for a view cell from a set of discrete point samples placed on the view cell's boundary. A necessary, though not sufficient condition that an object is occluded is that it is completely contained in the intersection of all sample points' umbrae. Obviously, this condition is not sufficient as there may be viewing positions between the sample points where the considered object is visible.

However, shrinking an occluder by ε provides a smaller umbra with a unique property: an object classified as occluded by the shrunk occluder will remain occluded with respect to the original larger occluder when moving the viewpoint no more than ε from its original position.

Consequently, a point sample used together with a shrunk occluder is a conservative approximation for a small view cell with radius ε centered at the sample point. If the original view cell is covered with sample

points so that every point on the boundary is contained in an ε -neighborhood of at least one sample point, then an object lying in the intersection of the umbrae from all sample points is occluded for the original view cell. A too small ε would require applying too many redundant samplings, while a too large ε would shrink the object too much causing the visibility to be overly conservative. Using this idea, multiple occluders can be considered simultaneously. If the object is occluded by the joint umbra of the shrunk occluders for every sample point of the view cell, it is occluded for the whole view cell. In that way, occluder fusion for an arbitrary number of occluders is implicitly performed (see Figure 18).

They recently extended this method to online computation [95]. They proposed an asynchroneous scheme, where a visibility server computes visibility for a region around the current viewpoint and transmits it to the drawing client. Their method is in fact quite generic, it can use occluder shrinking with any point-based occlusion culling method.

7.5 Hardware-accelerated using a dual ray space

Koltun et al. [58] introduce a method that drastically improves from-region techniques for scenes represented by 2.5D models, both in terms of accuracy and speed. It utilizes a dual space transform, enabling visibility to be represented in an alternative two-dimensional space, which allows using graphics hardware for rapidly performing visibility computation.

The algorithm's speed and accuracy aims at computing from-region visibility *on-line*, eliminating the need for preprocessing and storing prohibitive amounts of visibility information on the walkthrough server. A notable advantage of the algorithm is that it retains its speed and accuracy even when applied to large viewcells.

The algorithm processes a model that is represented by a kd-tree. For a given viewcell, the algorithm hierarchically traverses the tree in a top-down fashion. For each node, the algorithm determines whether the bounding box of the node is visible from the viewcell. When an occluded node is reached, the recursion terminates. The fast cell-to-cell visibility determination is the core of the algorithm.

The visibility between two cells is conservatively reduced to a problem of visibility between two segments on a plane: the source segment, representing the viewcell, and the target segment, representing the bounding box of a kd-tree node. Every ray originating in the source segment, and intersecting the target segment, corresponds to a single point in a bounded two-dimensional *dual ray space*. All the rays emanating from the source segment and passing through a segment that represents one occluder form a polygon, which is either a trapezoid or a double-triangle. This polygon represents the rays that are blocked by the occluder. If the occluders together block all the rays that emanate from the source segment, then there is no single visibility ray between the source and the target segments. This can be determined by testing whether the union of the polygons (that correspond to occluders) covers the bounded dual ray space.

This test can be accomplished conservatively by discretizing the union of the polygons into a bitmap using graphics hardware. All the polygons are drawn in white, without Z-buffering or shading, onto an initially black background. If a black pixel remains, the source and target segments are reported to be mutually visible. The discretization avoids the complex analytic computation of the union and alleviates robustness problems common in geometric algorithms. Related approaches can be found in [17, 69].

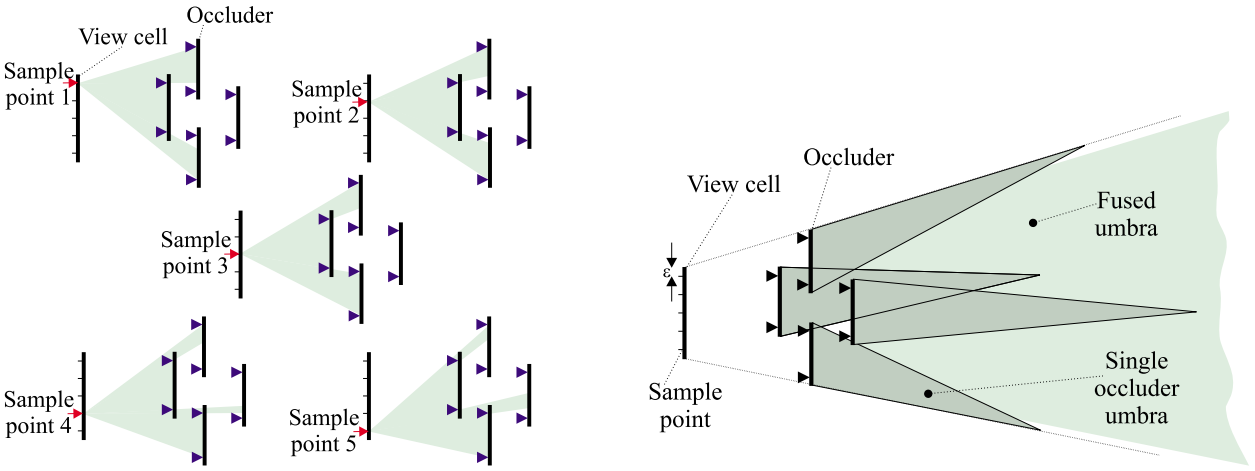


Figure 18: (a) Sampling of the occlusion from five sampling points. (b) The fused umbra from the five points is the intersection of the individual umbrae. It is larger than the union of umbrae of the original view cell. Courtesy of Peter Wonka, Vienna University of Technology.

7.6 Discussion

When the visibility from a region is concerned, occlusion caused by individual occluders in a general setting is insignificant. Thus, it is essential to take advantage of aggregate occlusion caused by groups of nearby objects. The above papers address the problem of occlusion aggregation also referred to as occluder fusion.

All five techniques are conservative; they aggregate occlusion in most cases, but not in all possible ones. In some techniques, the criterion to fuse two occluders or to aggregate their occlusions is based on the intersection of two umbrae. However, in [57, 58, 94], more elaborate criteria are used, which permit aggregation of occlusions even in cases where the umbrae are not necessarily intersected. These cases are illustrated in Figure 15(c). Unfortunately, these methods do not handle the 3D case.

To cope with the complexity of the visibility in 3D scenes, all the techniques use some discretization.

The first method discretizes the space into voxels, and operates only on voxels. This leads to the underestimation of occlusion when the umbra of occluders is relatively small and partially overlaps some large voxels, but does not completely contain any. The advantage of this approach is its generality: it can be applied to any representation of 3D scenes, and not necessarily polygonal.

The second method discretizes the space in two ways. First, it projects all objects onto a discrete set of projection planes, and second, the representation of objects in those planes is also discrete. Moreover, 3D projections are replaced by two 2D projections (see Figure 16), to avoid performing analytical operations on objects in 3D space. The advantage of this algorithm is that, since most operations are performed at image-precision, they can be hardware-assisted to shorten the preprocessing time.

The third method is object-precision analytical in the 2D case. It treats the 3D case as a 2.5D scene

and solves it by a series of 2D cases by discretizing the height dimension. It is shown that in practice the visibility of 2.5D entities approximates well the visibility of the original 3D models.

The forth method samples the visibility from a view cell from a discrete number of sample points. Although it underestimates occlusion, it is also a conservative method. This may be insignificant in the case of close and large occluders, but in cases where the occlusion is created by a large number of small occluders, the approximation might be too crude. The method of Koltun et al. [58] is a significant step towards the computation of from-region visibility in real-time. However, it only deals with 2.5D scenes and seems applicable to urban and architectural models only.

Something that could prove useful when computing visibility from a region is a method for depth-ordering objects with respect to the region. Finding such an ordering can be a challenging task, if at all possible, since it might vary at different sample points in the given region. Chrysanthou in [18] (Section 3.2) suggests a hybrid method based on graph theory and BSP trees which will sort a set of polygons as far as possible and report unbreakable cycles where they are found.

7.7 Approximate from-region visibility

In [4] a scheme to combine approximate occlusion culling with level-of-detail (LOD) techniques is presented. The idea is to identify partially-occluded objects in addition to fully-occluded ones. The assumption is that partially-occluded objects take less space on the screen, and therefore can be rendered using a lower LOD. The authors use the term *Hardly-Visible Set* (HVS) to describe a set consisting of both fully and partially visible objects.

A set of occluders is selected and simplified to a collection of partially-overlapping boxes. Occlusion culling is performed from the view cell using these boxes as occluders to find the ”fully-visible” part of the HVS. It is performed considering only occlusion by individual boxes [20, 73]. There is no occlusion fusion, but a single box may represent several connected occluder objects.

To compute partially-visible objects, all the occluders (boxes) are enlarged by a certain small degree, and occlusion culling is performed again using these magnified occluders. The objects that are occluded by the enlarged occluders and not by the original ones are considered to be partially occluded from the view cell, and are thus candidates to be rendered at a lower LOD.

Several parts of the HVS are computed by enlarging the occluders several times, each time by a different degree, thus, classifying objects with a different degree of visibility. During real-time rendering, the LOD is selected with respect to the degree of visibility of the objects.

It should be noted that this basic assumption of the degree of visibility is solely heuristic, since an object partially occluded from a region does not mean it is partially occluded from any point within the region. It could be fully visible at one point and partially visible or occluded at another.

In [41] another approximate from-region visibility technique is proposed. Casting rays from a five-dimensional space samples the visibility. The paper discusses how to minimize the number of rays cast to achieve a reliable estimate of the visibility from a region.

7.8 The PVS storage space problem

Precomputing the PVS from a region requires solving a prominent space problem. The scene is partitioned into view cells and for each cell a PVS is precomputed and stored readily for the online rendering stage. Since the number of view cells is inherently large, the total size of all the visibility sets is much larger than the original size of the scene. Aside for a few exceptions this problem has not received enough attention yet. Van de Panne and Stewart [88] present a technique to compress precomputed visibility sets by clustering objects and view cells of similar behavior. Gotsman et al. [41] present a hierarchical scheme to encode the visibility efficiently. Cohen-Or et al. [20, 23] deal with the transmission of visibility sets from the server to the client and in [20, 66] discuss the selection of the best view cell size in terms of the size of the PVS.

A completely different approach is taken by Koltun et al. [57]. The PVS of each view cell does not need to be stored explicitly. An intermediate representation that requires much less storage space than the PVS is created and used to generate the PVS on-the-fly during rendering. In [58] they take it one step further by attempting to compute the PVS on the fly during the walkthrough avoiding any precomputation and storage.

8 Conclusion

In summary, this survey is our attempt to cover the visibility literature as it relates to walkthrough applications. Our goal in writing this paper was to produce a study guide for both researchers and practitioners involved in writing real-time rendering systems (e.g., computer games), covering the issues involved, and the available literature. For this, we surveyed most of the relevant visibility literature available, and provided a classification framework. Figure 19 lists and compares the various methods. We see that a considerable amount of knowledge has been assembled in the last decade, and the number of papers in the area has increased substantially in the last couple of years.

Despite the tremendous progress in the area, much interesting work remains:

Quantitative comparison of existing approaches. At this point in time very little work has been done in performing direct comparisons of different techniques. Several factors complicate this, including the fact that very few researchers make their code and data available.

Hardware-assisted culling. As aforementioned, hardware manufacturers integrate more and more occlusion-culling capabilities in the graphics cards. We see the interaction between the hardware and the CPU for efficient high-level culling as an important issue, especially because of latency problems.

From-region visibility. More research is needed to develop a 3D from-region algorithm that computes a tight PVS for large viewing cells.

Preprocessing time; PVS storage. Most from-region techniques perform a considerable amount of preprocessing, which generates quite a bit of storage overhead. Reducing this overhead is an important area of research. Moreover, further research is necessary into techniques which lower the amount of preprocessing

Method	2D/3D	Conservative	Occluders	Fusion	Precomputation	Hardware	Dynamic
From-region cell-portals							
Airey	2D & 3D	yes or no	portals		PVS	no	no
Teller	2D & 3D	yes	portals		PVS	no	no
Object-precision point-based							
Luebke & Georges	3D	yes	portals		cell connectivity	no	need update hierarchy
Coorg & Teller	3D	yes	large convex	if convex silhouette	occluders selection	no	need update hierarchy
Hudson et al.	3D	yes	large convex	no	occluders selection	no	need update hierarchy
Bittner et al.	3D	yes	large	yes	occluder selection	no	need update hierarchy
Klosowski & Silva ’00	3D	no	all	yes	volumetric	no	need update hierarchy
Image-precision point-based							
Greene et al.	3D	yes	all	yes	no	needs z readback	yes, temporal coherence
Zhang et al.	3D	yes or aggressive	big subset	yes	occluder database	current graphics hardware	need update hierarchy
Bernardini et al.	3D	yes	preprocessed	yes	occluders	no	no
Bartz et al.	3D	no	all	yes	no	secondary buffer	need update hierarchy
Klosowski & Silva ’01	3D	yes	all	yes	volumetric	yes	need update hierarchy
Wonka et al. ’99	2.5D	yes	large subset	yes	no	Z-buffer	yes
Generic from-region visibility							
Cohen-Or et al.	3D	yes	all	no	PVS	no	motion volume for occludees
Schaufler et al.	2D or 3D	yes	all watertight	yes	PVS	no	motion volume for occludees
Durand et al.	3D	yes	large subset	yes	PVS	yes	motion volume for occludees
Koltun et al. ’00	2D & 2.5D	yes	large subset	yes	virtual occluders	no	motion volume for occludees
Wonka et al. ’00	2.D	yes	large subset	yes	PVS	yes	motion volume for occludees
Koltun et al. ’01	2.5D	yes	all	yes	no	yes	need update hierarchy

Figure 19: Comparison of the various methods.

required (and not only for from-region techniques, but for visibility culling algorithms in general). Also, memory is a big issue for large scenes, especially in the context of from-region techniques.

Occluder simplification and synthesis. Despite recent work in the area [12, 14, 57, 60, 97], most methods use the raw objects of the scene as occluders. Simplifying these models might speed-up visibility calculations, but that should not come at the cost of conservativeness. Moreover, more appropriate representation might yield simpler computations or more effective culling.

Integration with other acceleration techniques. Visibility culling is only one of a set of real-time rendering techniques. The issue of integrating a visibility-culling algorithm with other forms of acceleration is an important one (see [3, 4, 33, 37]), which we believe is an area still rich in interesting problems.

Dynamic objects. Another largely under-explored area is the handling of dynamic objects [82].

Acknowledgments

We would like to thank all the authors who have helped us with material from their work. In particular, we thank Vladlen Koltun, Pere Brunet, Seth Teller, James Klosowski, Craig Gotsman, Gernot Schaufler and Peter Wonka, for helping us with text and figures from their papers, and Joseph Mitchell for useful comments on this survey.

This research was partially supported by the Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (Center for Geometric Computing and its Applications), and by a grant of the Israel Ministry of Science.

References

[1] John Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations*. PhD thesis, University of North Carolina, Chappel Hill, 1991.

[2] John M. Airey, John H. Rohlf, and Frederick P. Brooks, Jr. Towards image realism with interactive update rates in complex virtual building environments. *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, 24(2):41–50, March 1990.

[3] D. Aliaga, J. Cohen, A. Wilson, Eric Baker, H. Zhang, C. Erikson, K. Hoff, T. Hudson, W. Stuerzlinger, R. Bastos, M. Whitton, F. Brooks, and D. Manocha. MMR: An interactive massive model rendering system using geometric and image-based acceleration. In *ACM Symp. on Interactive 3D Graphics*, pages 199–206, 1999.

[4] Carlos Andujar, Carlos Saona-Vazquez, Isable Navazo, and Pere Brunet. Integrating occlusion culling and levels of details through hardly-visible sets. *Computer Graphics Forum*, 19(3):499–506, 2000.

[5] Arthur Appel. Some techniques for shading machine renderings of solids. In *AFIPS 1968 Spring Joint Computer Conf.*, volume 32, pages 37–45, 1968.

[6] Ulf Assarsson and Tomas Möller. Optimized view frustum culling algorithms for bounding boxes. *Journal of Graphics Tools*, 5(1):9–22, 2000.

[7] Kavita Bala, Julie Dorsey, and Seth Teller. Radiance interpolants for accelerated bounded-error ray tracing. *ACM Transactions on Graphics*, 18(3):213–256, July 1999.

[8] Kavita Bala, Julie Dorsey, and Seth Teller. Ray-traced interactive scene editing using ray segment trees. *Eurographics Rendering Workshop 1999*, June 1999.

[9] D. Bartz, J. Klosowski, and D. Staneker. k-dops as tighter bounding volumes for better occlusion performance. In *ACM SIGGRAPH Visual Proceedings 2001*, 2001.

[10] Dirk Bartz, Michael Meiner, and Tobias Httner. Opengl-assisted occlusion culling for large polygonal models. *Computer & Graphics*, 23(5):667–679, 1999.

[11] Dirk Bartz, Michael Messner, and Tobias Httner. Extending graphics hardware for occlusion queries in opengl. In *Proc. Workshop on Graphics Hardware '98*, pages 97–104, 1998.

[12] F. Bernardini, J. T. Klosowski, and J. El-Sana. Directional discretized occluders for accelerated occlusion culling. *Computer Graphics Forum*, 19(3):507–516, 2000.

[13] J. Bittner, V. Havran, and P. Slavik. Hierarchical visibility culling with occlusion trees. In *Proceedings of Computer Graphics International '98*, pages 207–219, June 1998.

[14] Pere Brunet, Isabel Navazo, Jarek Rossignac, and Carlos Saona-Vázquez. Hoops: 3D curves as conservative occluders for cell-visibility. *Computer Graphics Forum*, 20(3):431–442, 2001.

[15] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. Ph.D. thesis, University of Utah, December 1974.

[16] N. Chin and S. Feiner. Near real-time shadow generation using BSP trees. *ACM Computer Graphics*, 23(3):99–106, 1989.

[17] Franklin S. Cho and David Forsyth. Interactive ray tracing with the visibility complex. *Computer & Graphics*, 23(5):703–717, 1999.

[18] Y. Chrysanthou. *Shadow Computation for 3D Interaction and Animation*. PhD thesis, Queen Mary and Westfield College, University of London, February 1996.

[19] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, October 1976.

[20] Daniel Cohen-Or, Gadi Fibich, Dan Halperin, and Eyal Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum*, 17(3):243–254, 1998.

[21] Daniel Cohen-Or, Eran Rich, Uri Lerner, and Victor Shenkar. A real-time photo-realistic visual fly-through. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):255–264, September 1996.

[22] Daniel Cohen-Or and Amit Shaked. Visibility and dead-zones in digital terrain maps. *Computer Graphics Forum*, 14(3):171–180, August 1995.

[23] Daniel Cohen-Or and Eyal Zadicario. Visibility streaming for network-based walkthroughs. *Graphics Interface '98*, pages 1–7, June 1998.

[24] Satyan Coorg and Seth Teller. Temporally coherent conservative visibility. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 78–87, 1996.

[25] Satyan Coorg and Seth Teller. Real-time occlusion culling for models with large occluders. *1997 Symposium on Interactive 3D Graphics*, pages 83–90, April 1997.

[26] Ross Cunniff. Visualize fx graphics scalable architecture. In *Hot3D Proceedings (talk), Graphics Hardware Workshop*, 2000.

[27] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[28] D. P. Dobkin and S. Teller. Computer graphics. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 42, pages 779–796. CRC Press LLC, Boca Raton, FL, 1997.

[29] S. E. Dorward. A survey of object-space hidden surface removal. *Internat. J. Comput. Geom. Appl.*, 4:325–362, 1994.

[30] F. Durand. *3D Visibility: Analytical study and Applications*. PhD thesis, Universite Joseph Fourier, Grenoble, France, July 1999.

[31] Frédo Durand, George Drettakis, Joëlle Thollot, and Claude Puech. Conservative visibility preprocessing using extended projections. *Proceedings of SIGGRAPH 2000*, pages 239–248, July 2000.

[32] David W. Eggert, Kevin W. Bowyer, and Charles R. Dyer. Aspect graphs: State-of-the-art and applications in digital photogrammetry. In *Proc. ISPRS 17th Cong.: Int. Archives Photogrammetry Remote Sensing*, pages 633–645, 1992.

[33] J. El-Sana, N. Sokolovsky, and C. T. Silva. Integrating occlusion culling with view-dependent rendering. In *IEEE Visualization 2001*, pages 371–378, October 2001.

[34] S. Fleishman, D. Cohen-Or, and D. Lischinski. Automatic camera placement for image-based modeling. In *Proceedings of Pacific Graphics 99*, pages 12–20, October 1999.

[35] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics, Principles and Practice, Second Edition*. Addison-Wesley, Reading, Massachusetts, 1990.

[36] T.A. Funkhouser. Database management for interactive display of large architectural models. *Graphics Interface*, pages 1–8, May 1996.

[37] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of ACM SIGGRAPH 93*, Computer Graphics Proceedings, Annual Conference Series, pages 247–254, August 1993.

[38] Thomas A. Funkhouser, Carlo H. Séquin, and Seth J. Teller. Management of large amounts of data in interactive building walkthroughs. *1992 Symposium on Interactive 3D Graphics*, 25(2):11–20, March 1992.

[39] B. Garlick, D. Baum, and J. Winget. *Parallel Algorithms and Architectures for 3D Image Generation*, volume SIGGRAPH '90 Course Notes, Vol 28, chapter Interactive Viewing of Large Geometric Data Bases Using Multiprocessor Graphics Workstations, pages 239–245. ACM SIGGRAPH, 1990.

[40] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modelling the interaction of light between diffuse surfaces. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18(3), pages 212–22, July 1984.

[41] Craig Gotsman, Oded Sudarsky, and Jeffry Fayman. Optimized occlusion culling. *Computer & Graphics*, 23(5):645–654, 1999.

[42] N. Greene. Occlusion culling with optimized hierachical z-buffering. In *ACM SIGGRAPH Visual Proceedings 1999*, 1999.

[43] N. Greene. Occlusion culling with optimized hierachical z-buffering (cdrom only). In *ACM SIGGRAPH Visual Proceedings 1999*, 1999.

[44] N. Greene. Occlusion culling with optimized hierachical z-buffering. In *In ACM SIGGRAPH Course Notes #30*, 2001.

[45] N. Greene. A quality knob for non-conservative culling with hierarchical z-buffering. In *In ACM SIGGRAPH Course Notes #30*, 2001.

[46] Ned Greene, M. Kass, and Gary Miller. Hierarchical z-buffer visibility. *Proceedings of SIGGRAPH 93*, pages 231–240, 1993.

[47] Ned Greene and Michael Kass. Error-bounded antialiased rendering of complex environments. In *Proceedings of SIGGRAPH '94*, Computer Graphics Proceedings, Annual Conference Series, pages 59–66. ACM SIGGRAPH, July 1994.

[48] P. Hanrahan, D. Salzman, and L. Aupperle. A rapid hierarchical radiosity algorithm. In Thomas W. Sederberg, editor, *ACM Computer Graphics*, volume 25, pages 197–206, July 1991.

[49] Lichan Hong, Shigeru Muraki, Arie Kaufman, Dirk Bartz, and Taosong He. Virtual voyage: Interactive navigation in the human colon. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 27–34. ACM SIGGRAPH, Addison Wesley, August 1997.

[50] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frustra. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 1–10, 1997.

[51] W. F. H. Jiménez, C. Esperança, and A. A. F. Oliveira. Efficient algorithms for computing conservative portal visibility information. *Computer Graphics Forum*, 19(3):489–498, August 2000.

[52] C. B. Jones. A new approach to the 'hidden line' problem. *Computer Journal*, 14(3):232–237, August 1971.

[53] James T. Klosowski, Martin Held, Joseph S. B. Mitchell, Henry Sowizral, and Karel Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, January-March 1998.

[54] James T. Klosowski and Cláudio T. Silva. The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):108–123, April - June 2000.

[55] James T. Klosowski and Cláudio T. Silva. Efficient conservative visibility culling using the prioritized-layered projection algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):365–379, October - November 2001.

[56] James T. Klosowski and Cláudio T. Silva. Rendering on a budget: A framework for time-critical rendering. *IEEE Visualization '99*, pages 115–122, October 1999.

[57] Vladlen Koltun, Yiorgos Chrysanthou, and Daniel Cohen-Or. Virtual occluders: An efficient intermediate pvs representation. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 59–70, June 2000.

[58] Vladlen Koltun, Yiorgos Chrysanthou, and Daniel Cohen-Or. Hardware-accelerated from-region visibility using a dual ray space. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, pages 205–216. Eurographics, June 2001.

[59] Subodh Kumar, Dinesh Manocha, William Garrett, and Ming Lin. Hierarchical back-face computation. *Computers and Graphics*, 23(5):681–692, October 1999.

[60] Fei-Ah Law and Tiow-Seng Tan. Preprocessing occlusion for real-time selective refinement (color plate S. 221). In *Proceedings of the 1999 Symposium on Interactive 3D Graphics*, pages 47–54, New York, April 26–28 1999.

[61] Hong Lip Lim. Toward a fuzzy hidden surface algorithm. In *Computer Graphics International*, Tokyo, 1992.

[62] David Luebke and Chris Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 105–106. ACM SIGGRAPH, April 1995.

[63] D. Meagher. Efficient synthetic image generation of arbitrary 3D objects. In *IEEE Computer Society Conference on Pattern Recognition and Image Processing*, pages 473–478, 1982.

[64] Tomas Möller and Eric Haines. *Real-Time Rendering*. A.K. Peters Ltd., 1999.

[65] Steve Morein. ATI Radeon Hyper-Z Technology. In *Hot3D Proceedings (talk), Graphics Hardware Workshop*, 2000.

[66] Boaz Nadler, Gadi Fibich, Shuly Lev-Yehudi, and Daniel Cohen-Or. A qualitative and quantitative visibility analysis in urban scenes. *Computer & Graphics*, 23(5):655–666, 1999.

[67] Bruce F. Naylor. Partitioning tree image representation and generation from 3D geometric models. In *Proceedings of Graphics Interface '92*, pages 201–212, 1992.

[68] J. O'Rourke. *Art Gallery Theorems and Algorithms*. The International Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

[69] Rachel Orti, Stéphane Riviére, Frédo Durand, and Claude Puech. Radiosity for dynamic scenes in flatland with the visibility complex. *Computer Graphics Forum*, 15(3):237–248, August 1996.

[70] Steven Parker, William Martin, Peter-Pike J. Sloan, Peter S. Shirley, Brian Smits, and Charles Hansen. Interactive ray tracing. *1999 ACM Symposium on Interactive 3D Graphics*, pages 119–126, April 1999.

[71] Harry Plantinga. Conservative visibility preprocessing for efficient walkthroughs of 3D scenes. In *Proceedings of Graphics Interface '93*, pages 166–173, Toronto, Ontario, Canada, May 1993. Canadian Information Processing Society.

[72] John Rohlf and James Helman. IRIS performer: A high performance multiprocessing toolkit for real–time 3D graphics. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 381–395. ACM SIGGRAPH, ACM Press, July 1994.

[73] Carlos Saona-Vazquez, Isabel Navazo, and Pere Brunet. The visibility octree: A data structure for 3D navigation. *Computer & Graphics*, 23(5):635–644, 1999.

[74] Gernot Schaufler, Julie Dorsey, Xavier Decoret, and François X. Sillion. Conservative volumetric visibility with occluder fusion. *Proceedings of SIGGRAPH 2000*, pages 229–238, July 2000.

[75] N. Scott, D. Olsen, and E. Gannet. An overview of the visualize fx graphics accelerator hardware. *The Hewlett-Packard Journal*, May:28–34, 1998.

[76] K. Severson. VISUALIZE Workstation Graphics for Windows NT. HP product literature, 1999.

[77] Inc. Silicon Graphics. SGI Visual Workstation OpenGL Programming Guide for Windows NT. Technical report, Document Number 007-3876-001, 1999.

[78] François X. Sillion. A unified hierarchical algorithm for global illumination with scattering volumes and object clusters. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):240–254, September 1995.

[79] François X. Sillion and George Drettakis. Feature-based control of visibility error: A multi-resolution clustering algorithm for global illumination. *Proceedings of SIGGRAPH 95*, pages 145–152, August 1995.

[80] M. Slater and Y. Chrysanthou. View volume culling using a probabilistic cashing scheme. In S. Wilbur and M. Bergamasco, editors, *Proceedings of Framework for Immersive Virtual Environments FIVE*, December 1996.

[81] Wolfgang Stuerzlinger. Imaging all visible surfaces. *Graphics Interface '99*, pages 115–122, June 1999.

[82] Oded Sudarsky and Craig Gotsman. Dynamic scene occlusion culling. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):13–29, January - March 1999.

[83] I. E. Sutherland, R. F. Sproull, and R. A. Schumaker. A characterization of ten hidden surface algorithms. *ACM Computer Surveys*, 6(1):1–55, March 1974.

[84] Seth Teller. *Visibility Computations in Densely Occluded Environments*. PhD thesis, University of California, Berkeley, 1992.

[85] Seth Teller and Pat Hanrahan. Global visibility algorithms for illumination computations. *Proceedings of SIGGRAPH 93*, pages 239–246, August 1993.

[86] Seth J. Teller. Computing the antipenumbra of an area light source. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):139–148, July 1992.

[87] Seth J. Teller and Carlo H. Sequin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):61–69, July 1991.

[88] Michiel van de Panne and James Stewart. Efficient compression techniques for precomputed visibility. In *Eurographics Rendering Workshop 1999*, June 1999.

[89] I. Wald and P. Slusallek. State-of-the-art in interactive ray tracing, 2001.

[90] I. Wald, P. Slusallek, and C. Benthin. Interactive distributed ray tracing of highly complex models. In *Rendering Techniques 2001 - Proceedings of the 12th EUROGRAPHICS Workshop on Rendering*, pages 274–285, 2001.

[91] I. Wald, P. Slusallek, C. Benthin, and M. Wagner. Interactive rendering with coherent raytracing. *Computer Graphics Forum*, 20(3):153–164, 2001.

[92] Peter Wonka and Dieter Schmalsteig. Occluder shadows for fast walkthroughs of urban environments. *Computer Graphics Forum*, 18(3):51–60, September 1999.

[93] Peter Wonka, Michael Wimmer, and Dieter Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 71–82, June 2000.

[94] Peter Wonka, Michael Wimmer, and Dieter Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. Technical Report TR-186-2-00-06, Institute of Computer Graphics, Vienna University of Technology, Karlsplatz 13/186, A-1040 Vienna, Austria, March 2000. human contact: technical-report@cg.tuwien.ac.at.

[95] Peter Wonka, Michael Wimmer, and François X. Sillion. Instant visibility. *Computer Graphics Forum*, 20(3):411–421, 2001.

[96] A. Woo, P. Poulin, and A. Fourier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–31, 1990.

[97] Hansong Zhang. *Effective Occlusion Culling for the Interactive Display of Arbitrary Models*. Ph.D. thesis, Department of Computer Science, UNC-Chapel Hill, 1998.

[98] Hansong Zhang, Dinesh Manocha, Thomas Hudson, and Kenneth E. Hoff III. Visibility culling using hierarchical occlusion maps. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 77–88. ACM SIGGRAPH, Addison Wesley, August 1997.