Virtual Occluders: An Efficient Intermediate PVS Representation

Vladlen Koltun
Tel Aviv University

Yiorgos Chrysanthou
University College London

Daniel Cohen-Or
Tel Aviv University

Fig. 1. A virtual occluder (the red and white rectangle) represents aggregate occlusion from a region.

Abstract. In this paper we introduce the notion of virtual occluders. Given a scene and a viewcell, a virtual occluder is a view-dependent (simple) convex object, which is guaranteed to be fully occluded from any given point within the viewcell and which serves as an effective occluder from the given viewcell. Virtual occluders are a compact intermediate representation of the aggregate occlusion for a given cell. The introduction of such view-dependent virtual occluders enables applying an effective region-to-region or cell-to-cell culling technique and efficiently computing a potential visibility set (PVS) from a region/cell. We present a technique that synthesizes such virtual occluders by aggregating the visibility of a set of individual occluders and we show the technique's effectiveness.

1 Introduction

Visibility algorithms have recently regained attention in computer graphics as a tool to handle large and complex scenes, which consist of millions of polygons. Twenty years ago hidden surface removal (HSR) algorithms were developed to solve the fundamental problem of determining the visible portions of the polygons in the image. Today, since the z-buffer hardware is the de-facto standard HSR technique, the focus is on visibility culling algorithms that quickly reject those parts of the scene which do not contribute to the final image.

Conventional graphics pipelines include two simple visibility culling techniques: view-frustum culling and backface culling. These visibility techniques are local in the sense that they are applied to each polygon independently of the other polygons in the scene. Occlusion culling is another visibility technique in which a polygon is culled if it

is fully occluded by some other part of the scene. This technique is global and thus far more complex than the above local techniques.

Apparently, occlusion culling techniques and hidden surface removal techniques are conceptually alike and have a similar asymptotic complexity. However, to apply an occlusion culling technique as a quick rejection process, it must be significantly more efficient than the hidden surface removal process. The answer is the use of conservative methods in which for a given scene and view point the conservative occlusion culling algorithm determines a superset of the visible set of polygons [3, 14, 8]. These methods yield a potential visibility set (PVS) which includes all the visible polygons, plus a small number of occluded polygons. Then the HSR processes the (hopefully small) excess of polygons included in the PVS. Conservative occlusion culling techniques have the potential to be significantly more efficient than the HSR algorithms. It should be emphasized that the conservative culling algorithm can also be integrated into the HSR algorithm, aiming towards an output sensitive algorithm [13]. A good overview of most recent culling techniques can be found in [16].

To reduce the computational cost, the conservative occlusion culling algorithms usually use a hierarchical data structure where the scene is traversed top-down and tested for occlusion against a small number of selected occluders [8, 14]. In these algorithms the selection of the candidate occluders is done before the online visibility calculations. The efficiency of these methods is directly dependent on the number of occluders and their effectiveness. Since the occlusion is tested from a point, these algorithms are applied in each frame during the interactive walkthrough.

A more promising strategy is to find the PVS from a region or viewcell, rather than from a point. The computation cost of the PVS from a viewcell would then be amortized over all the frames generated from the given viewcell. Effective methods have been developed for indoor scenes [2, 19, 11, 1], but for general arbitrary 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 [12] yields an approximated PVS, which may then cause unacceptable flickering temporal artifacts during the walkthrough. Conservative methods were introduced in [6, 17] which 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 viewcell [6]. However, this condition is rarely met in real applications. For example the objects in Figure 2 are smaller than the viewcell, and their umbra (with respect to the viewcell) 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 conservative methods are emerging [18, 10, 20] which apply occlusion fusion based on the intersection of the umbrae of individual occluders.

In this paper we present a novel way of representing and computing the visibility from a viewcell. For that purpose, we introduce the notion of virtual occluders. Given a scene and a viewcell, a virtual occluder is a view-dependent (simple) convex object, which is guaranteed to be fully occluded from any given point within the viewcell and which serves as an effective occluder from that viewcell. Virtual occluders compactly represent the occlusion information for a given cell. Each virtual occluder represents the aggregate occlusion of a cluster of occluders. The introduction of such view-dependent virtual occluders enables one to apply an effective region-to-region or cell-to-cell culling technique and to efficiently compute the PVS from a region or a cell. Figure 1 depicts a virtual occluder that aggregates occlusion of four columns in the Freedman Museum model. On the right, the scene is shown from above. The virtual occluder is the vertical

rectangle placed behind the furthest column. On the left, a view is shown from inside the region for which this virtual occluder was computed. The virtual occluder is completely occluded behind the columns (which are rendered transparent, for the sake of demonstration). We present a technique that synthesizes such virtual occluders by aggregating the occlusion of a set of individual occluders and show its effectiveness.

The rest of the paper is organized as follows: We give an overview of the method in Section 2, as well as summarizing its main contributions. In Section 3 we describe the algorithm for constructing the set of virtual occluders. The results and their analysis are presented in Section 4, and we conclude in Section 5.

2 Overview

The virtual occluders are constructed in preprocessing. For simplicity in the discussion we assume regular partitioning of the scene into axis-aligned box-shaped cells. However, this is not inherent to our algorithm, which may handle any partitioning of the scene into cells of arbitrary non-convex shape. This algorithm is applied to a given viewcell and constructs a set of virtual occluders that effectively represents the occlusion from this cell. It yields a large, dense set of potential virtual occluders. From this set, an effective small sorted subset is selected and stored for the on-line stage. Since the virtual occluders are large, convex and few, the PVS of the associated viewcell can be quickly constructed by applying a simple and effective culling mechanism similar to [6, 17].

The PVS of a viewcell is constructed only once before the walkthrough enters the cell, by culling the scene against the virtual occluders. The frame-rate of the walkthrough is not significantly interrupted by the visibility determination, since the cost of constructing the viewcell's PVS is amortized over the large number of frames during the walk through the cell. Note that one of the advantages of our method is that it generates large effective occluders and thus enables the use of a larger viewcell, which further reduces the relative cost of computing the PVS.

The main advantages of the presented method can be summarized as follows:

Aggregate occlusion. Each virtual occluder encapsulates the combined contribution of a cluster of occluders. This results in the ability of culling larger portions of the scene-graph using just a single virtual occluder. Moreover, a small set of virtual occluders faithfully represents the occlusion from a viewcell.

Accuracy. The presented method for constructing virtual occluders is an object-space continuous method. Their shape and location are not constrained by a space partition of the scene (e.g., quadtree or kd-tree). The placement of the virtual occluders adapts to the scene and not to an independent fixed subdivision. This leads to accuracy and thus a stronger conservative visibility set.

Speed. Since the number of per-viewcell virtual occluders is small, the visibility culling process is faster. The virtual occluders occlude more than the individual objects, and are thus able to cull larger cells of the scene-graph. This results in a highly reduced amount of computation at run-time for each viewcell.

Storage size. Given a viewcell and a small set of virtual occluders, the PVS can be computed on-the-fly during the walkthrough. This avoids storing the PVS but rather a small set of virtual occluders, which requires less space. This is vital since the potential visibility sets of all viewcells of a complex scene tend to be too large for storage (see Section 4).

There are various applications that can benefit from virtual occluders. All of them exploit the fact that the cost of computing the conservative visibility set can be amortized over several frames. Rendering the scene consists of three processes:

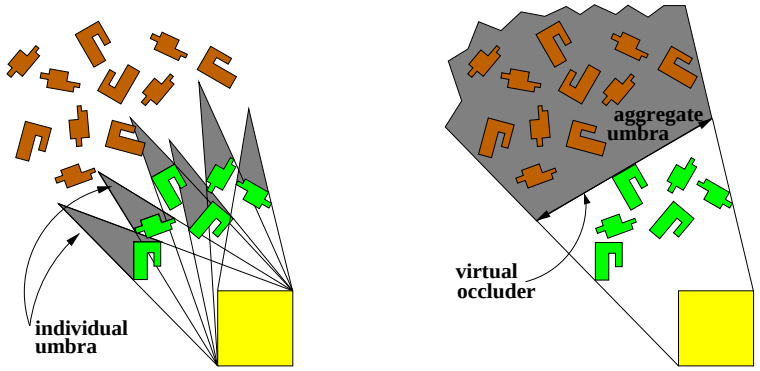

Fig. 2. The union of the umbrae of the individual objects is insignificant, while their aggregate umbra is large and can be represented by a single virtual occluder.

1. computing the viewcell virtual occluders;
2. computing the viewcell PVS;
3. rendering the PVS.

The first process is computed offline, while the other two online. We can consider a rendering system, which is conceptually partitioned into a client and a server. The server can compute the virtual occluders in a preprocess and store them in a spatial data structure. During the walkthrough, the client uses the virtual occluders to compute the PVS. The PVS must be readily available for the real-time rendering of the scene. This requires the system to precompute the PVS of nearby viewcells before the walkthrough enters those viewcells. As mentioned in the introduction, a remote walkthrough application necessarily requires the computation of a from-region PVS to avoid the latency problem [7].

3 Constructing the Virtual Occluders

In this section we show how to construct a set of virtual occluders that represents the aggregate occlusion from a given viewcell. The algorithm description is visualized with a set of illustrations: Figure 2 shows the viewcell in yellow and a set of occluders. The umbrae of the individual occluders is illustrated in 2 (a), showing that the occlusion of the individual objects is insignificant. Nevertheless, the aggregate occlusion of these objects is much larger, as can be seen in 2 (b). To construct virtual occluders that effectively capture the aggregate occlusion we use the following algorithm: (1) select a set of seed objects, (2) build a set of virtual occluders from a given seed and the cluster of objects around this seed and (3) decimate the initial dense set of virtual occluders to a cost effective smaller set. The exact definitions and details as applied for a given viewcell are elaborated below.

The set of seed objects is defined according to the solid-angle criterion [14] defined from the viewcell center. Objects with a large solid-angle are likely to be effective occluders from the given viewcell and thus included in a cluster of occluders that builds up larger occlusion. The seed object in Figure 3 is colored in light blue. It should be noted that the algorithm is not sensitive to the accuracy of the definition of the set of seed ob-

jects. However, the more seeds used, the better the set of virtual occluders is in terms of its effectiveness (less conservative).

For a given seed object we now construct an aggregate umbra starting from its own umbra and augmenting it with the occlusion of its surrounding objects. First, the two supporting lines that connect the viewcell and object extents build the initial umbra. An initial virtual occluder is placed behind the object in its umbra (see Figure 3 (a)). Now, let us first assume that during this process one of the supporting lines is defined as the *active supporting line* while the other remains static (the active supporting line is drawn in purple). If the active line intersects an object, then this object is a candidate to augment the umbra. If the candidate object intersects the umbra of the seed object, then it augments the umbra and the active line shifts to the extent of the inserted object (see Figure 3 (b)). By iteratively adding more and more objects the umbra extends, and gets larger and larger. There are cases where a candidate object does not intersect the current umbra, but can still augment it. To treat these cases we define and maintain the *active separating line(polyline)* (colored in red).

Initially, the active separating line is defined between the seed object and the viewcell (in the standard way [8]). Then objects which intersect the active separating line redefine it to include the new objects and the separating line becomes a polyline. In Figure 3 (b) we can see that object 2, which intersects the active supporting line, but not the active separating line, cannot contribute its occlusion to the augmented umbra before the contribution of object 3 is considered. As illustrated in Figure 3 (b), object 3 intersects the active separating line and thus redefines it to the polyline shown in Figure 3 (c). Then, object 2 intersects both active lines, augments the aggregate umbra and extends the virtual occluder further (3 (d)). Formally, let us define the evolving aggregate umbra U, the active supporting line P, and the active separating polyline Q. Given an object B:

1. If B intersects U then B updates U, Q and P, and a new virtual occluder is placed behind B.
2. If B intersects only Q then Q is updated to include B.
3. If B intersects both Q and P then B updates U, Q and P, and the furthest virtual occluder is extended to the new location of P.

Once no more objects intersect the active lines, the static line on the other side is activated, the process is repeated for the other side aiming to further augment the umbra by adding objects from the other side of the seed object. In our implementation both left and right separating lines are maintained active and the insertion of objects can be on either side of the umbra. Thus, the initial active supporting and separating lines are as shown in Figure 4. Note that in case 2 above, B has to be above the opposite active polyline. Since a virtual occluder is placed behind all the individual objects that make it up, as it grows bigger it also grows further away from the viewcell. For this reason we periodically 'dump' some of the intermediate virtual occluders into the dense set.

This aggregate umbra algorithm bears some conceptual similarity to algorithms that compute shadow volumes from an area light source [4, 5], or even to discontinuity meshing methods [9, 15]. However, here we have two important advantages. First, the aggregate umbra does not necessarily have to be accurate, but conservative, and can thus be calculated significantly faster than area shadow algorithms. The other advantage lies in the effectiveness of the aggregation. While shadow algorithms detect additional objects that intersect an umbra and expand it, they don't make full use of the separating lines. See the example in Figure 5, even if the polygons are processed in front-to-back order, none of the shadow methods successfully merge the umbrae into one, in contrast to the

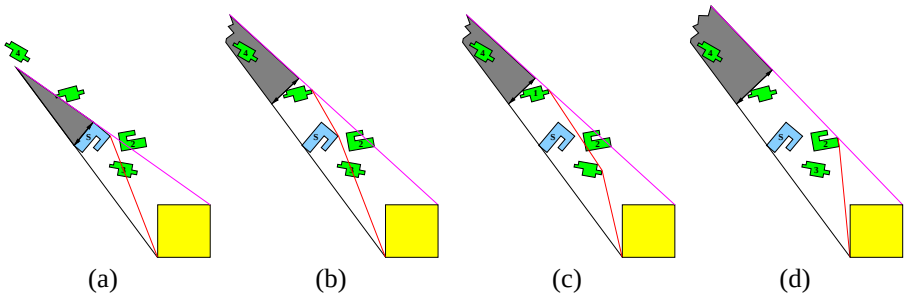

Fig. 3. Growing the virtual occluders by intersecting objects with the active separating and supporting lines.

method presented here.

After a dense set of virtual occluders is computed, it is sufficient to select only a small subset of them for the on-line stage. This saves the per-cell storage space and accelerates the process of computing the cell visibility set. The idea is that the subset can represent the occlusion faithfully, since there is a large amount of redundancy in the dense set. In practice, we have found that just less than ten virtual occluders per cell are sufficient to represent the occlusion effectively.

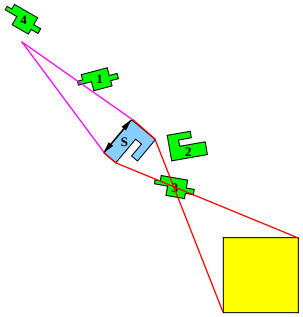

Fig. 4. The active lines are processed on both sides simultaneously.

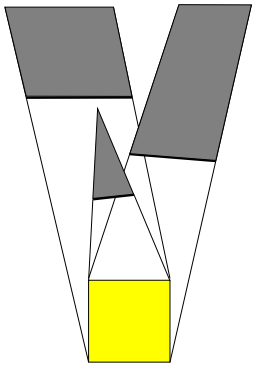

Fig. 5. Current shadow algorithms do not necessarily aggregate the occlusion.

The greedy algorithm that is used to select this subset is described in Figure 6. Figure 8 shows a subset of virtual occluders selected by the algorithm. The key observation behind this algorithm is that the addition of a virtual occluder to the subset is a tradeoff - the additional virtual occluder improves the PVS by occluding some part of the scene, but the down-side is that the process of computing the PVS will take longer. Therefore the algorithm selects the most effective occluders one by one, until the addition of another occluder is not cost effective since the occlusion contributed by it is not significant enough to justify the enlargement of the subset. A beneficial consequence of this algorithm is that the final subset is sorted, i.e. the most effective occluders appear first. This will accelerate the PVS construction process. This algorithm of course is non-optimal, but practically, there is no significant difference since conservative occlusion is not sensitive to small details.

```
for each virtual occluder O in the dense set D do
    initialize weight of O
        to the size of the area it occludes.
endfor
repeat
    let V be the virtual occluder with largest weight in D
    if V contributes effectively to occlusion
    then
        add it to the final sorted set
        remove it from D
        for every occluder O in D do
            reduce the weight of O by the size of the
            area occluded both by O and V
        endfor
    else
        output the final set, and exit
    endif
endrepeat
```

Fig. 6. Algorithm for selecting a sorted effective subset from a dense set of virtual occluders.

3.1 Treating 3D Problems Using a 2.5D Visibility Solution

Above we have described the algorithm in 2D. It is possible to extend the algorithm to 3D by extending the supporting and separating active lines to their counterpart planes in 3D. Then objects intersecting one of the active planes update these planes similarly to the 2D case. However, in 3D the complexity and running time of the process increases considerably. Fortunately, in practice, full 3D visibility culling is not always necessary. When dealing with typical (outdoor and indoor) walkthroughs, a 2.5D visibility culling is almost as effective but much faster. Moreover, in these cases a simpler implementation of the technique significantly accelerates the process, while losing insignificant conservativeness.

The 2.5D visibility problem is reduced to a set of 2D problems by discretizing the scene at numerous heights, while considering the perspective adjustment from the viewcell. For each height we run the 2D algorithm that constructs virtual occluders using only the parts of the objects that extend from the ground up to and above the given height. These virtual occluders extend from the ground to the associated height. This yields a dense set of virtual occluders of different heights. The degree to which this representation is conservative, depends on the discretization resolution. This approach is conservative, because in 2.5D, if a vertical object (e.g. virtual occluder) is completely occluded by other objects at a given height, it is guaranteed to be occluded by them at all smaller heights as well.

The discretization does not necessarily lead to a loss of precision. In practice it is enough to construct virtual occluders at only a small number of "interesting" heights. There is a tradeoff between the precision and the processing time. As a heuristic strategy, a height-histogram of the scene is quantized and analyzed to select an appropriate number of heights for slicing the scene. The histogram of heights is constructed by considering the perspective-adjusted heights of the scene, as seen from the cell. In practice, five or less slices provide sufficiently good results, as shown in Section 4. In should be emphasized that the virtual occluders are conservative by nature and the effectiveness of the method is not sensitive to small details of the occluders.

4 Results

We have implemented the described algorithms in C-language using the OpenGL libraries. The tests described below were carried out on an SGI InfiniteReality, with a 196Mhz R10000 processor. We have tested the method on two highly complex scenes. One is a model of London, accurately depicting an area of 160 sq. kilometers (some parts are shown in Figures 8 and 10). The model was created from detailed maps and consists of over 250K objects having more than 4M vertices. The other model is the Freedman virtual museum, which spans an area of approximately 50,000 sq. feet, and consists of about a thousand objects having altogether 85K vertices (see Figure 9).

London model

# VO	% of occ.	δ	PVS (#vertices)
1	43.73	43.73	2607241
2	72.05	28.31	1295049
3	86.93	14.88	605592
4	93.92	6.98	281713
5	95.91	1.99	189508
6	96.52	0.61	161244
7	96.77	0.25	149660
8	96.95	0.18	141320

Freedman Museum model

# VO	% of occ.	δ	PVS (#vertices)
1	23.41	23.41	65353
2	33.17	9.75	57025
3	41.95	8.78	49533
4	47.80	5.85	44541
5	49.75	1.94	42877
6	51.55	1.80	41341
7	53.26	1.71	39882
8	54.73	1.46	38628

Table 1. The magnitude of occlusion as a function of the number of virtual occluders saved for real-time use. A small number of virtual occluders represent most of the occlusion.

Figure 7 shows how the aggregate umbra of the virtual occluders improves the performance compared to an occlusion culling based on individual objects, for three different cell sizes. Each line shows the percent of occlusion along a long path around the city. We see that as the viewcells get larger the relative effectiveness of using virtual occluders increases, while the effectiveness of the individual occluders sharply decreases. We see that for large viewcells the occlusion of individual buildings is on average less than two percent of the scene, while virtual occluders occlude more than 85%. For smaller cells the occlusion by virtual occluders is on average more than 98%. An example of the effectiveness of virtual occluders can be seen in Figure 8. Note that the vast majority of the occluded objects are culled by the eight virtual occluders, while only an insignificant fraction of the scene can be culled by individual objects.

The London model is partitioned into 16x10 regions of one squared kilometer, and each region is partitioned by a kd-tree into cells (typically hundreds of cells). Without

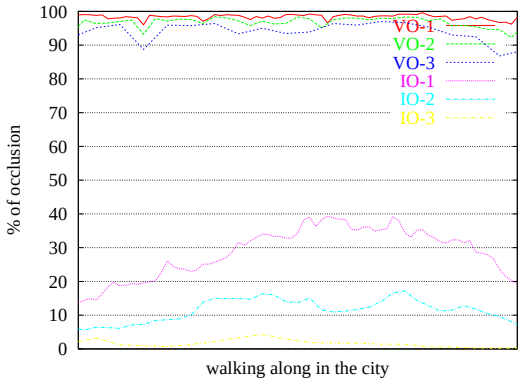

Fig. 7. A comparison between the occlusion of individual occluders and virtual occluders. The graph displays the percentage of occlusion for all viewcells along a path across the London model. Different colors correspond to different cell sizes. Red corresponds to cells of size 160x100 meters; green for 320x200 meters; blue for 640x400.

# slices	% of occlusion
5	97.27
4	96.91
3	96.21
2	94.52
1	72.07

Table 2. The occlusion as a function of the number of height slices.

occlusion culling the system cannot render the entire scene, but rather few regions only (*see in the attached video*). The size of the PVS of a typical viewcell is only few thousands of kd-tree cells or thousands of buildings. Figure 11 shows a view of London with a virtual occluder (colored in red) with respect to a viewcell (marked in light green). The buildings that are occluded by this virtual occluder are colored in blue.

The Museum model is an example of a sparse model, where less occlusion is present from any given region. In our tests, we have found that no object occludes other objects in the scene single-handedly. Nevertheless, more than 50% of the scene are usually found occluded, when virtual occluders are used. In this case, the virtual occluders faithfully represent the little occlusion present, despite the sparse nature of the scene and the ineffectiveness of its individual objects.

Table 1 shows the effectiveness of a small set of virtual occluders in terms of their occlusion. We can see that using just five virtual occluders already provides an effective occlusion of 95% of the London model. The use of more than ten virtual occluders does not contribute much to the occlusion. This means that in terms of per-viewcell storage space the virtual occluders are by far more economical than naive storage of the viewcell PVS list. A virtual occluder is a vertical quadrilateral, represented by opposite corners, which can be represented by only five values (the 2D endpoints and its height). These coordinate values can be quantized to one byte each, since the effective occlusion of the virtual occluder is not sensitive to its fine sizes. Thus, storing ten virtual occluders per viewcell requires just fifty bytes.

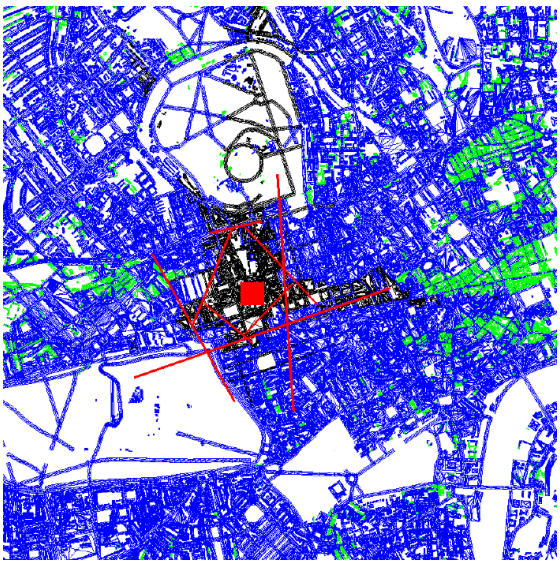

Fig. 8. A top view of central London. Virtual occluders (in red) are placed around the viewcell (red square). The blue buildings are those occluded by the virtual occluders and the green ones are those occluded by individual buildings. Only a small set of buildings remains potentially visible (colored in black) after using just eight virtual occluders.

Table 2 shows how the slicing resolution affects the conservativeness of the virtual occluders. The table shows the size of the PVS as a function of the number of slices. We see that the size of the PVS is improved greatly by taking only two slices. Using more than five slices does not yield a significant reduction in the size of the PVS.

For an average viewcell, it takes about a minute to produce the dense set of virtual occluders and their decimation into an effective small set. Considering the typical size of the viewcell in the London model, e.g., 200x200 to 600x600 meters, it may take a user less time to cross a viewcell at walking-speed. However, once the virtual occluders are given, the time spent on the computation of the PVS is negligible provided the scene is traversed hierarchically in a standard top-down fashion.

5 Conclusion

We have presented the new concept of virtual occluders as a means for representing the aggregate occlusion of groups of objects. They can have forms other than the one presented, but the idea is that they are an effective intermediate representation of the occlusion from a cell. One of their important features is that they are ordered in terms of importance. This provides an efficient culling mechanism since the visibility test of each object is applied first with the most effective occluders. Only those few objects that are not culled by the first most effective virtual occluders are tested against the rest of the occluders down the ordered list.

It is important to note that in the London model the buildings are fairly simple and consist of a relatively small number of polygons. This means that level-of-detail techniques (LOD) cannot help much in rendering such a huge model. Thus, occlusion culling

is a vital tool for such walkthrough application. In other cases a scene can consist of some very detailed geometric models. This would require incorporating dynamic LOD techniques, image-based rendering and modeling, and other acceleration techniques to handle rendering the potential visibility sets.

References

1. Michael Abrash. *Zen of Graphics Programming*. Coriolis Group Books, 2^{nd} edition, 1996.
2. J. Airey, J. Rohlf, and F. Brooks. Towards image realism with interactive update rates in complex virtual building environments. *ACM Siggraph Special Issue on 1990 Symposium on Interactive 3D Graphics*, 24(2):41–50, 1990.
3. 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.
4. A. T. Campbell, III and D. S. Fussell. An analytic approach to illumination with area light sources. Technical Report R-91-25, Dept. of Computer Sciences, Univ. of Texas at Austin, August 1991.
5. N. Chin and S. Feiner. Fast object-precision shadow generation for area light sources using BSP trees. In *ACM Computer Graphics (Symp. on Interactive 3D Graphics)*, pages 21–30, 1992.
6. Daniel Cohen-Or, Gadi Fibich, Dan Halperin, and Eyal Zadicario. Conservative visibility and strong occlusion for viewspace partitionin of densely occluded scenes. *Computer Graphics Forum*, 17(3):243–254, 1998. ISSN 1067-7055.
7. Daniel Cohen-Or and Eyal Zadicario. Visibility streaming for network-based walkthroughs. *Graphics Interface '98*, pages 1–7, June 1998. ISBN 0-9695338-6-1.
8. Satyan Coorg and Seth Teller. Real-time occlusion culling for models with large occluders. *1997 Symp. on Interactive 3D Graphics*, pages 83–90, April 1997. ISBN 0-89791-884-3.
9. G. Drettakis and E. Fiume. A fast shadow algorithm for area light sources using backprojection. In Andrew Glassner, editor, *ACM Computer Graphics*, pages 223–230, July 1994.
10. Frédo Durand, George Drettakis, Joëlle Thollot, and Claude Puech. Conservative visibility preprocessing using extended projections. *To appear in the proceedings of SIGGRAPH 2000*, 2000.
11. T.A. Funkhouser. Database management for interactive display of large architectural models. *Graphics Interface*, pages 1–8, May 1996.
12. Craig Gotsman, Oded Sudarsky, and Jeffry Fayman. Optimized occlusion culling. *Computer & Graphics*, 23(5):645–654, 1999.
13. Ned Greene and M. Kass. Hierarchical Z-buffer visibility. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 231–240, 1993.
14. T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frusta. In *Proceedings of the 13th International Annual Symposium on Computational Geometry (SCG-97)*, pages 1–10, New York, June4–6 1997. ACM Press.
15. D. Lischinski, F. Tampieri, and D. P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics & Applications*, 12(6):25–39, November 1992.
16. Tomas Moller and Eric Haines. *Real-Time Rendering*. A. K. Peters Limited, 1999.
17. Carlos Saona-Vazquez, Isabel Navazo, and Pere Brunet. The visibility octree: A data structure for 3d navigation. *Computer & Graphics*, 23(5):635–644, 1999.
18. Gernot Schaufler, Xavier Decoret, Julie Dorsey, and Francois Sillion. Conservative volumetric visibility with occluder fusion. *To appear in the proceedings of SIGGRAPH 2000*, 2000.
19. Seth J. Teller and Carlo H. Sequin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics (Proceedings of SIGGRAPH 91)*, 25(4):61–69, July 1991.
20. 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.

Fig. 9. A ray-traced view over the Freedman virtual museum with the ceiling removed. The yellow square in the bottom is the viewcell and the red and white rectangle is one of the virtual occluders.

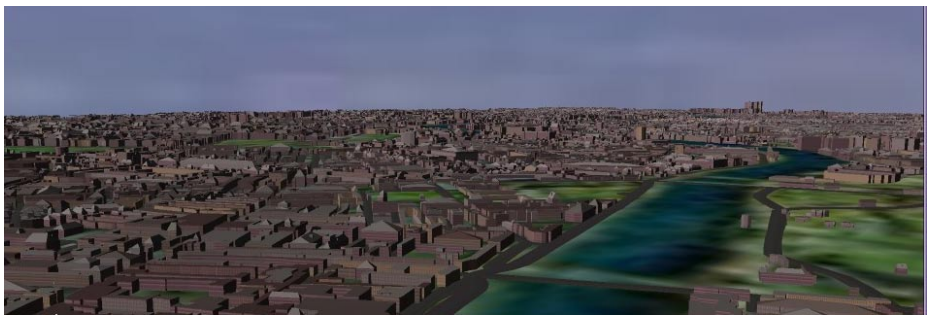

Fig. 10. A view over a part of the London model.

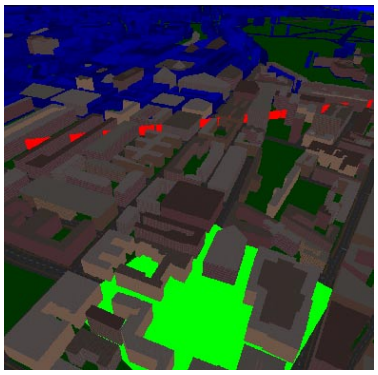

Fig. 11. A viewcell (marked in light green) and one of the corresponding virtual occluders (the long red rectangle piercing through the buildings). The buildings that are occluded by this virtual occluder are colored in blue.