Visibility Preprocessing for Urban Scenes using Line Space Subdivision

Jiří Bittner
Center for Applied Cybernetics
Czech Technical University in Prague
bittner@fel.cvut.cz

Peter Wonka
Institute of Computer Graphics
Vienna University of Technology
wonka@cg.tuwien.ac.at

Michael Wimmer
Institute of Computer Graphics
Vienna University of Technology
wimmer@cg.tuwien.ac.at

Abstract

We present an algorithm for visibility preprocessing of urban environments. The algorithm uses a subdivision of line space *to analytically calculate a conservative potentially visible set for a given region in the scene. We present a detailed evaluation of our method including a comparison to another recently published visibility preprocessing algorithm. To the best of our knowledge the proposed method is the first algorithm that scales to large scenes and efficiently handles large view cells.*

1. Introduction

Applications in urban simulation, such as architectural walkthroughs, driving simulation, or visual impact analysis, have to cope with large amounts of data. A popular approach to reduce the amount of geometry to be rendered is to precompute visibility in order to render only objects that can be seen. The *view space* is usually broken down into a number of *view cells*, and for each view cell, a *potentially visible set* (PVS) is precalculated [23, 12, 31, 16]. This concept is depicted in Figure 1.

Calculating visibility for a 3D spatial region is a complex problem. Previous methods (Schaufler et al. [23] and Durand et al. [12]) rely on several simplifications to cope with the computational complexity of 3D visibility. While these algorithms can handle a large variety of scenes, they only consider a subset of possible occluder interactions (also called *occluder fusion*) and require comparatively high calculation times.

It is useful to develop visibility algorithms building on simplifications that match the demands of specific types of scenes. Wonka et al. observed that urban environments can be seen as 2.5D scenes [31]. Although they propose an algorithm that handles all types of occluder fusion and treats occlusion systematically, it does not scale very well to large scenes and large view cells.

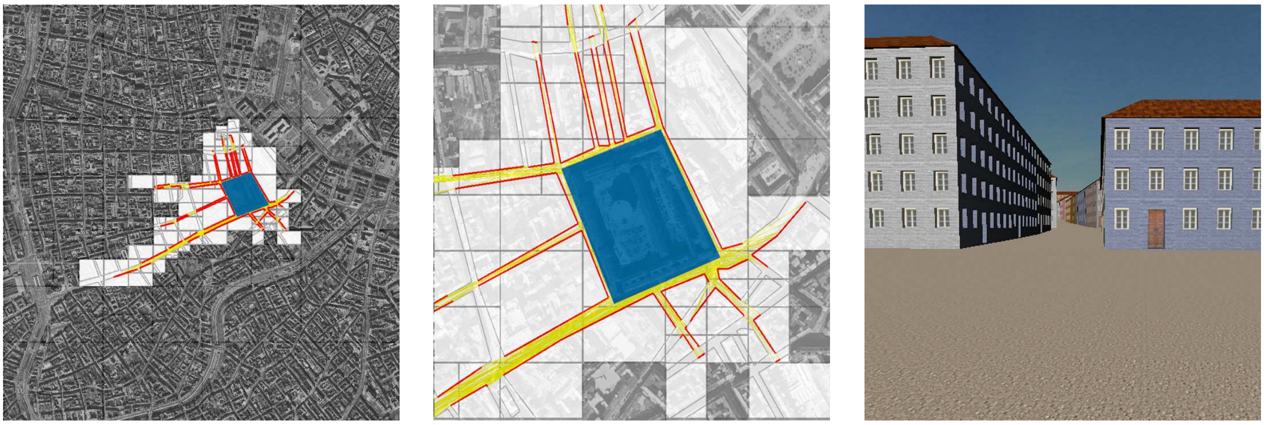
In this paper, we propose a visibility algorithm for 2.5D environments which significantly improves the *scalability* of previous methods. The algorithm exhibits *output-sensitive* behavior and therefore it is especially useful for large scenes and large view cells, both of which cannot be easily handled by previous techniques[1]. Additionally, we will demonstrate on a test model of Vienna that the visibility solution provided by our algorithm finds a *tighter PVS* than the method proposed by Wonka et al. [31], while requiring less calculation time.

The main idea of the algorithm presented in this paper is to combine an exact solution to 2D visibility in *line space* with a conservative solution in *primary space* for the remaining "half" dimension. The only simplification used in our algorithm is based on conservatively approximating edge-edge-edge (EEE) event surfaces (ruled quadrics [27]) by planes. The key observation is that while EEE event surfaces do occur for typical view cells in an urban scene, they rarely result in a change of visibility classification.

2. Related work

A lot of research has been devoted to visibility problems due to their importance in computer graphics, computer vision, and robotics. An excellent interdisciplinary survey was recently published by Durand [10]. We first review relevant visibility algorithms for 3D scenes, then we survey related 2.5D and 2D visibility methods.

[1] After submitting this paper for review, Koltun et al. [17] published a method that also efficiently handles large view cells.



(a) (b) (c)

Figure 1. (a) Selected view cell in the scene representing the city of Vienna and the corresponding PVS. The dark regions were culled by hierarchical visibility tests. (b) A closeup of the view cell and its PVS. (c) Snapshot of an observer's view from a viewpoint in the view cell.

Exact regional visibility for 3D scenes is a very demanding task. It was addressed by Plantinga and Dyer [20], who used the *aspect graph*—the graph of all qualitatively different views of an object (aspects). Durand et al. [11] introduced the *visibility skeleton* that captures all critical *visual events*. Teller [27], Drettakis and Fiume [9], and Stewart and Ghali [25] dealt with the 3D regional visibility problem in the context of computation of shadow boundaries. All currently published exact methods are not directly applicable to large scenes due to their computational complexity and robustness problems.

Visibility culling techniques have been introduced to speedup rendering of large scenes where only a fraction of the scene is actually visible [5]. Generally, we can distinguish between offline methods that preprocess visibility, and online methods that perform most visibility culling in real time. General techniques for image-space online visibility culling are the *hierarchical z-buffer* introduced by Greene et al. [13] and *hierarchical occlusion maps* described by Zhang et al. [32]. Object-space methods for online occlusion culling were proposed by Coorg and Teller [8], Manocha et al. [15] and Bittner et al. [2]. Online techniques require recomputation of visibility for each change of the viewpoint and do not provide global visibility information. On the contrary, the offline techniques typically precompute a superset of visible objects for each viewpoint of a given view cell.

Airey et al. [1] applied visibility preprocessing to architectural models. Visibility in indoor scenes was further studied by Teller and Séquin [28]. Both methods partition the scene into cells and portals. For each cell they identify objects visible through sequences of portals. These objects form a PVS for each cell. An online variant of the *cell-portal* visibility algorithm was published by Luebke and Georges [18]. These methods are restricted to indoor scenes with a particular structure. Recently, several techniques for visibility preprocessing were introduced that are suited to urban environments. Cohen-Or et al. [6] use ray-shooting to sample occlusion due to a single convex occluder. Schaufler et al. [23] use *blocker extensions* to handle *occluder fusion*—occlusion due to multiple occluders. Durand et al. [12] propose *extended occluder projections* and an occlusion sweep to handle occluder fusion. Wonka et al. [31] use *cull maps* for visibility preprocessing in 2.5D scenes. Visibility in terrains was studied by Stewart [24], and Cohen-Or and Shaked [7].

2D visibility was studied intensively in computational geometry as well as in computer graphics. The visibility graph [30] is a well-known structure for capturing visibility in 2D scenes. Vegter introduced the visibility diagram [29], which contains more information than the visibility graph. The visibility complex is a similar structure introduced by Pochiolla and Vegter [21]. The visibility complex for polygonal scenes was studied by Riviere [22]. Orti et al. use the visibility complex for 2D radiosity [19], Cho and Forsyth [4] applied it to 2D ray tracing. Hinkenjann and Müller [14] describe *hierarchical blocker trees*—a discrete structure similar to the visibility complex.

3. Overview

In this paper, we consider scenes of 2.5D nature, as for example models of urban environments. Such scenes can contain arbitrary geometry, but *occluders* are restricted to be vertical trapezoids connected to the ground (typically building façades). The goal of the algorithm is to determine all objects that are potentially visible from a convex polyhedral view cell made up of several *vertical faces*. In order to solve this task, it is sufficient to consider only the *top edges* of all scene entities—the view cell faces, occluders and object bounding boxes [31].

Our algorithm organizes the scene in a spatial hierarchy. For each top edge of a given view cell, it processes occluders in an approximate front-to-back order and incrementally builds a hierarchical structure in *line space* which represents the currently visible parts of the scene with respect to the already processed occluders.

For each occluder, we perform the following steps:

- Construct a line space *blocker polygon* from its 2D footprint on the ground plane.
- Calculate intersections with already processed blocker polygons. As a result, the blocker polygon is split into several fragments. Each fragment represents a set of rays that can be blocked by the same sequence of occluders. This set of rays also induces a certain *occluder fragment*.
- For each blocker polygon fragment, we test visibility of the corresponding occluder fragment by mapping the problem back to primary space.
- If an occluder fragment is found visible, the line space structure is updated accordingly.

The remainder of the paper is organized as follows: In Section 4 we discuss the correspondence between *primary space* and *line space*. In Section 5 we describe the *funnel visibility test* used to determine if an occluder fragment is visible with respect to a given set of rays. In Section 6 we outline the complete hierarchical visibility algorithm. In Sections 7 and 8 we evaluate and discuss our implementation of the proposed methods.

4. Visibility and line space

The proposed visibility algorithm operates mainly on a 2D projection of the scene. The "heights" of scene entities are considered only when necessary.

In order to solve the underlying 2D visibility problem, we use a mapping of oriented 2D lines to points in 2D oriented projective space that we call *line space* [26, 27]. Consequently, points in primary space map to oriented lines in line space. To denote entities in line space, we use the "starred notation", e.g. line l maps to l^*.

The mapping of the problem to line space allows us to represent complex 2D ray bundles (which carry the crucial part of visibility information) by simple polygons. For more details on the mapping see [3].

4.1. Blocker polygon

In this section, we describe the correspondence of occluders in primary space and blocker polygons in line space. A blocker polygon carries the 2D visibility information induced by an occluder and an edge of the given view cell. More specifically, it represents all 2D rays that emanate from the view cell edge and intersect the occluder. This set of rays is bounded by four *critical lines*, forming an hourglass shaped region that we call *funnel* (see Figure 2-(a)). The four critical lines map to points in line space and these points define the corresponding line space *blocker polygon* (see Figure 2-(b)).

Conversely, starting from a blocker polygon a corresponding funnel in primary space can be constructed by inverse mapping of the vertices of the blocker polygon to oriented lines in primary space.

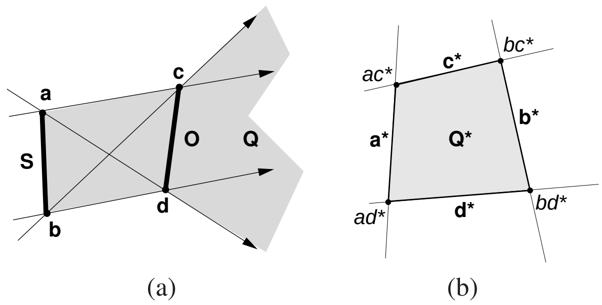


Figure 2. (a) A view cell edge S and an occluder O. (b) Blocker polygon Q^* corresponding to the primary space funnel Q.

If only 2D visibility were required, blocker polygons could be used to solve the visibility problem in the following way: process occluders in front-to-back order. To determine whether a newly added occluder is visible, it suffices to test whether its associated blocker polygon is completely covered by other blocker polygons in line space. In such a case, the occluder is invisible: any 2D ray through which the new occluder could be visible is occluded by the already processed occluders.

4.2. Subdivision of line space

Mapping several occluders to line space induces a subdivision of line space into polygonal cells. Each cell contains a sequence of blocker polygons ordered by the distance of the occluders to the given view cell edge. This means that a 2D ray defined by a point in such a cell intersects all occluders associated with the cell. Figure 3 depicts a line space subdivision induced by three occluders.

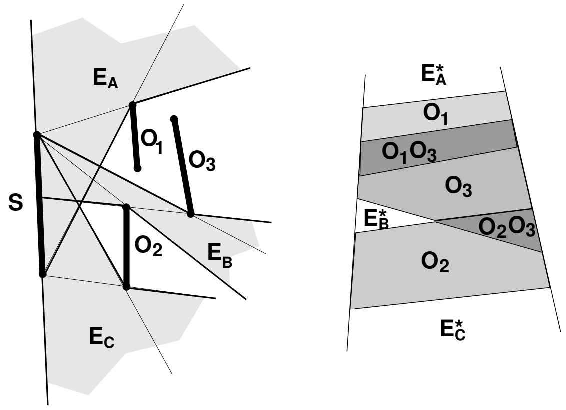


Figure 3. (a) The projection of a view cell edge and three occluders. E_A, E_B and E_C denote unoccluded funnels. (b) The line space subdivision. For each cell, the corresponding occluder-sequence is depicted. Note the cells E_A^*, E_B^* and E_C^* corresponding to unoccluded funnels.

The line space subdivision holds important visibility information. Each cell corresponds to a funnel of 2D rays that intersect the same sequence of occluders. Consequently the occluders can be visible only through 3D rays that project to the funnel. The edges of the subdivision correspond to changes in visibility.

Essentially, edges of blocker polygons only encode changes in *2D visibility*. In Section 5, we will show how to introduce additional edges into the subdivision corresponding to changes in visibility due to the height structure of the occluders.

We would like the subdivision to contain only blocker polygons that correspond to occluders actually visible in 2.5D. Blocker polygons corresponding to invisible occluders need not be considered. We therefore construct the subdivision of line space incrementally and determine whether the currently processed occluder O is visible with respect to the occluders already processed.

The overall algorithm for constructing the line space subdivision proceeds as follows: for each occluder O, we identify the cells of the subdivision that are intersected by the corresponding blocker polygon. For each such cell, visibility of O is tested in primary space using occluders associated with this cell (this test will be discussed in the next section). If O is found visible, it is inserted into the sequence of occluders for the cell, or the cell is further subdivided, depending on the height structure of occluders. If O is occluded, no changes are necessary.

5. Funnel visibility test

This section describes the *funnel visibility test* used at the core of our algorithm. The goal is to classify visibility of a new occluder O_n with respect to a cell $\mathcal{Q}^*$ of the current line space subdivision. The test is carried out inside the primary space funnel $\mathcal{Q}$ which the cell $\mathcal{Q}^*$ maps to. The solution we propose is conservative—it approximates occlusion due to EEE event surfaces.

The funnel visibility test can have three possible results:

- O_n is completely occluded by occluders associated with $\mathcal{Q}^*$. In this case, O_n does not contribute to this cell.
- The top edge of O_n is visible across the whole funnel. In this case, it is simply added to the sequence of relevant occluders for $\mathcal{Q}^*$.
- O_n is visible in only a part of the funnel. This means that a change of visibility occurs inside $\mathcal{Q}$, and the line space subdivision needs to be updated.

The visible part of O_n is computed as follows: We select all occluders stored in the cell $\mathcal{Q}^*$ which lie in front of O_n in primary space. For each such occluder O, we calculate two *shadow planes*, against which we clip the top edge of O_n. The first plane, π_o, is defined by the top edge of O and a vertex of the top edge of the view cell face S, such that S and O lie on the same side of π_o. The second plane, π_s, is defined by an edge of S and a vertex of the top edge of O, such that S and the top edge of O lie on opposite sides of π_s (see Figure 4). The planes are defined so that any point below them is occluded by O considering all rays from the funnel.

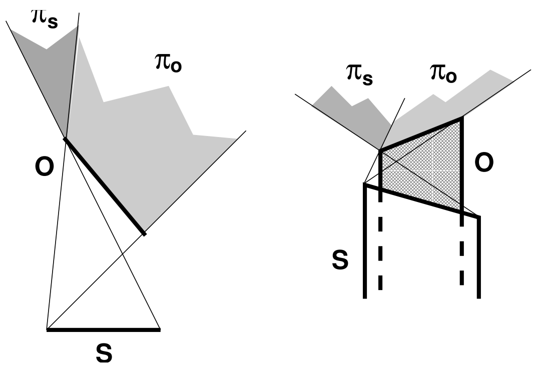


Figure 4. (a) Projections of the two shadow planes due to an occluder O and a view cell face S. (b) Front view of the view cell face, occluder and the two shadow planes.

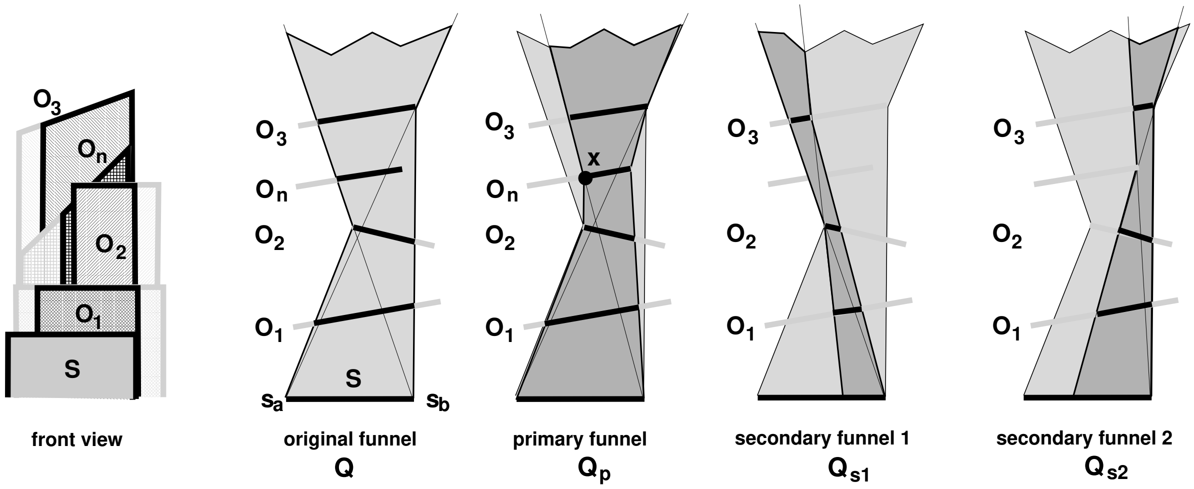


Figure 5. Adding a new occluder O_n into a funnel induced by three occluders can yield three new funnels.

Note that clipping the top edge of O_n against π_o and π_s results in at most one visible fragment due to the concavity of the two shadow planes. So even after clipping the edge by the shadow planes of all occluders in front of O_n, there is either a single fragment of the top edge left, or O_n is considered occluded.

If there is a visible fragment left, each of its endpoints lying inside the funnel is mapped to an edge in line space. The resulting line space edge(s) are used to subdivide the original line space cell $\mathcal{Q}^*$ into two or three new cells. In exactly one of these cells, a fragment of O_n is visible and it is added to the associated sequence of occluders.

To see what this means in primary space, consider the two or three newly established funnels corresponding to the new cells. The *primary funnel* includes the visible part of O_n. The *secondary funnels* correspond to parts of the original funnel where O_n is either invisible or which it does not intersect at all.

Figure 5 depicts a funnel $\mathcal{Q}$ containing three occluders. The new occluder O_n is partially visible with respect to the funnel. Funnel $\mathcal{Q}$ is split into three new funnels. The primary funnel $\mathcal{Q}_p$ contains the occluder sequence O_1, O_2, O_n, O_3. The first secondary funnel $\mathcal{Q}_{s1}$ is induced by the part of O_n hidden by the shadow plane of O_2. It contains the occluder sequence O_1, O_2, O_3. The other secondary funnel $\mathcal{Q}_{s2}$ corresponds to the set of 2D rays that do not intersect O_n. Figure 6 depicts the three blocker polygons corresponding to funnels $\mathcal{Q}_p$, $\mathcal{Q}_{s1}$ and $\mathcal{Q}_{s2}$.

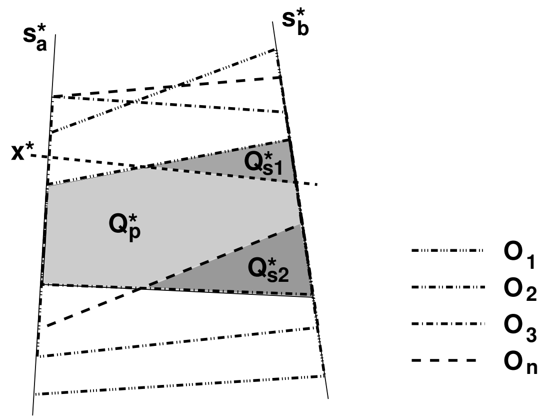


Figure 6. Three blocker polygons corresponding to funnels Q_p, Q_{s1}, Q_{s2} from Figure 5. x^* is the mapping of point x introduced by clipping occluder O_n by a shadow plane.

6 Hierarchical visibility algorithm

To efficiently implement the algorithms described in this paper, we make use of two hierarchical structures:

- The subdivision of line space is maintained by a *Binary Space Partitioning* (BSP) tree. The BSP tree greatly facilitates the required operations on the line space subdivision.
- The whole scene including the occluders is organized in a kD-tree.

The kD-tree is used for two main purposes:

- First, it allows to determine an *approximate front-to-back order* of occluders efficiently.

- Secondly, pruning of kD-tree nodes by hierarchical visibility tests leads to the expected output-sensitive behavior of the algorithm.

The hierarchical visibility algorithm traverses the kD-tree in an approximate front-to-back order with respect to the given view cell edge. The order is established using a priority queue, where the priority of a node is inversely proportional to the minimal distance of the node from the view cell. Occluders stored within a leaf node are processed in random order.

The line space BSP tree is constructed incrementally by inserting blocker polygons corresponding to the currently processed occluder.

```
HierarchicalVisibility( Viewcell edge S, kDTree KD ) {
 LSSD.Init(S)  // initiate line space subdivision
 pqueue.Put(KD.root)  // initiate priority queue
 while (pqueue is not empty) {
  N ← pqueue.Get() // get next node from the queue
  if (R_N intersects S)
   N.vis ← VISIBLE
  else
   N.vis ← LSSD.TestVisibility(R_N)
  if (N.vis != INVISIBLE) {
   if (N is leaf)
    LSSD.InsertOccluders( N.occluders )
   else
    pqueue.Put( children of N )
  }
 } // while
 // classify visibility of scene objects
 foreach visible KD node N
  foreach object O of N
   O.visibility = LSSD.TestVisibility(O)
}
```

Figure 7. Pseudo-code of the complete hierarchical visibility algorithm.

BSP tree construction is interleaved with visibility tests of the currently processed kD-tree node. The visibility test classifies visibility of the node with respect to the already processed occluders. If the node is invisible, the subtree rooted at the node and all occluders it contains are culled. If it is visible, the algorithm recursively continues testing visibility of its descendants. In the special case of a node intersecting the view cell edge, it is classified visible.

The visibility classification of all scene objects is carried out after the complete line space subdivision has been constructed by testing object bounding boxes inside visible kD-tree nodes. The pseudo-code of the hierarchical visibility algorithm is outlined in Figure 7.

7. Results

We have evaluated the proposed method using a scene representing a large part of the city of Vienna. We made a comparison with the discrete hardware-accelerated approach by Wonka et al. [31]. In the comparison, we call our method the *line space subdivision* method (LSS) and Wonka et al.'s method the *discrete cull map* method (DCM).

For evaluation of the LSS we used a PC equipped with a 950MHz Athlon CPU, and 256MB RAM. The DCM was evaluated on a PC with 650MHz Pentium III, 512MB RAM, and a GeForce DDR graphics card.

The tested scene represents 8 km^2 of the city of Vienna and consists of approximately 8 million triangles. The triangles are grouped into 17854 objects that are used for visibility classification. We automatically synthesised 15243 larger polygons to be used as occluders. Most occluders correspond to building façades.

We conducted three different tests. In the first test, we randomly selected 105 out of 16447 view cells in the whole scene. For each view cell edge we computed a PVS. Figure 8 shows two plots depicting the sizes of the PVS and the running times of the two methods for each processed view cell edge.

The second test was designed to test the scalability of the two methods with respect to view cell size. We manually placed 10 larger view cells (with perimeter between 600 and 800 meters) and applied the algorithms on the corresponding edges. Table 1 summarizes the results for both the first and the second tests.

Test	Method	Avg. PVS size	Avg. time $[ms]$
Test I	DCM	105.2	202.1
	LSS	**84.7**	**44.6**
Test II	DCM	274.0	4304.8
	LSS	**236.8**	**211.9**

Table 1. Average number of visible objects and corresponding computation times for the first and the second tests.

The last test was carried out only for LSS. The goal was to test the scalability of the method with respect to the size of the scene and verify its output-sensitive behavior. We

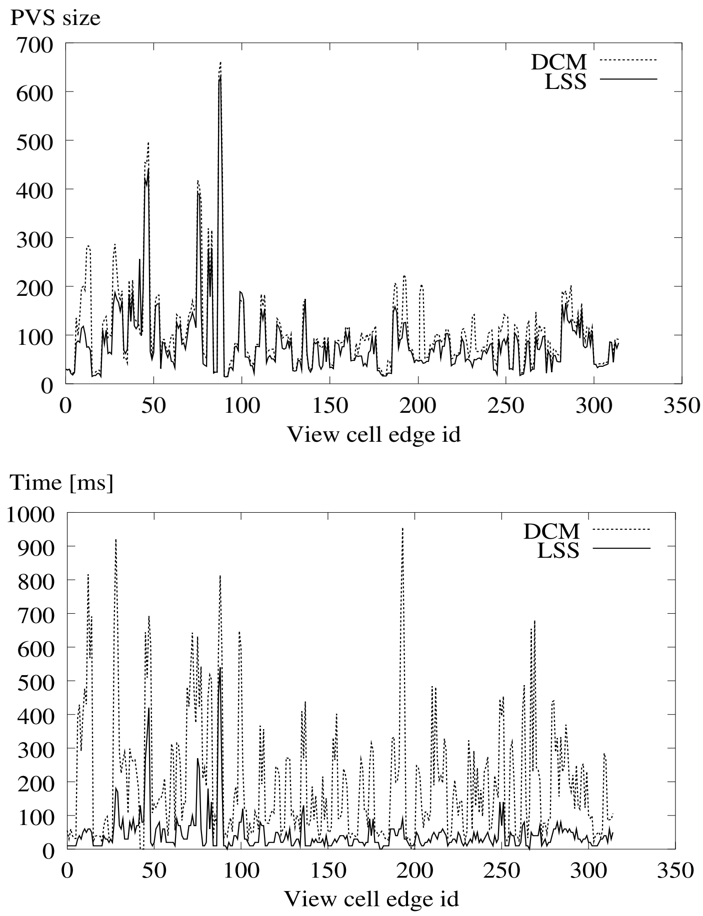


Figure 8. (top) The number of potentially visible objects for 305 view cell edges. (bottom) The running times of the two tested methods.

replicated the original scene on grids of size 2x2, 4x4 and 6x6. We selected a few view cells that provided the same size of the resulting PVS for each of the tested scenes. Table 2 depicts the size of the kD-tree and computation times for the original scene and the three larger replicated scenes.

Grid	Area $[km^2]$	kD nodes	Avg. time $[ms]$
1x1	8	1609	90
2x2	32	6511	92
4x4	128	26191	101
6x6	288	57935	105

Table 2. Average PVS computation times for different scene sizes.

8 Discussion

In this section, we give an interpretation of the results. We discuss the suitability of our method for real-time rendering, the importance of large view cells and the scalability of the method to large scenes.

8.1 Real-time rendering

The first test shows the calculation times and the PVS sizes for smaller view cells. We observe that both methods produce comparable results. The view cell size for this test was chosen so as to give reasonably-sized PVSs that would allow for walkthroughs with high frame rates [31]. The LSS analytical method often produces a tighter PVS, because it does not rely on discretization and occluder shrinking.

8.2 Large view cells

The second test shows the scalability of the method for larger view cells. Although smaller view cells are more interesting for real-time rendering applications, larger view cells can be very useful. If we consider a very simple model, for example, where each façade is just one large flat polygon, it can be sufficient to calculate a solution for a rather large view cell. Another very important application is the hierarchical precalculation of visibility information. Similar to previous methods [12], the visibility calculation could start with a subdivision of the view space into larger view cells. Smaller view cells are only calculated when necessary (e.g., when the size of the PVS is too large or when a heuristic determines large changes in visibility within the view cell). For urban environments, this hierarchical approach can be efficiently combined with a priori knowledge about the scene structure. If we use street sections as view cells, we can observe that visibility within one street section hardly changes (see Figure 9). In this context, we also want to emphasize that our view cells are not restricted to axis-aligned boxes.

8.3 Output sensitivity

The third test shows the scalability of the method to larger scenes. It is a desired property of a visibility algorithm that the computation time mainly depends on the size of the PVS (=output) and not on the size of the scene (=input). The results strongly indicate output sensitivity of the algorithm in practice: the calculation times hardly change when the size of the scene is increased. Such a behavior can not be achieved easily by previous methods [31, 12, 23].

9. Conclusion and Future Work

We have introduced an algorithm that determines visibility from a given view cell in a 2.5D scene, such as an urban environment. It combines an exact solution to the 2D visibility problem with a tight conservative solution for the remaining “half” dimension. Coherence of visibility is exploited by using a hierarchical subdivision of line space as well as a hierarchical organization of the scene. In practice, our algorithm achieves output-sensitive behavior by combining ordered processing of occluders and hierarchical visibility culling.

We have demonstrated that the method is suitable for visibility preprocessing of large scenes by applying it to a scene representing a large part of the city of Vienna. The proposed method compares favorably with the previously published algorithm by Wonka et al. [31]. To the best of our knowledge, our method is the first algorithm that scales to large scenes and efficiently handles large view cells.

Currently, we are designing an exact analytic solution for regional visibility in 2.5D scenes. Such a solution is crucial for evaluation of efficiency of all recently proposed methods [23, 12, 31, 16].

Acknowledgements

Many thanks to Vlastimil Havran and Jan Přikryl for their helpful comments. This research was supported by the Czech Ministry of Education under Project LN00B096, the Aktion Kontakt OE/CZ grant number 1999/17 and the Austrian Science Foundation (FWF) contract no. p-13876-INF.

References

[1] J. M. Airey, J. H. Rohlf, and F. P. Brooks, Jr. Towards image realism with interactive update rates in complex virtual building environments. In *1990 Symposium on Interactive 3D Graphics*, pages 41–50. ACM SIGGRAPH, Mar. 1990.

[2] J. Bittner, V. Havran, and P. Slavík. Hierarchical visibility culling with occlusion trees. In *Proceedings of Computer Graphics International '98 (CGI'98)*, pages 207–219. IEEE, 1998.

[3] J. Bittner and J. Přikryl. Exact regional visibility using line space partitioning. Technical Report TR-186-2-01-06, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Karlsplatz 13/186/2, A-1040 Vienna, Austria, 2001. human contact: technical-report@cg.tuwien.ac.at.

[4] F. S. Cho and D. Forsyth. Interactive ray tracing with the visibility complex. *Computers and Graphics*, 23(5):703–717, Oct. 1999.

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

[6] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. In *EUROGRAPHICS'98*, 1998.

[7] D. Cohen-Or and A. Shaked. Visibility and dead-zones in digital terrain maps. *Computer Graphics Forum*, 14(3):C/171–C/180, Sept. 1995.

[8] S. Coorg and S. Teller. Real-time occlusion culling for models with large occluders. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 83–90, New York, Apr.27–30 1997. ACM Press.

[9] G. Drettakis and E. Fiume. A Fast Shadow Algorithm for Area Light Sources Using Backprojection. In *Computer Graphics (Proceedings of SIGGRAPH '94)*, pages 223–230, 1994.

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

[11] F. Durand, G. Drettakis, and C. Puech. The visibility skeleton: A powerful and efficient multi-purpose global visibility

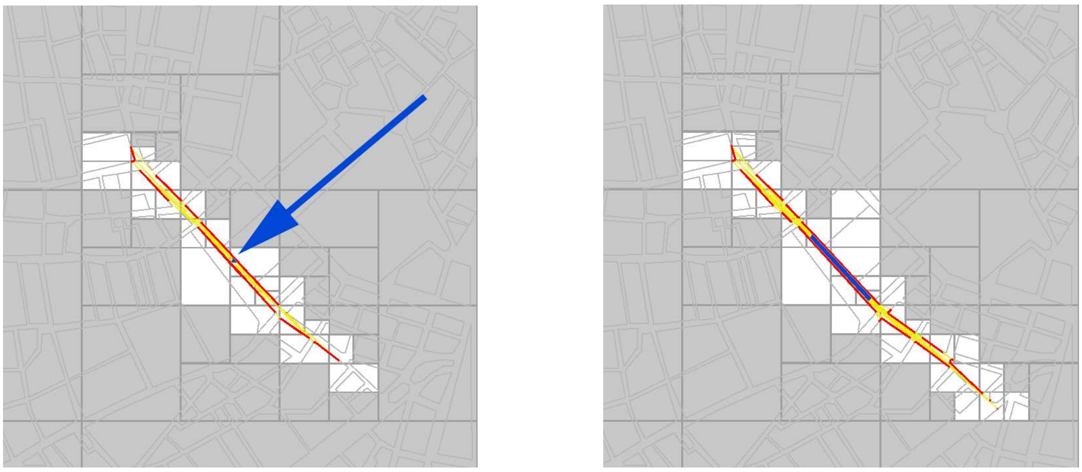


Figure 9. (left) A small view cell and its PVS. (right) The PVS for a larger view cell can be very similar.

tool. In *Computer Graphics (Proceedings of SIGGRAPH '97)*, pages 89–100, 1997.

[12] F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. In *Computer Graphics (Proceedings of SIGGRAPH 2000)*, pages 239–248, 2000.

[13] N. Greene, M. Kass, and G. Miller. Hierarchical Z-buffer visibility. In *Computer Graphics (Proceedings of SIGGRAPH '93)*, pages 231–238, 1993.

[14] A. Hinkenjann and H. Müller. Hierarchical blocker trees for global visibility calculation. Research Report 621/1996, University of Dortmund, Aug. 1996.

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

[16] V. Koltun, Y. Chrysanthou, and D. Cohen-Or. Virtual occluders: An efficient intermediate pvs representation. In *Proceedings of the 11th EUROGRAPHICS Workshop on Rendering*, 2000.

[17] V. Koltun, Y. Chrysanthou, and D. Cohen-Or. Hardware-accelerated from-region visibility using a dual ray space. In *Proceedings of the 12th EUROGRAPHICS Workshop on Rendering*, 2001.

[18] D. Luebke and C. Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In P. Hanrahan and J. Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 105–106. ACM SIGGRAPH, Apr. 1995.

[19] R. Orti, S. Riviere, F. Durand, and C. Puech. Using the Visibility Complex for Radiosity Computation. In *Lecture Notes in Computer Science (Applied Computational Geometry: Towards Geometric Engineering)*, volume 1148, pages 177–190, Berlin, Germany, May 1996. Springer-Verlag.

[20] H. Plantinga and C. Dyer. Visibility, occlusion, and the aspect graph. *International Journal of Computer Vision*, 5(2):137–160, 1990.

[21] M. Pocchiola and G. Vegter. The visibility complex. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 328–337, 1993.

[22] S. Rivière. Dynamic visibility in polygonal scenes with the visibility complex. In *Proceedings of the 13th International Annual Symposium on Computational Geometry (SCG-97)*, pages 421–423, New York, June 4–6 1997. ACM Press.

[23] G. Schaufler, J. Dorsey, X. Decoret, and F. X. Sillion. Conservative volumetric visibility with occluder fusion. In *Computer Graphics (Proceedings of SIGGRAPH 2000)*, pages 229–238, 2000.

[24] A. J. Stewart. Hierarchical visibility in terrains. In *Proceedings of Eurographics Rendering Workshop '97*, pages 217–228, 1997.

[25] A. J. Stewart and S. Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. In *Computer Graphics (Proceedings of SIGGRAPH '94)*, pages 231–238, 1994.

[26] J. Stolfi. *Oriented Projective Geometry: A Framework for Geometric Computations*. Academic Press, 1991.

[27] S. J. Teller. Computing the antipenumbra of an area light source. *Computer Graphics*, 26(2):139–148, July 1992.

[28] S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. In *Computer Graphics (Proceedings of SIGGRAPH '91)*, pages 61–69, 1991.

[29] G. Vegter. The visibility diagram: a data structure for visibility problems and motion planning. In *SWAT 90, 2nd Scandinavian Workshop on Algorithm Theory*, volume 447 of *Lecture Notes in Computer Science*, pages 97–110. Springer, 1990.

[30] E. Welzl. Constructing the visibility graph for n-line segments in $O(n^2)$ time. *Information Processing Letters*, 20(4):167–171, May 1985.

[31] P. Wonka, M. Wimmer, and D. Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. In *Proceedings of the 11th EUROGRAPHICS Workshop on Rendering*, pages 71–82, 2000.

[32] H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff III. Visibility culling using hierarchical occlusion maps. In *Computer Graphics (Proceedings of SIGGRAPH '97)*, Annual Conference Series, pages 77–88, 1997.