Visibility Streaming for Network-based Walkthroughs

Daniel Cohen-Or and Eyal Zadicario

Computer Science Department
Tel-Aviv University, Ramat-Aviv 69978, Israel

Abstract

In network-based walkthroughs the server transmits on-line only the primitives which are potentially visible from the client's current viewpoint (visibility streaming). To reduce the bandwidth requirements it is necessary to minimize the set of view-dependent potentially visible primitives using occlusion culling techniques. However, even a slight change in the viewpoint might require the computation and transmission of many new primitives and thus latency is inevitable.

In this paper we present an algorithm for determining a conservative superset of an ϵ-neighborhood of a given viewpoint. Having such an ϵ-superset, the client is free to render the model independently of the server as long as he moves within that ϵ-neighborhood.

Résumé

Dans un parcours via le re'seau, le serveur ne transmet que les primitives qui sont potentiellement visibles a' partir du point de vue courant du client. Afin de re'duire les contraintes sur la bande passante il est ne'ce'ssaire de minimiser l'ensemble des primitives dont la visibilite' de'pend du point de vue. Ceci est re'alise' par des techniques d'e'limination par occlusion. Toutefois, même de faibles variations du point de vue peuvent impliquer le calcul et la transmission de nombreuses nouvelles primitives et donc des temps de latences sont ine'vitables.

Dans ce papier nous pre'sentons un algorithme qui de'termine un sur-ensemble stable d'un ϵ-voisinage d'un point de vue donne'. Avec un tel ϵ-sur-ensemble le client peut visualiser le mode'le inde'pendemment du serveur aussi longtemps qu'il se de'place a l'inte'rieur de ce ϵ-voisinage.

Keywords: Visibility, Walkthrough, Streaming, Occlusion, Culling, Client/Server.

Introduction

Interactive network-systems based on client/server computation are introducing new challenges to computer graphics. In remote walkthroughs, the virtual environment, consisting of millions of primitives, is stored in a remote server and the clients are usually low-end computers. With the increasing popularity of the Web, the network bandwidth and transmission latency have become critical bottlenecks for interactive rate remote walkthroughs. Although the rendering power of graphics workstations has recently significantly increased, common low-end computers cannot render a large number of geometric primitives in real-time.

To alleviate these two bottlenecks, the server and the client should collaborate during the walkthrough. The client constantly updates the server about his location and viewing parameters and the server transmits only *view-dependent* data which is necessary to the client for a proper rendering (see [12, 13]). There are different methods to reduce the complexity of the model representation with respect to the viewpoint based on view-dependent object simplification and visibility culling [4].

Visibility culling aims at determining the set of invisible primitives from a given viewpoint, to avoid their further processing. View-frustum culling culls away a subset of the invisible primitives not lying in the current view frustum. However, there might still be too many primitives within the view frustum, many of which might be obscured by other primitives. By taking occlusions into consideration, it is possible to determine a visible set (either exact or overestimated). These *occlusion culling* techniques were developed to accelerate the rendering of large geometric models [6, 8, 17, 18] by reducing the number of polygons fed into the graphics hardware.

In spite of the apparent potential of such culling techniques, there is a drawback which prevents their direct adaptation for a network-based system. To maintain a correct visible set, the server will need to constantly update the client's visible set according to the moving client viewpoint [7]. Even a slight change in the viewpoint might require the computation and transmission of many new primitives and thus some latency is inevitable.

Hence, given a viewpoint and a model, what is actually needed is a *superset* of the visible set which includes at least all the visible primitives seen from an ϵ-neighborhood of the viewpoint. Having such an ϵ-superset, the client is free to render the model independently of the server as long as he moves within that ϵ-neighborhood. The solution required must be *conservative* in the sense that it includes **all** the visible primitives

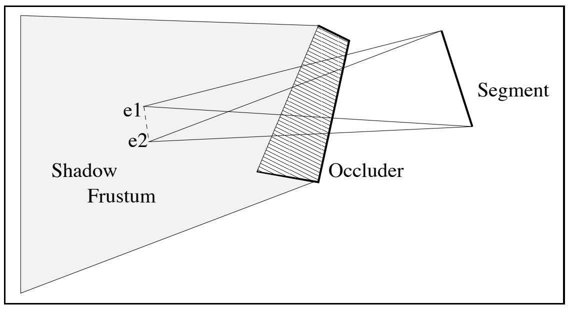


Figure 1: *The two viewpoints in the shadow frustum defined by the occluder and the occludee.*

from that ϵ-neighborhood, though it might be overestimated. The overhead of the overestimated primitives, however, must be small enough for an effective culling. Nevertheless, the server still needs to update the client's visible superset, but not necessarily in real-time, thus avoiding transmission latency. This relaxes the client-server loop, since the client can render the model autonomously for some frames and the server is free to serve other clients.

In this paper we present a fast algorithm for determining a conservative superset of an ϵ-neighborhood of a given viewpoint. Unlike [6, 7, 20, 8] we compute a conservative visibility superset which is valid for more than a single frame. The algorithm combines the visibility information from a few viewpoints to define the conservative superset which is valid for any viewpoint within the convex combination of the viewpoints. This can be regarded as a coarse "convex interpolation" of the visibility.

Previous work

Visibility culling algorithms attempt to avoid rendering objects that cannot be visible in the image. This approach was first investigated by Clark [4], who used an object hierarchy to rapidly cull entire groups of objects that lie outside the viewing frustum. This technique is most effective when only a small part of the virtual world falls within the view frustum at any single frame. Slater and Chrysanthou [15] used a view volume representing the view frustum and frame-to-frame coherence to quickly cull away primitives totally outside the volume. In a complex environment enough geometry remains inside the view frustum to overload the graphics pipeline, and additional acceleration techniques are required.

Airey et al. [1] and Teller [17] described methods for interactive walkthroughs of complex buildings that compute the potentially visible set of surfaces for each room in a building. These methods take advantage of the inherent property of in-door scenes which partition the space into "cells" and "portals". Then cell-to-cell and cell-to-object visibility can be precomputed. Only the potentially visible set of surfaces for the room currently containing the viewpoint needs to be rendered at each frame. Such methods are very effective for building interiors, but are not suited for other types of virtual worlds.

Hudson et al. [8] proposed an occlusion culling algorithm based on the shadow frusta of a list of potential occluders determined in a preprocess stage. Each frustum serves as a mask used to cull away any primitive completely enclosed within it. The remaining set of primitives is a superset of the set of visible primitives. That set is valid only for the current position of the viewpoint. Coorg and Teller [6, 7] used a similar technique with a different visibility test for occlusion culling. It is based on a set of separating and supporting planes between the occluder's and the occludee's endpoints with respect to the current viewpoint. In every frame it checks whether the viewpoint crosses one of the planes and incrementally updates the set of potentially visible primitives.

To avoid latency, Funkhouser [10, 9] shows how the potentially visible primitives can be pre-fetched to maintain a conservative cache that never fails. Although this work is motivated by the need to avoid disk-to-memory delays, it can be applied to avoid server-to-client delays for network-based applications. However, his method is based on the "cells and portals" techniques which are effective for in-door walkthroughs only.

Other related work deals with shadow algorithms [3]. Regarding the viewpoint and its ϵ-neighborhood as an area light source, occlusion culling algorithms search for the primitives which are in the umbra. However, shadow algorithms are usually analytical and much too expensive to simply cull away the primitives in the umbra. Another class of occlusion culling techniques is based on a hierarchical structure [16, 11, 20], which culls away hidden octree nodes and their underlying subtrees. These methods are, however, equivalent to culling the primitives in the umbra of a point light source, and are used to accelerate the rendering process of a single frame.

The culling algorithm that we introduce in this paper is geared towards out-door walkthroughs where no cells and portals can be easily defined, but still, the occlusion is rather dense. For example, a walkthrough in the streets of a city or small village [14]. Our technique is based on a fairly simple ray shooting technique and thus can be optimized by well-known techniques to yield a rapid and effective occlusion culling mechanism.

Outline of the approach

In the following we first present a 2D solution for the ϵ-visibility problem, and then show its extension to 3D. A *scene* consists of a set of convex objects and a viewpoint in a plane, where each object consists of a set of segments. Given a viewpoint, the visibility of each segment is determined, where the convex objects are the potential occluders. A segment is hidden by a single occluder if both of its endpoints are occluded from the viewpoint by that same occluder. Given two viewpoints e_1 and e_2 (see Figure 1), then if from both viewpoints a segment is occluded by a single convex occluder, the entire segment is also hidden from any point lying along the line connecting e_1 and e_2. This is extended to three (or more) viewpoints in the plane, yielding that a segment occluded by a single convex object from three (or more) viewpoints, is necessarily hidden from any point in the convex hull of the three (or more) viewpoints.

Regarding the hidden segment as an illuminator, then its endpoints together with the occluder silhouettes define a *shadow frustum*. If the occluders are convex (convex in 3D or just connected in 2D), then the shadow frustum is convex. Thus, if a set of viewpoints are all in a convex shadow frustum, so is any convex combination of them. The convex hull defined by the set of viewpoints is the ϵ-neighborhood for which we construct the visibility superset.

Based on the above observation, segments which are occluded by a single convex object are culled. Note that this does not cull away all the invisible segments since some segments are occluded by more than one object. This is true even for a single viewpoint [6]. However, we assume that the viewpoints are close enough and the visibility is temporally coherent. Indeed, as we shall show later this culling technique is effective for many out-door scenes encountered in practice.

Consider a given segment π with endpoints denoted by p_1 and p_2, and a given viewpoint e_1. Let $p_1 - e_1$ and $p_2 - e_1$ denote the line-of-sight between e_1 and p_1 and p_2, respectively. Let S_{e_1,p_1} and S_{e_1,p_2} be the lists of objects intersected by the endpoint line-of-sight, respectively. If the intersection of both lists is not empty,

$$S_{e_1,p_1} \cap S_{e_1,p_2} \neq \emptyset,$$

then π is hidden from e_1. This is the *conservative visibility* test from a single viewpoint. Given three viewpoints e_1, e_2 and e_3, if the intersection of the six lists is not empty,

$$S_{e_1,p_1} \cap S_{e_1,p_2} \cap S_{e_2,p_1} \cap S_{e_2,p_2} \cap S_{e_3,p_1} \cap S_{e_3,p_2} \neq \emptyset,$$

then π is hidden from any point within the triangle e_1, e_2, e_3. This is a *conservative ϵ-visibility* test from an ϵ-neighborhood defined by e_1, e_2 and e_3. The visibility superset is constructed by testing each segment in the scene against three viewpoints defined on-line according to the walkthrough. Each segment detected as *ϵ-invisible* is culled away from the potential visible set. Note that to increase the temporal visibility coherence, the occludees are individual segments, while the occluders are objects consisting of many segments.

The extension to 3D is straightforward. The ϵ-neighborhood is a tetrahedron defined by four vertices. The occluders are convex objects, and the occludees are triangular polygons (but not necessarily). Here again, the shadow frusta (umbra) created by casting light from a triangle and that emanates from a convex shape, is convex, and if a set of points are in the convex shadow frusta so does their convex hull. That is, if the four corners of the tetrahedron are in the umbra so too is any point within the tetrahedron. This requires casting 12 rays; three rays from each of the four vertices towards the three vertices of the triangle. Each ray detects the set of objects that intersect the line-of-sight connecting the tetrahedron-corner and the triangle-vertex. If the intersection of the 12 sets is not empty then there is at least a single occluder which occludes the triangle from any viewpoint within the tetrahedron.

Ray Shooting

The expensive calculation is the intersection of a line-of-sight with the potential occluding objects. Assuming that the scene consists of m convex objects (the occluders), and n triangles (the occludees), $n >> m$, then a naive algorithm requires $12 \times n \times m$ intersection operations for each tetrahedron-neighborhood. Note, that m is much smaller than n. However, the main advantage of our technique is that it is based on a simple mechanism of ray shooting, and the culling process can be significantly accelerated by well-known techniques developed for ray-tracing.

Hierarchical space-subdivision schemes like octrees, BSP-trees, or Kd-trees, partition the space into cells with a reduced complexity. Although the cell complexity is theoretically unbounded, except for extremely degenerated cases, a cell can contain no more than, say, three objects. Using a space-subdivision reduces the number of intersections per ray. A single ray does not need to intersect all the m objects, but by incrementally traversing through the cells along the ray, only a fraction of the objects needs to be intersected. Roughly speaking, the number of intersections is in the order of $\sqrt{m}$, which is supported by our experimental results. Note that unlike ray tracing where the traversal is stopped at the first hit, here we need to continue the traversal and detect all the intersections along the ray.

The hierarchical structure also offers a top-down traversal of the space. Instead of testing the visibility of individual triangles, it is possible to test the visibility of the space-subdivision cells and to quickly cull all the triangles contained in the occluded cells. This idea is appealing at first, but practically whenever the size of the occludee is larger than of the occluder, the algorithm does not cull many objects [19].

Better results are achieved by enclosing each occludee in a bounding box and testing first the visibility of the bounding boxes. The number of bounding boxes, denoted by N, is much smaller than of the triangles ($N << n$), and is closer to the number of occluders (m). Empirical statistics are reported in Section .

Another significant acceleration can reduce the constant number of 12 rays per tetrahedron. Since the intersection of the 12 sets is required, then except for the first ray, for the other 11 rays it is enough to test their visibility only against the occluders reported by the first ray. That is, first a ray is shot which incrementally creates a set O of all the potential occluders of a given primitive with respect to the tetrahedron. The next ray does not need to traverse the space, but just to incrementally test its intersection with the objects in O. This leaves in O only the set of objects intersected by all the rays cast so far. Thus, the size of O is quickly reduced, so that most of the rays actually test their intersection with a very small number of objects without traversing the space. This suggests that it pays off to have a high resolution partition since the overhead of the traversal is covered by just one ray per tetrahedron. As in [7, 8] it is also possible to use only "large occluders" defined by a solid-angle function. This strategy assumes that a small list of effective occluders is sufficient to cull away most of the redundant primitives.

The above also suggests that it would not cost more to use cubes instead of tetrahedra. Theoretically, a naive implementation would have required shooting 24 rays from the cube's eight corners towards the three vertices of the triangle. However, in the above incremental computation of the intersections the addition of more rays is not significantly more expensive. On the other hand, it is much easier to deal with cubes, especially regarding the location of the current viewpoint. Moreover, it is enough to use only the subset of the rays since rays which do not lie on the convex hull defined by the vertices of the ϵ-cell and the vertices of the candidate are redundant [19].

For some applications it is also possible to build the ϵ-visibility cells in a coarse-to-fine fashion. The coarse culling creates large cells. Then, the fine culling refines these large cells into subcells with less effort, since most of the primitives have already been removed.

It is possible to exploit the z-buffer hardware as a fast visibility primitive operation by which the visible occluders can be rapidly detected. Considering only the visible occluders yields a more conservative visibility test. Given a primitive and four viewpoints, if all the lines-of-sight agree on the closest occluder, then the primitive is hidden from the convex hull defined by the four viewpoints. This *super-conservative visibility test* culls only a subset of the primitives culled by the conservative visibility test. However, whenever the ϵ-neighborhood is small enough, the overestimation of the super-conservative visibility test is small. Yet, this coarser mechanism can be employed in the coarser levels of a hierarchical computation of the ϵ-visibility cells, since the finer resolutions will be refined anyhow.

Results

We have implemented the algorithm for the conservative ϵ-visibility described in the previous section on an SGI R4400 machine. The main parameter by which we can measure the performance of the visibility algorithm is the effectiveness of the culling. Since the results are scene dependent, we have generated different scenes consisting of a number of simple objects distributed randomly. The results are based on two typical urban models consisting of buildings distributed evenly and randomly in a plane (as in Figure 2). The "city1" model consists of 21280 polygons which create a city of 2128 buildings of two types. Most buildings are simple boxes and about 25% are made of three boxes of random height placed one above the another.

Since in large dense models most of the polygons are hidden, we have tested another typical model with a more sparse distribution of buildings in the city ("city2"). Both models have the same number of polygons.

The ϵ-neighborhood used in the experiments is a box at ground level which has been located randomly in the above static scenes (see Figure 3). Note that in the above figures the models are rendered from the outside to get a better view of the model and the culling effect. However, in the experiments, the ϵ-neighborhoods were located inside the models, where the viewer is surrounded by the objects as in the walkthrough. The size of the ϵ-neighborhood we tested was one fourth the size of the typical object in the models.

The occlusion results depend on the location of the ϵ-neighborhood in the model. Table 1 shows the average occlusion results of the tested models. Besides the actual occlusion results, given in rows 1-4, we show in row 5 the number of boxes that were occluded by the first potential occluder tested. As expected, we get more occlusions in the dense model. This can be seen both in the bounding box occlusion stage and in the polygon occlusion. The results show that in most cases the first potential occluder encountered is the actual occluder of the bounding box. This gives basic support for using the z-buffer hardware for occlusion culling.

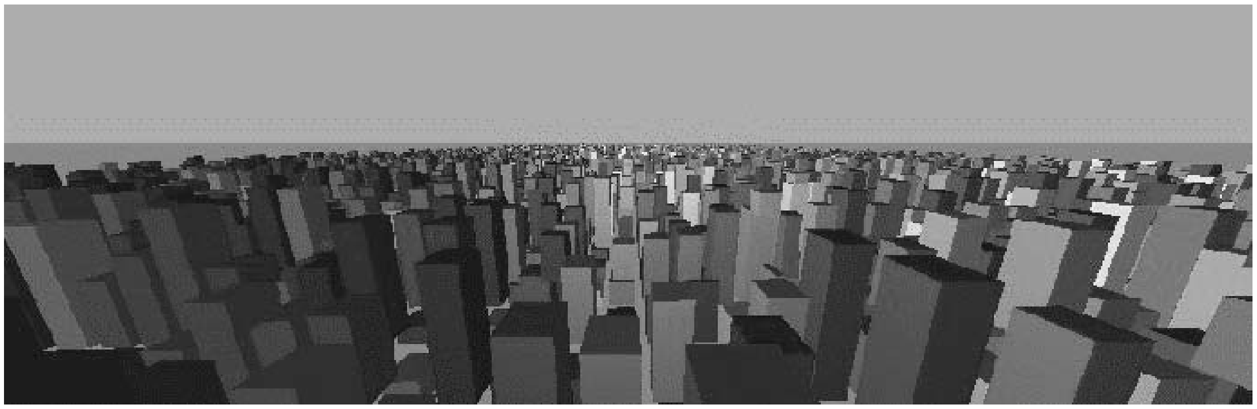


Figure 2: An overview of the city model.

Table 1: *Polygon and Box Culling.*

model	city 1	city 2
total polygons	21280	21280
occluded polygons	20062	18713
total bounding boxes	2128	2128
occluded boxes	1954	1755
first occluders	1427	1225

We tested these models with several optimization techniques described above. The first column in Table 2 shows the results of the most naive and basic technique with no accelerating optimizations. The second column shows the effectiveness of using bounding box occlusion prior to polygon occlusion. In this case only the polygons which were not included in occluded boxes are tested. The third column includes the fact that occluded objects are redundant occluders. The fourth column shows the results of eliminating redundant rays. These are the rays which are not on the convex hull created by the ϵ-neighborhood and the tested box. The last column combines all these techniques showing the results of a fully optimized algorithm. The results are shown in two parts. The first part gives the results of the box occlusion stage. The second part shows the results for culling polygons which were not occluded in the first stage.

The first row shows the average total time required for the ϵ-visibility test. The second row gives the time used for box occlusion routines, and the sixth row the time used for polygon occlusion. Rows 3-5 give a breakdown of the box occlusion time and rows 7 and 8 the corresponding values for the polygon occlusion. In the second part of the table, rows 7 and 8 show the actual number of intersections performed in each stage of the calculation.

It can be seen that bounding box occlusion significantly reduces the time used for calculating the visibility set. Excluding occluded objects as potential occluders yields only a minor improvement, but does not require any additional computations. The redundant ray optimization requires initial calculations to exclude the redundant rays, but still was found effective since the total time used for box occlusion is reduced by 20%-25%. As can be seen from Table 2, using all the optimizations reduces the total time by 76%. The majority of the time in each stage of the visibility set calculation is spent in the ray traversal and in the intersection routines. Intersection times are reduced by minimizing the intersection calls as can be seen in row 5 in table 2. The ray traversal time may be significantly accelerated by using the optimized technique of grid traversal [5].

Discussion

The ϵ-visibility can be combined with any other culling or levels-of-detail techniques. In particular view frustum culling seems vital. The occlusion visibility can be calculated assuming a 360 degree field of view, and the frustum culling can be added as a post process running either at the server or the client. Executing the frustum culling at the server is not necessarily the best option as it first seems. Executing the frustum culling at the client yields a stable superset, since it is invariant to the viewer (camera) orientation. On the other hand, the superset, and consequently the initial transmission, is larger. Although the size of the initial transmission is not that large, it can still be transmitted progressively (lazy evaluation) during the very first frames of the walkthrough.

Table 2: *Results using various optimizations (times are given in milliseconds).*

	no optimization	bounding box	redundant occluders	redundant rays	fully optimized
total time	30170	8900	8850	7040	7030
box occlusion time	-	7450	7340	5540	5550
ray traversal time	-	3460	3530	3460	3420
ray exclusion time	-	-	-	640	630
box intersection time	-	2350	2310	880	850
polygon occlusion time	29820	1410	1400	1410	1390
ray traversal time	24070	940	970	990	850
polygon intersection time	4000	380	290	280	400
ray/box intersections	-	135278	135089	45570	42414
ray/polygon intersections	517800	38648	38648	38648	38648

As mentioned above, our method requires the occluders to be convex as in [6, 7, 8]. However, in practice, most objects are not convex, but one can decompose a general polyhedral into convex parts [2]. The decomposition should not be exact; however, the aggregation of the convex parts must be included in the original object. A simple approximated decomposition can be achieved by an octree, where only the coarse levels are used as the occluders. Another possible convex decomposition is to use the individual triangles as the convex occluders. But these occluders might be too small to be effective. Currently, we assume that the occluders are built as part of the model. We are now investigating several techniques by which an object can be represented by a union of convex shapes. The goal is to automatically find the union of a small number of convex bodies, which on one hand can be intersected easily, and on the other, their union is as large as possible, but still included in the interior of the original object.

Yet another possible way to deal with non-convex objects is to implicitly test the intersection of the tetrahedra (defined by the convex hull of the viewpoints) and the "shadow" volume defined by the occludee and the occluder silhouettes (as in [8]). This test is more expensive but may be effective for very large and complex occluders.

Based on the presented method, we are currently implementing a client-server walkthrough system for the internet. On the server side, a C program precomputes the visibility for each cell. For the run-time part, both the client and the server are written in JAVA using TCP/IP protocol. The client is an applet loaded into the Netscape Navigator browser. For rendering we are currently considering using WorldView VRML 2.0 plugin of Intervista, which includes an External Authoring Interface (EAI) for dynamic update of the scene.

Acknowledgments
This work was supported in part by a grant from the Israeli Ministry of Science and the French Ministry of Research and Technology (AFIRST). The authors would like to thank Dan Halperin for his encouragement and helpful comments, and Shuly Lev-Yehody for assisting them with the implementation.

References

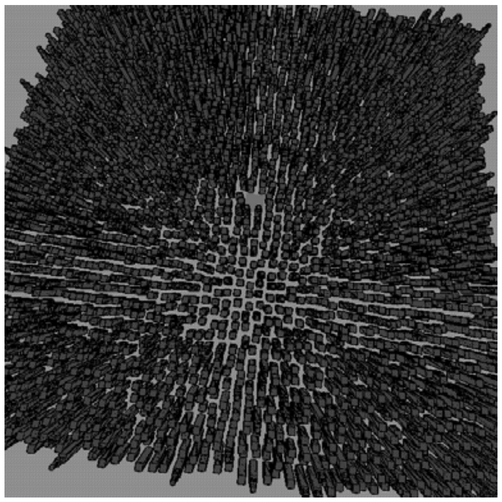
[1] J.M. Airey, J.H. Rohlf, and Jr. F.P. Brooks. Towards image realism with interactive update rates in complex. In *1990 Symposium on Interactive 3D Graphics*, volume 24(2), pages 41–50, 1990.

[2] B. Chazelle, D. Dobkin, N. Shouraboura, and A. Tal. Strategies for polyhedral surface decomposition: An experimental study. *Computational Geometry: Theory and Applications*, 7(4-5):327–342, 1997.

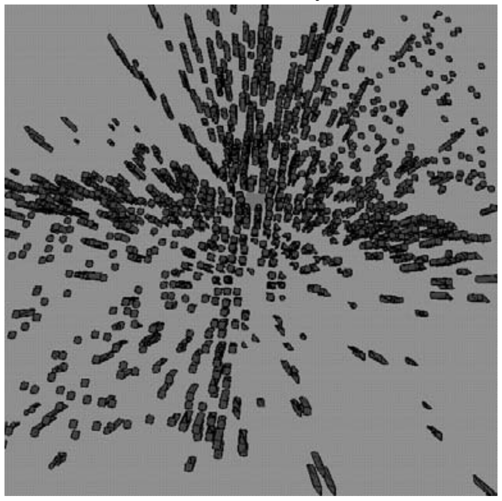
[3] Y. Chrysanthou. *Shadow Computation for 3D Interactive and Animation*. PhD thesis, Department of Computer Science, College University of London, January 1996.

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

[5] D. Cohen and A. Shaked. Photo-realistic imaging of digital terrains. In *Proceedings of Eurographics*, volume 12(3), pages 363–373, September 1993.



The entire city



The potentially visible polygons

Figure 3: Two birds-eye views of the city.

[6] S. Coorg and S. Teller. Temporally coherent conservative visibility. In *Proceedings of 12th ACM Symposium on Computational Geometry*, 1996.

[7] S. Coorg and S. Teller. Real-time occlusion culling for models with large occluders. In *Proceedings of 1997 Symposium on Interactive Graphics*, 1997.

[8] T. Hudson et al. Accelerated occlusion culling using shadow volumes. In *Proceedings of 13th ACM Symposium on Computational Geometry*, Nice, France, June 4–6 1997.

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

[10] T.A. Funkhouser. Ring: A client-server system for multi-user virtual environments. In *Computer Graphics (1995 SIGGRAPH Symposium on Interactive 3D Graphics)*, pages 85–92, April, 1995.

[11] N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. In *Proceedings of ACM*, pages 231–238. Siggraph, 1993.

[12] Y. Mann and D. Cohen-Or. Selective pixel transmission for navigating in remote virtual environments. *Computer Graphics Forum*, 16(3):201–206, 1997.

[13] D. Schmalstieg and M. Gervautz. Demand-driven geometry transmission for distributed virtual environment. In *Computer Graphics Forum*, volume 15(3), pages 421–432, 1996.

[14] F. Sillion, G. Drettakis, and B. Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery. In *Proceedings of Eurographics*, September 1997.

[15] M. Slater and Y. Chrysanthou. View volume culling using a probabilistic caching scheme. In *ACM VRST'97*, Lausanne, 1997.

[16] O. Sudarsky and C. Gotsman. Output-sensitive visibility algorithms for dynamic scenes with applications to virtual reality. *Computer Graphics Forum*, 15(3):249–258, 1996.

[17] S. Teller and C.H. Sequin. Visibility preprocessing for interactive walkthrough. In *Proceedings of ACM*, pages 61–69. Siggraph, 1991.

[18] R. Yagel and W. Ray. Visibility computations for efficient walkthrough of complex environments. *Presence*, 5(1):1–6, 1996.

[19] E. Zadicario. Conservative visibility streaming of densely occluded scenes. Master's thesis, Tel-Aviv University, School of Mathematical Sciences, 1998.

[20] H. Zhang, D. Manocha, T. Hudson, and K. Hoff. Visibility culling using hierarchical occlusion maps. In *Proceedings of ACM*. Siggraph, 1997.