Visibility Preprocessing with Occluder Fusion for Urban Walkthroughs

Peter Wonka, Michael Wimmer, Dieter Schmalstieg

Vienna University of Technology

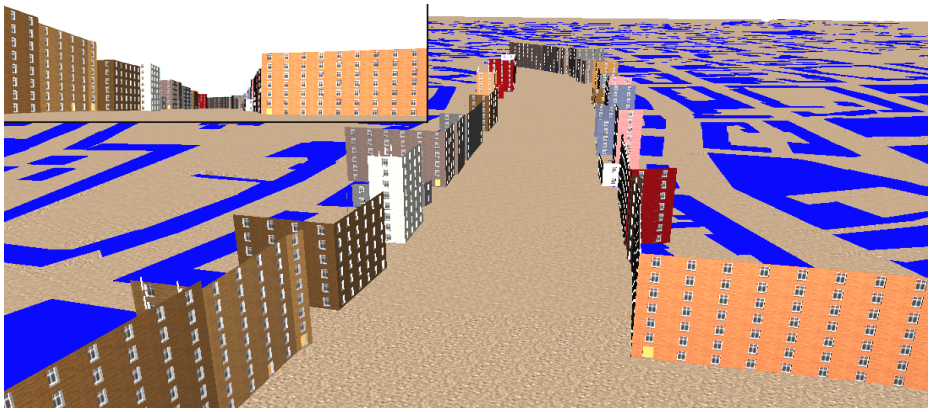

Figure 1. Views from the 8 million polygon model of the city of Vienna used in our walkthrough application. The inset in the upper left corner shows a typical wide open view, while the large image shows the portion of the scene rendered after occlusion culling. Note how occlusion fusion from about 200 visible occluders allows to prune over 99% of the scene.

Abstract. This paper presents an algorithm for occlusion culling from regions of space. It is conservative and able to find all significant occlusion. It discretizes the scene into view cells, for which cell-to-object visibility is precomputed, making on-line overhead negligible. Unlike other precomputation methods for view cells, it is able to conservatively compute all forms of occluder interaction for an arbitrary number of occluders. We describe an application of this algorithm to urban environments. A walkthrough system running an 8 million polygon model of the city of Vienna on consumer-level hardware illustrates our results.

Keywords. Visibility determination, occlusion culling, occluder fusion, urban walkthroughs

1 Introduction

Applications of visual simulation, such as architectural walkthroughs, driving simulators, computer games, or virtual reality require sufficiently high update rates for interactive feedback. While update rates starting from 10 frames per second (fps) are often called interactive, they are rarely considered satisfactory. To avoid temporal aliasing, true real-time behavior requires update rates equivalent to the monitor refresh rate, i. e. 60 Hz or more.

For very large databases, this kind of performance cannot be achieved with hardware acceleration alone, but must be assisted by algorithms that reduce the geometry to be rendered to a tolerable amount. A popular means is occlusion culling, which quickly prunes large portions of the scene. Visibility calculations for a *single viewpoint* have to be carried out online and infer significant runtime overhead. Moreover, in a typical single-CPU system, online processing time must be shared with

other tasks, such as rendering and collision detection. Therefore it is useful to calculate visibility for a region of space (*view cell*) in a preprocessing step.

1.1 Motivation

Occlusion culling shares many aspects with shadow rendering. Occlusion culling from a view cell is equivalent to finding those objects which are completely contained in the umbra (shadow volume) with respect to a given area light source. In contrast to occlusion from a point, exact occlusion culling for regions in the general case (or its equivalent, shadow computation for area light sources) is not a fully solved problem. Two main problems impede a practical closed-form solution:

- The umbra with respect to a polygonal area light source is not only bounded by planes, but also by reguli, i. e. ruled quadratic surfaces of negative Gaussian curvature [4][21]. Such reguli are difficult to store, intersect etc.
- The maximum number of topologically distinct viewing regions constitutes the so called *aspect graph* [7][14], which is costly to compute ($O(n^9)$ time [7]). For applications such as radiosity, a more practical approach is the *visibility skeleton* [5]. However, the algorithmic complexity and robustness problems of analytic visibility methods impede their practical use for large scenes.

For visibility from a point, the joint umbra of many occluders is the union of the umbrae of the individual occluders, which is simple to compute. In contrast, for view cells, the union of umbrae is only contained in the exact umbra, which also depends to a great extent on contributions of merged penumbrae (see Figure 2). While umbra information for each occluder can be described by a simple volume in space ('in/out'-classification for each point), penumbra information also has to encode the visible part of the light source, making it hard to find a practical spatial representation. Therefore a general union operation for penumbrae is not easily defined.

Although an approximation of the umbra from multiple occluders by a union of individual umbrae is conservative, it is not sufficiently accurate. There are frequent cases where a significant portion of occlusion coming from occluder fusion is missed, making the solution useless for practical purposes. In particular, we distinguish between the cases of connected occluders, overlapping umbrae, and overlapping penumbrae (see Figure 2 (a), (b) and (c) respectively).

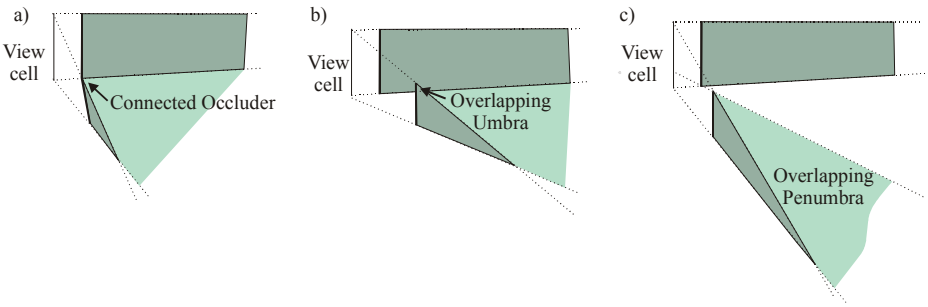

Figure 2. Different types of occluder interactions (illustrated in 2D). Individual umbrae (shaded dark) cover only a small area, while the occluders jointly cover a large unbounded area (shaded light).

This paper describes a fast, conservative occlusion culling algorithm that solves the aforementioned problems. It is based on the idea that conservative visibility for a region of space can be calculated by *shrinking* occluding objects and *sampling* visibility from points on the boundary of the region. Thus, conservative visibility for each cell can be computed as a preprocess, and visibility determination consumes no time during actual walkthroughs. This approach has several essential advantages:

Occluder Fusion: Our technique places no inherent restriction on the type of occluder interaction. Fusion of occluder umbrae[1] and even penumbrae is implicitly performed. This includes the hard case where individual umbrae of two occluders are disjoint (see Figure 2 (c)). We also discretize umbra boundaries, which are typically made up of reguli (curved surfaces). Other published algorithms do not currently handle such cases of occluder interactions.

Conservativity. Our method never reports visible objects as invisible. While some authors argue that non-conservative (approximate) visibility *can* be beneficial, it is not tolerable for all applications.

Note that while occlusion culling can typically speed up rendering by orders of magnitude, no visibility algorithm alone can guarantee sufficiently high frame rates. If the complexity of visible objects is too high, occlusion culling has to lay a good foundation for further simplification techniques like level of detail and image-based rendering.

1.2 Related Work

Several methods were proposed to speed up the rendering of interactive walkthrough applications. General optimizations are implemented by rendering toolkits like Performer [15] that aim for an optimal usage of hardware resources. Level of detail (LOD) algorithms [9] are very popular in urban simulation [11], because they do not need a lot of calculation during runtime. With image-based simplification, coherent parts of the scene are replaced by impostors (textured quadriliterals [17][18] or textured depth meshes [19]).

Occlusion culling algorithms calculate a conservative estimation of those parts of the scene that are definitely invisible. Final hidden surface removal is usually done with a z-buffer. A simple and general culling method is view frustum culling, which is applicable for almost any model.

Occlusion culling from a point can be calculated in image space [8][25] or geometrically [1][3][10]. The implementation presented in this paper makes use of graphics hardware to calculate occlusion. We use the concept of *cull maps*, which have recently been proposed by Wonka and Schmalstieg [23] for online occlusion culling of city-like scenes.

To calculate occlusion for regions, a general method is to break down the view space into cells and precompute for each cell a set of objects that are potentially visible (potentially visible sets, PVS). For general scenes, visibility precomputation can

[1] The umbra with respect to a view cell is the region of space from where no point of the view cell can be seen, whereas the penumbra is the region of space from where some, but not all points of the view cell can be seen.

become quite costly with respect to time and memory, but for certain scenes, like terrains [20] or building interiors [22] it is possible to use the a priori knowledge about the scene structure for visibility calculation. Cohen-Or et al. [2] show an algorithm for densely occluded scenes that does not make use of occluder fusion.

Several view cell visibility methods with occluder fusion have only recently been proposed. Durand et al. [6] position six planes around a view cell and project occluders on those planes. To calculate conservative occlusion they define an extended projection for occluders and occludees. Schaufler et al. [16] perform occluder fusion by extending occluders in an octree into octree nodes already found occluded. Koltun et al. [12] calculate large cross sections through the shadow volume from a view cell and use them as virtual occluders during online occlusion culling.

1.3 Organization of the paper

The remainder of the paper is organized as follows. Section 2 introduces the main idea of how to calculate occluder fusion for the region visibility problem. Section 3 describes how we apply the algorithm to urban environments. Results are shown in section 4, and section 5 contains a discussion of our algorithm in comparison with other methods. Section 6 concludes the paper and shows avenues of future research.

2 Occluder Fusion

This section explains the main idea of our algorithm: visibility can be calculated fast and efficiently for point samples. We use such point samples to calculate visibility for a *region* of space. To obtain a conservative solution, occluders have to be shrunk by an amount determined by the density of point samples.

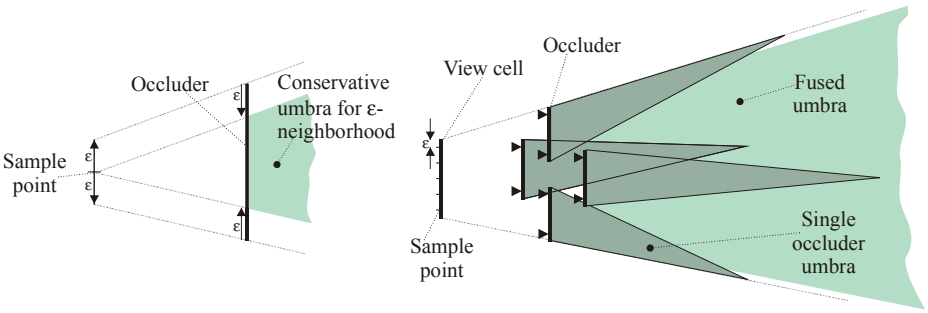

Figure 3. *Left*: Occluder shrinking: By considering an occluder after shrinking it by ε, the umbra from a point sample provides a good conservative approximation of the umbra of the original occluder in a neighborhood of radius ε of the sample point. *Right*: The fused umbra from the 5 point samples shown in Figure 4 is the intersection of the individual umbrae (shaded light). It is here compared to the union of umbrae from the original view cell (shaded dark). As can be seen, point sampling computes superior occlusion and only slightly underestimates exact occlusion.

2.1 Occluder fusion by occluder shrinking and point sampling

As can be concluded from the examples in Figure 2, occluder fusion is essential for good occlusion culling, but difficult to compute for view cells directly. To incorporate

the effects of occluder fusion, we present a much simpler operation, which can be constructed from point sampling (for clarity, our figures show simple 2D cases only).

Our method is 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. An approximation of actual visibility can be obtained by computing the intersection of all sample points' umbrae. This approach might falsely classify objects as occluded because there may be viewing positions between the sample points from 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 (see Figure 3, left side).

Consequently, a point sample used together with a shrunk occluder is a conservative approximation for a small area with radius ε centered at the sample point. If the boundary of the original view cell is covered with sample points so that every point on its boundary is contained in an ε-neighborhood of at least one sample point, an object lying in the intersection of the umbrae from all sample points is therefore occluded for the original view cell (including its interior).

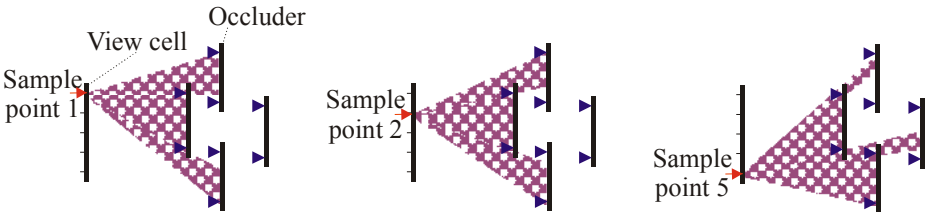

Figure 4. When performing point sampling for occlusion after occluders have been shrunk (as indicated by small triangles), all four occluders can be considered simultaneously, cooperatively blocking the view (indicated by the shaded area) from all sample points.

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 can be performed (see Figure 4 and Figure 3, right side) using two main approaches:

- If a PVS is available for every sample point of a view cell, the PVS for the view cell can be calculated as the union of all individual PVS (this approach is independent of the algorithm used to calculate point visibility).
- If umbra information is stored explicitly as a volume during the point visibility algorithm, a joint umbra volume for the view cell can be calculated by intersecting the umbra volumes of all sample points (this approach might be more efficient, depending on the data structure used in the point visibility algorithm). The PVS for the view cell is then obtained by testing objects against this joint umbra.

While the method is conservative and not exact in that it underestimates occlusion, the fact that an arbitrary number of occluders can cooperate to provide occlusion helps to find relevant occlusion as long as ε is small in relation to a typical occluder.

In general, the exact amount by which a *planar* occluder (e.g., a polygon) has to be shrunk in each direction is dependent on the relative positions of sample point and occluder. If we consider a *volumetric* occluder, however, it can be shown that shrinking this volumetric occluder by ε provides a correct solution for an arbitrary volumetric view cell (see the appendix for a formal proof of this fact). Therefore, the occlusion culling problem can be solved using occluder shrinking and point sampling.

2.2 Implications of occluder shrinking

Since occluder shrinking need only be applied once during preprocessing for each occluder, the actual visibility computation is efficient for each view cell.

Note that while the algorithm apparently discretizes the view cell, it actually discretizes the complicated occluder interactions (Figure 2), i.e., including merged penumbrae and umbra boundaries defined by reguli. This principle makes it possible to compute occlusion from simple and fast point sampling.

2.3 Algorithm Overview

In the following, we outline the preprocessing algorithm:

1. Subdivide environment into view cells, choose ε.
2. Shrink occluders to yield conservative visibility with respect to ε.
3. For each view cell:
    4. Determine a sufficient number of sample points for that view cell.
    5. Calculate visibility for each sample point.
    6. Calculate view cell visibility (merge PVS or intersect umbra volumes of sample points)

3 Application to Urban Environments

Using the occluder shrinking principle explained in the last section requires a great amount of point sampling. Our implementation builds on the occluder shadow work explained in [23] and thus operates on city-like scenes (2.5D). It makes use of graphics hardware, which allows for fast calculation of individual point samples.

Because of the 2.5D property of occluders, any object found visible from one particular point is also visible from all locations above this point. This implies that sample points need only be placed on the edges of the top of the view cells.

While this type of environment may appear restrictive, it is suitable for the applications we have in mind (urban walkthroughs, driving simulation, games) and typically provides a large amount of occlusion. We made the following choices in our implementation:

3.1 Subdivision into View Cells

For our system we chose to use a constrained Delaunay triangulation of free space. The actual view cells are found by erecting a prism above each triangle of the triangulation.

3.2 Occluder Shrinking

In 2.5D, occluders can be arbitrary discontinuous functions $z = f(x,y)$ with compact support. Occluders must be given as a triangular irregular network, with heights assigned to the vertices of the triangles and possibly with discontinuities. We use a simple geometric algorithm to shrink occluders by the required amount.

3.3 Occluder selection

Typical models of urban environments consist of a large number of small triangles. Instead of using those triangles directly, our modeling system automatically extracts and stores building facades as occluders, a feature that cannot be assumed to be available in a general application. Occluders obtained this way correspond to a subset of the visual hull (i.e., the maximal object that has the same silhouettes and therefore the same occlusion characteristics as the original object, as viewed from all view cells [13]). Therefore, we do not need to consider irregular features of the facade like doors or windows. Our system handles arbitrary 2.5D-occluders, i.e., heightfields. The footprint of the occluder may be (and usually is) concave.

3.4 Optimizations

To accelerate preprocessing for urban environments, we extended the frame buffer algorithm from [23] with two major extensions (see [24] for details concerning implementation issues):

1. The point visibility algorithm explicitly stores umbra information in a frame buffer section. We use stencil buffer operations available on current graphics hardware to incrementally intersect umbra volumes from all sample points of a view cell. Only the joint umbra needs to be read back from the frame buffer.
2. For rapid preprocessing of large scenes, rasterizing *all* occluders to obtain a close approximation of exact visibility is not feasible. Quick rejection of large portions of the scene including the contained *occluded occluders* is essential. To achieve this goal, we employ a hierarchical approach.

4 Implementation and Results

A walkthrough system using the techniques outlined in this paper was implemented in C++ and OpenGL. All tests reported here were run under Windows NT on a Pentium-III 650MHz with 1GB RAM and a GeForce 256 graphics accelerator.

4.1 Preprocessing

A model of the city of Vienna with 7,928,519 polygons was used throughout testing. It was created by extruding about 2,000 building block footprints from an elevation map

of Vienna, and procedurally adding façade details such as windows and doors (Figure 7, also on the color plate). One building block consists of one up to ten connected buildings. Note that each building block was modeled with several thousand polygons (3900 on average). Building blocks were further broken down into smaller objects (each consisting of a few hundred polygons) which were used as primitives for occlusion culling.

82,300 view cells were considered. The sample spacing constant ε was set to 1m, which led to the use of 6-8 sample points per view cell edge on average. Preprocessing took 523 minutes. 54 % of the preprocessing time was spent on reading back umbra information from the frame buffer.

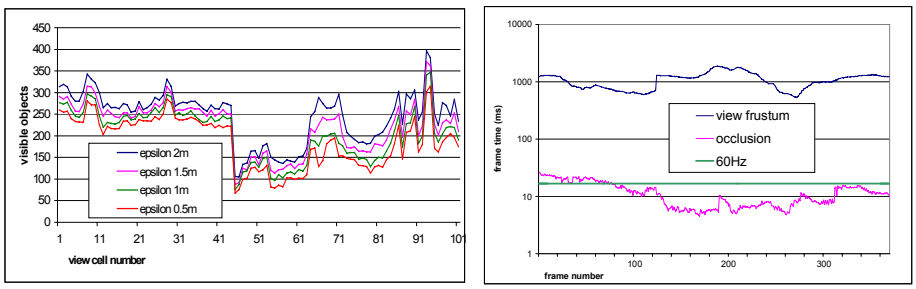

Figure 5. *Left*: Influence of various values of sample spacing (ε) on occlusion quality. Plotted is the number of visible objects (y-axis) against the view cells visited in the test walkthrough. *Right*: Frame times of occlusion culling vs. view frustum culling for the walkthrough. Note that frame times are plotted on a log scale to fit two orders of magnitude into the diagram.

4.2 Quality

We experimented with different settings for sample spacing (ε). Figure 5 (left side) shows the results for various choices of ε. As can be seen, larger values of ε also yield acceptable occlusion, allowing to handle larger models. On average, our algorithm identifies 99.34% (~1:150) of the scene as occluded. While these numbers are representative for a real-world data set, arbitrary occlusion ratios can be generated by increasing the depth complexity of the model. More interesting is the fact that the absolute numbers of occluders that contribute to occlusion is about 200 on average for our test model, which is quite high. See section 5.2 for a discussion of this fact. Figure 1 demonstrates the effect of occluder fusion.

4.3 Real-time rendering

To assess real-time rendering performance, precomputed occlusion was used to speed up rendering of a prerecorded walkthrough of the model, which was 372 frames long and visited 204 view cells. The model consumed approximately 700 MB and was fully loaded into main memory before the beginning of the walkthrough. Occlusion information (object IDs of potentially visible objects for each view cell) consumed 55 MB and was also fully loaded. Standard optimizations like triangle stripification, backface culling, compiled vertex arrays or display lists and static state sorting were used.

We measured frame times for two variants

- Full occlusion culling
- View frustum culling as a reference measurement

Occlusion culling provided an average speed-up of 93.78 over view frustum culling. Figure 5 (right side) shows the frame times from our walkthrough for occlusion culling vs. view frustum culling (logarithmic scale). Note that the desired 60Hz are achieved for the second part of the walkthrough (dense occlusion), while the first part (wide open view) would require further optimizations (e. g. impostors).

5 Discussion

5.1 Comparison

Competitive methods for view cell visibility have only recently been investigated independently by other authors. We are aware of two approaches: both require that an occluder intersects the accumulated umbra of previously considered occluders for occluder fusion to occur. Schaufler et al. [16] extend blockers into areas already found occluded, while Durand et al. [6] project occluders and occludees onto planes with new extended projections from volumetric view cells.

While these methods work in full 3D, they only consider a subset of occluder interactions handled by our method. Since their spatial data structure only represent *umbra* information, they cannot handle cases such as Figure 2 (c), for example, where the *penumbrae* of two occluders can be merged, even though there is no joint umbra. In most cases, missing such types of occluder interaction will cause a larger number of objects to appear in a PVS than necessary.

We found that complex shadow boundaries arise already in simple cases. Even in 2.5D, a simple occluder with non-convex footprint gives rise to so-called 'EEE event surfaces' [5], ruled quadric surfaces incident on three edges in the scene. Those event surfaces bound the umbra due to the occluder, but are usually not considered in other methods. Our approach discretizes such boundaries through point sampling, which gives a good approximation of the real umbra.

We believe that discretizaton is a reasonable trade-off between correctness and efficiency: our method implicitly handles all types of occluder interactions, and if we decrease ε, our method converges to a correct solution.

Although the idea of occluder shrinking and point sampling is applicable in 3D, the large number of point samples required would make a straightforward implementation rather slow.

5.2 Selection of occluders

Several algorithms achieve good results with a heuristic to select a set of occluders that is most likely to occlude a big part of the scene [3][10]. However, we have found that the number of occluders that contribute to occlusion is typically quite large (as indicated by results presented in section 4.1). Even missing small occluders can create holes that make additional objects visible. Therefore, all visible occluders should be considered for visibility calculations.

Using a large number of occluders requires significant computational effort. However, our preprocessing times are reasonable even for large scenes because the only geometric operation, occluder shrinking, needs to be performed only once during preprocessing. In our implementation, per view cell operations almost fully leverage the speed of current rasterization hardware, so we can calculate and render up to several hundred thousand occluder shadow polygons per second.

6 Conclusions and Future Work

Visibility preprocessing with occluder fusion is a new method for accelerated real-time rendering of very large urban environments. While it cannot compute exact visibility, no simplifying assumptions on the interaction between occluders or heuristics for occluder selection are necessary. Through point sampling, the proposed algorithm approximates actual umbra boundaries due to multiple occluders more exactly than previous methods, leading to better occlusion.

The measured number of occluders that contribute to occlusion (~200 on average) leads us to believe that simultaneous consideration of a large number of occluders is indeed crucial for achieving significant culling in hard cases where large open spaces are visible. The frame rates from our walkthrough, which are closely below or above the desired 60Hz, show that no significant time is available for on-line occlusion calculation, and that precomputed occlusion is necessary for true real-time performance.

Future work will focus on constructing suitable image-based representations for areas of the city where visibility preprocessing alone is not sufficient to guarantee a frame rate above 60 Hz. The restriction to volumetric occluders could be resolved with the introduction of view-dependent occluder simplification.

Acknowledgements

This research is supported by the Austrian Science Fund (FWF) contract no. p-13867-INF. The authors would like to thank Fredo Durand and Gernot Schaufler from MIT for fruitful discussions, Martin Held and Xinyu Xiang from Stony Brooks for their triangle stripper and triangulation code, Alan Murta from the University of Manchester for a polygon clipper, and Gerald Hummel from TU Vienna for support with the city model.

References

[1] J. Bittner, V. Havran, P. Slavík. Hierarchical Visibility Culling with Occlusion Trees. Computer Graphics International 1998 Proceedings, pp. 207-219, 1998.

[2] D. Cohen-Or, G. Fibich, D. Haperin, E. Zadicario. Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes. Computer Graphics Forum (Proceedings of EUROGRAPHICS'98), 17(3), pp. 243-253, 1998.

[3] S. Coorg, S. Teller. Real-Time Occlusion Culling for Models with Large Occluders. Proceedings of the Symposium on Interactive 3D Graphics, pp. 83-90, 1997.

[4] F. Durand. 3D Visibility. Analytical Study and Applications. PhD thesis, IMAGIS-GRAVIR/IMAG-INRIA, Grenoble, France, 1999.

[5] F. Durand, G. Drettakis, C. Puech. The Visibility Skeleton: A Powerful and Efficient Multi-Purpose Global Visibility Tool. SIGGRAPH 97 Conference Proceedings, pp. 89-100, 1997.

[6] F. Durand, G. Drettakis, J. Thollot, C. Puech. Conservative Visibility Preprocessing using Extended Projections. To appear in SIGGRAPH 2000 Conference Proceedings.

[7] Z. Gigus, J. Canny, R. Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(6), pp. 542-551, 1991.

[8] N. Greene, M. Kass. Hierarchical Z-Buffer Visibility, SIGGRAPH 93 Conference Proceedings, pp. 231-240, 1993.

[9] P. Heckbert, M. Garland. Survey of Polygonal Surface Simplification Algorithms. Multiresolution Surface Modeling (Course 25), SIGGRAPH, 1997.

[10] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, H. Zhang. Accelerated Occlusion Culling using Shadow Frusta. 13th International Annual Symposium on Computational Geometry (SCG-97), pp. 1-10, 1997.

[11] W. Jepson, R. Liggett, S. Friedman. An Environment for Real-time Urban Simulation. Proceedings of the Symposium on Interactive 3D Graphics, pp. 165-166, 1995.

[12] V. Koltun, Y. Chrysanthou, D. Cohen-Or. Virtual Occluders: An Efficient Intermediate PVS Representation. To appear in Eurographics Rendering Workshop 2000.

[13] A. Laurentini. The visual hull concept for silhouette-based image understanding. IEEE Transactions on Pattern Analysis and Machine Intelligence, 16(2), pp. 150-162, 1994.

[14] H. Plantinga, C. R. Dyer. Visibility, Occlusion, and the Aspect Graph, IJCV(5), No. 2, pp. 137-160, 1990.

[15] J. Rohlf, J. Helman. IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. SIGGRAPH 94 Conference Proceedings, pp. 381-395, 1994.

[16] G. Schaufler, X. Decoret, J. Dorsey, F. Sillion. Conservative Volumetric Visibility with Occluder Fusion. To appear in SIGGRAPH 2000 Conference Proceedings.

[17] G. Schaufler, W. Stürzlinger. A Three-Dimensional Image Cache for Virtual Reality. Computer Graphics Forum (Proceedings of EUROGRAPHICS'96), 15(3), pp. 227-235, 1996.

[18] J. Shade, D. Lischinski, D. Salesin, T. DeRose, J. Snyder. Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. SIGGRAPH 96 Conference Proceedings, pp. 75-82, 1996.

[19] F. Sillion, G. Drettakis, B. Bodelet. Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery. Computer Graphics Forum (Proceedings of EUROGRAPHICS'97), 16(3), pp. 207-218, 1997.

[20] A. Stewart. Hierarchical Visibility in Terrains. Eurographics Rendering Workshop 1997, pp. 217-228, 1997.

[21] S. Teller. Computing the antipenumbra of an area light source. Computer Graphics (SIGGRAPH 92 Conference Proceedings), vol. 26, pp. 139-148, 1992.

[22] S. Teller, C. Sequin. Visibility preprocessing for interactive walkthroughs. Computer Graphics (SIGGRAPH 91 Conference Proceedings), vol. 25, pp. 61-69, 1991.

[23] P. Wonka, D. Schmalstieg. Occluder Shadows for Fast Walkthroughs of Urban Environments. Computer Graphics Forum (Proceedings of EUROGRAPHICS'99), pp. 51-60, 1999.

[24] P. Wonka, M. Wimmer, D. Schmalstieg. Visibility Preprocessing with Occluder Fusion for Urban Walkthroughs. Technical Report TR-186-2-00-06, Vienna University of Technology, 2000.

[25] H. Zhang, D. Manocha, T. Hudson, K. E. Hoff. Visibility Culling Using Hierarchical Occlusion Maps. SIGGRAPH 97 Conference Proceedings, pp. 77-88, 1997.

Appendix: A proof for conservative point sampling

Let an occluder O be an arbitrary connected subset of R^3. A *shrunk occluder* O' is a subset of O so that for all points $A \in O'$: $|B-A| \leq \varepsilon \rightarrow B \in O$.

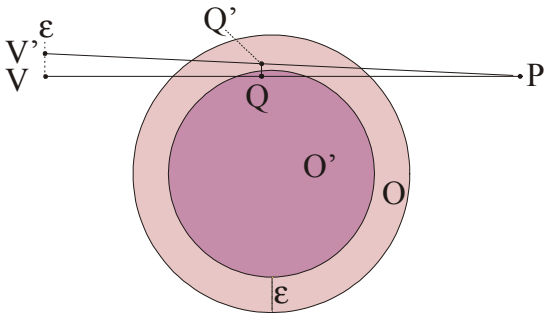

Figure 6. Any point P hidden by a shrunk occluder O' from V is also hidden by the original occluder O from any point V' in an ε-neighborhood of V.

Theorem: Any point P that is occluded by O' seen from a sample point V is also occluded by O from any sample point V' with $|V'-V| \leq \varepsilon$ (see Figure 6).

Proof: Assume there is a V' for which P is not occluded by O. Then any point along the line segment V'P must not be contained in O. Since P is occluded, there is at least one point $Q \in O'$ along the line segment VP. VV'P form a triangle and $|V'-V| \leq \varepsilon$, so there must be point Q' along V'P with $|Q'-Q| \leq \varepsilon$. By definition, all points within an ε-neighborhood of Q are contained in O, so Q' is contained in O. Therefore P cannot be visible from V'. q.e.d.

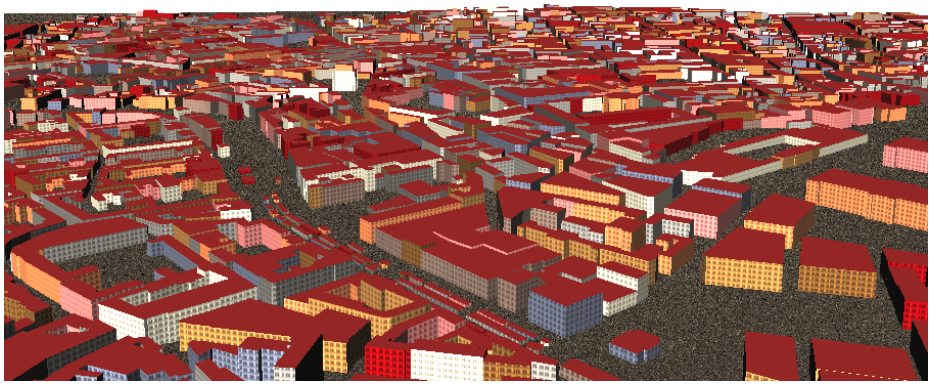

Figure 7. Overview of the 8 million polygon model of the city of Vienna used as a test scene. Note that the city is modeled in very high detail – every window, for example, is modeled separately with 26 polygons