Eurographics Symposium on Rendering (2005)
Kavita Bala, Philip Dutré (Editors)

# Fast Exact From-Region Visibility in Urban Scenes

Jiří Bittner[†], Peter Wonka[‡], Michael Wimmer[†]

[†]Vienna University of Technology
[‡]Arizona State University

**Abstract**

*We present a fast exact from-region visibility algorithm for 2.5D urban scenes. The algorithm uses a subdivision of line space for identifying visibility interactions in a 2D footprint of the scene. Visibility in the remaining vertical dimension is resolved by testing for the existence of lines stabbing sequences of virtual portals. Our results show that exact analytic from-region visibility in urban scenes can be computed at times comparable or even superior to recent conservative methods.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

## 1. Introduction

Visibility calculation methods aim to identify the objects visible either from a point or from a region in space (called *view cell*). A typical application for these methods is to accelerate rendering by sending only visible primitives to the graphics hardware [COCSD02]. Of particular interest are from-region visibility algorithms, because they can be computed in a preprocess or lazily parallel to the rendering process.

Unfortunately, the from-region visibility problem is highly complex even in 2.5D scenes. Recent advances in visibility research have lead to exact solutions for 3D scenes, albeit at a computational effort that is not justified for many practical—especially urban—scenes. Other algorithms calculate a conservative *potentially visible set* (PVS), which might include significantly more objects than actually visible from the view cell. These algorithms build on various heuristics and simplifications: omission or simplification of complex visibility interactions (often in the form of missing occluder fusion), approximations due to discretization, limitations on the size, shape and placement of view cells, overestimation of the occludee sizes, approximate ordering of occluders, etc.

In this paper, we show a new *exact* solution to the 2.5D from-region visibility problem that is both fast and suffers none of these shortcomings:

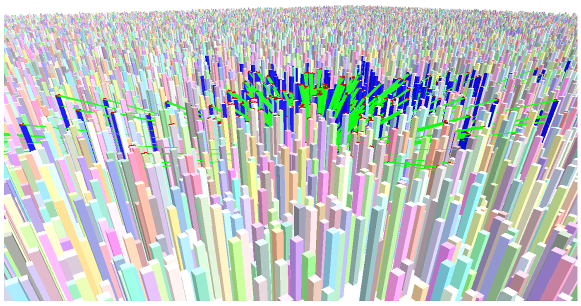
**Figure 1:** *Snapshot of a scene with 1M occluders with a PVS calculated using our new method. The buildings which form the PVS are shown in blue. The sightlines are shown in green. The PVS for the given view cell contained 1171 occludees and took 1.98 seconds to compute.*

- Its speed is comparable to or faster than previous conservative methods.
- It is exact, i.e., it accounts for all types of occluder fusion (unlike [DDTP00, SDDS00, WWS00, BWW01]), and does not rely on discretization (like [KCCO01, LSCO03, WWS00]), avoiding the need for excessive occluder shrinking.
- It poses no restriction on the size or shape of the view cell or occluders.

The main contribution of this paper is that it provides the first exact and practical solution of an important and widely researched topic, 2.5D from-region visibility. Visibility computations are needed in many different fields. It is very difficult for a practitioner to judge which of the many existing methods employs simplifications in a way that it is still accurate or flexible enough for a particular visibility problem. To give an example, an architect using a commercial GIS system is not likely to have the time nor the knowledge to find out how to approximate a desired view cell with squares, or how to set up parameters like discretization resolution so that his desired accuracy is achieved.

2.5D visibility is an important and easily identifiable subproblem, and therefore we believe that there is a high value in a solution that "just works", while at the same time being fast enough for practical usage. To demonstrate this point, we will show in Section 6 how simplifications commonly used in conservative methods can lead to non-intuitive results, which can be significantly worse than those obtained by the exact method.

Our method uses a subdivision of line space in order to find sequences of potential occluders for a given object in a 2D footprint of the scene. Visibility of the object is computed by interpreting the vertical extensions of the potential occluders as 3D portals and testing for the existence of a stabbing line through the portal sequence [Tel92b].

## 2. Related work

Visibility is a widely researched topic covered in a number of thorough surveys [Dur99, BW03, COCSD02]. We focus here on previous from-region visibility methods.

Visibility algorithms for indoor scenes typically exploit a cells-and-portals subdivision. Visibility from a cell is computed by checking sequences of portals for possible sight-lines. To solve this task, Airey [ARB90] used ray shooting, and Teller et al. [TS91, Tel92b] used stabbing line computations, which are also used as a part of our new algorithm.

For outdoor scenes, Wonka et al. [WWS00] use occluder shrinking and point sampling to calculate visibility in 2.5D scenes with the help of a hardware accelerated z-buffer. Koltun et al. [KCCO01] transform the 2.5D problem to a series of 2D visibility problems. The 2D problems are solved using dual ray space and the z-buffer algorithm. They also show an analytic method which is exact for a restricted 2D problem, but which turns out to be only conservative in the general 2.5D case. Leyvand et al. [LSCO03] use a different discretization of line space, and also relax the 2.5D constraint in some cases. All of these algorithms rely on discretization in graphics hardware, which limits their precision and scalability. Bittner et al. [BWW01] use a line space subdivision maintained by a BSP tree to calculate the PVS. This method forms the basis of our new algorithm.

For general 3D scenes, Durand et al. [DDTP00] proposed extended projections and an occlusion sweep to calculate conservative from-region visibility. Schaufler et al. [SDDS00] used blocker extensions to compute conservative visibility in scenes represented as volumetric data. Bittner [Bit02] proposed an algorithm using Plücker coordinates and BSP trees to calculate exact from-region visibility. A similar method was published by Nirenstein et al. [NBG02]. While these methods can handle general scenes, they are also significantly slower than dedicated methods for 2.5D scenes, and the conservative methods rely on discretization and other simplifications. Recently, Nirenstein et al. [NB04] advocate aggressive visibility algorithms, that provide a smaller PVS than the exact one at the cost of image errors.

The remainder of the paper is organized as follows: Section 3 gives an overview of the algorithm. In Section 4 we discuss the line space subdivision. In Section 5 we describe the stabbing line computation. In Section 6, we evaluate the proposed method and show some examples for its application.

## 3. Overview

Our method calculates exact from-region visibility for *2.5D scenes*, i.e., scenes that can be represented as a polygonal height field (a piece-wise linear function $f : \Omega \rightarrow R$, where $\Omega$ is a 2D domain). It can be shown that visibility of such scenes is equivalent to the visibility of the contained edges. Therefore, in our algorithm occluders and view cell faces are represented by general 3D edges and their vertical extensions to the ground. The algorithm computes (typically in a preprocess) an exact potentially visible set (PVS) in three interleaved steps:

**Hierarchical scene traversal.** The occluders and the scene geometry are stored in a spatial hierarchy (we use a kD-tree). For each face of a given view cell, the spatial hierarchy is traversed to obtain an approximate front-to-back order of occluders. The processing of occluders is interleaved with visibility tests of the currently processed kD-tree node. If a node is invisible, the subtree rooted at the node and all contained occluders are culled. If it is visible, the algorithm recursively continues testing visibility of its descendants.

**2D line space subdivision.** To resolve visibility in the horizontal direction, the algorithm incrementally builds a subdivision of *line space*, maintained by a 2D BSP tree. When an occluder is processed, a line space *blocker polygon* is constructed from its footprint and inserted into the tree. A blocker polygon is split according to the BSP leaves it intersects, each of which has a sequence of potential occluders associated.

**Portal visibility test.** Final visibility of an occluder is resolved using the occluder sequences identified by the line space subdivision. For each occluder in such a sequence,

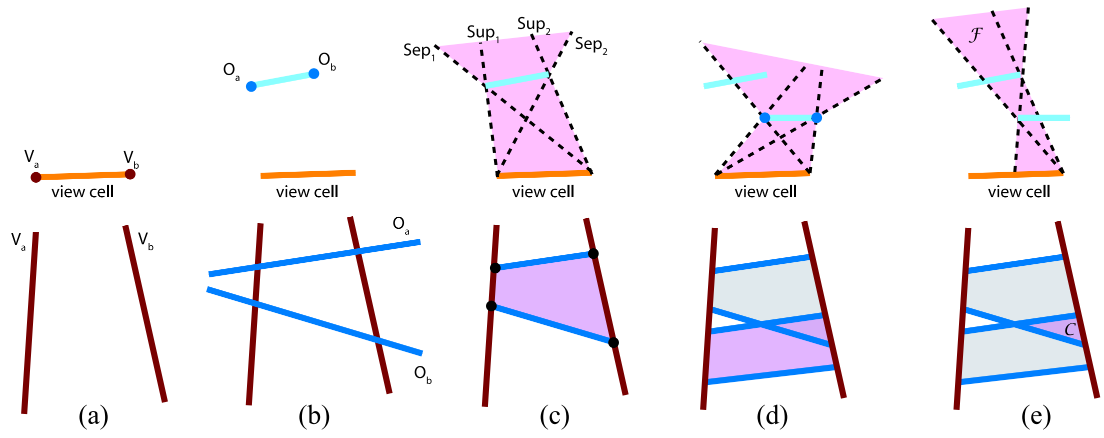

**Figure 2:** *Primal space (top) and line space (bottom). (a), (b) View cell and occluder endpoints map to lines in line space. (c) The resulting line space blocker polygon and its corresponding funnel of lines in primal space. (d) Another blocker polygon (e) The cell $\mathcal{C}$ in the line space subdivision corresponds to a funnel $\mathcal{F}$ in primal space.*

we construct a semi-infinite vertical portal. Then we test for the existence of a line stabbing all portals in the sequence. Thus we reduce the 2.5D visibility problem to the well-known portal visibility problem. Visible occluder fragments update the line space subdivision accordingly. Since the exact portal visibility test is quite costly, we introduce an efficient acceleration based on penumbra wedges that works in the majority of cases.

In practice, scenes will often not be given as height fields directly. As in previous 2.5D visibility algorithms, occluders are typically synthesized from the original 3D input model (how this should be done depends on the scene and is beyond the scope of this paper), whereas visibility tests for the original objects are performed using separate occludees, for example their bounding boxes. Note that to provide exact results in the case of separate occludees, they need to be clipped against the height field in order to guarantee a 2.5D input structure.

## 4. Line Space Subdivision

Our algorithm makes use of line space, a dual space where each point corresponds to a line in primal space. The 2D portion is based on an algorithm by Bittner et al. [BWW01, BPS03] and similarly uses Plücker coordinates to parameterize line space. The rays that emanate from the view cell and intersect an occluder correspond to a polygon in line space, the so-called *blocker polygon*. These polygons induce a 2D subdivision of line space into polygonal *line space cells*. Each line space cell $\mathcal{C}$ is associated with a sequence of blocker polygons ordered by the distance of occluders to the given view cell. Each line space cell corresponds to a *funnel* $\mathcal{F}$ in primal space; the funnel bounds all rays intersecting the sequence of occluders associated with the cell (see Figure 2).

The line space subdivision is created incrementally by using an *occlusion tree*, a BSP tree where nodes correspond to edges of blocker polygons. For each occluder $O$, we identify the line space cells that are intersected by the corresponding blocker polygon. For each such cell, visibility of $O$ is determined by the portal visibility test. If $O$ is occluded, no changes are necessary. If $O$ is visible in the whole cell, it is inserted into the sequence of occluders for the cell. Otherwise, the cell is further subdivided according to visible parts of $O$.

## 5. Portal visibility computation

The portal visibility test is the core of our exact visibility algorithm. The goal is to classify visibility of a blocker polygon (belonging to an occluder $O$) in a line space cell $\mathcal{C}$.

We construct 3D *virtual portals* in primal space for the given view cell face, the new occluder ($O$), and for each occluder associated with $\mathcal{C}$. For the view cell face and $O$, the virtual portal is bound by the top edge and two vertical half-lines directed downwards. For the other occluders, it is bound by the top edge and two vertical half-lines directed upwards (see Figure 3).

### 5.1. Stabbing line computation

The portal visibility test of $O$ in $\mathcal{C}$ is based on a *stabbing line computation* [Tel92b] which tests the existence of a ray which leaves the virtual view cell portal, passes through the virtual portals of all occluders in between (i.e., it is not blocked by these occluders, see Figure 3), and hits the virtual portal of $O$. If such a ray exists, $O$ is visible in the line space cell $\mathcal{C}$ (and its corresponding primal space funnel $\mathcal{F}$).

The actual stabbing line computation is carried out in 5D

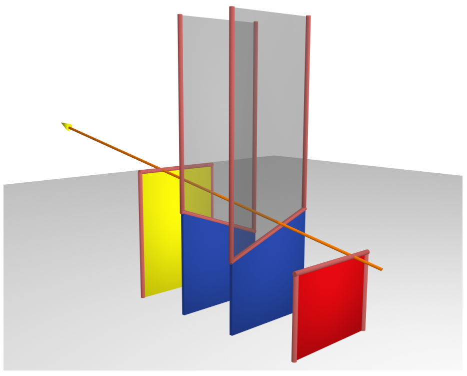

**Figure 3:** *Stabbing line computation. The view cell face is shown in red, two occluders identified by the line space subdivision in blue, and the currently processed occluder $O$ in yellow. The portals extend above the top edges of the occluders and below the top edges of the view cell and $O$.*

line space. Each line of each portal in primal space corresponds to a halfspace bounded by a hyperplane in line space. A line in primal space that stabs all the portals corresponds to a point in line space which (1) is in the intersection of all these halfspaces, and (2) lies on the Plücker quadric (see [Tel92a]). If there is no such point, we conclude that there is no stabbing line and hence $O$ is not visible with respect to the given occluder sequence. The intersection of 5D halfspaces is calculated using polyhedra enumeration techniques [AF96].

### 5.2. Updating the line space subdivision

In order to update the line space subdivision, more detailed information about the visibility of $O$ is gathered by making use of the *antipenumbra* [Tel92a] (this is only necessary for occluders, not for occludees). The antipenumbra is the set of 3D rays corresponding to all the stabbing lines found by the 5D halfspace intersection. By reconstructing the antipenumbra induced by the given set of virtual portals and intersecting it with $O$, we determine the exact visible part of $O$ in the funnel $\mathcal{F}$. This leads to three possible results:

- $O$ is completely occluded. In this case, $O$ does not contribute to the line space cell $\mathcal{C}$.
- The top edge of $O$ is visible across the whole funnel. In this case, it is simply added to the sequence of relevant occluders for $\mathcal{C}$.
- $O$ is visible in only a part of the funnel. This means that a change of visibility due to the height structure of occluders occurs inside the funnel $\mathcal{F}$.

Only this last case requires the line space subdivision to be updated: The endpoints of the visible parts of $O$ are mapped to one or more edges in line space, which are used to subdivide the original line space cell $\mathcal{C}$. In the newly created cells where a part of $O$ is visible it is added to the associated sequence of occluders.

### 5.3. Acceleration using penumbra wedges

The 5D halfspace intersection necessary for the portal test is a rather costly algorithm with a high constant cost and asymptotic time complexity $O(n^2)$, where $n$ is the number of halfspaces [Tel92a]. Fortunately, we can apply conservative tests that quickly decide if the new occluder is either definitely visible or definitely invisible. As a result, the higher-dimensional algorithm is invoked very rarely in practice. The conservative tests are based on the construction of *penumbra wedges*, which bound the penumbra of the view cell with respect to an occluder and a funnel.

First, we select all occluders associated with the line space cell $\mathcal{C}$. For each occluder, the penumbra wedge is calculated as the region bounded by the four shadow planes defined by its top edge and the top edge of the view cell. Some typical penumbra wedges are shown in Figure 4.

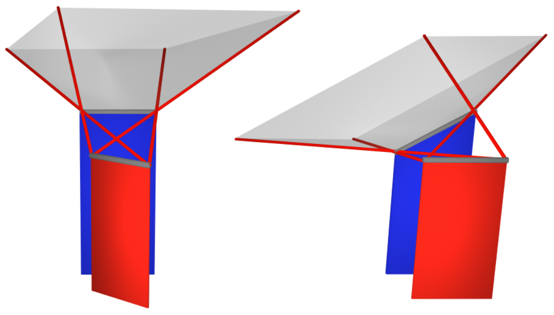

**Figure 4:** *Penumbra wedges for two different view cell face/occluder configurations. (left) Non-parallel edges induce penumbra wedge with non-zero volume. (right) The viewcell face and the occluder are parallel, but due to the tilted occluder edge the penumbra wedge expands.*

To determine visibility of $O$ with respect to $\mathcal{F}$, we first clip it against the two lower shadow planes of each penumbra wedge. If $O$ is completely *below any* two such planes, it is definitely invisible.

An occludee can be trivially accepted as visible in $\mathcal{C}$ if any part of its edge lies above all upper shadow planes. If this is not the case, or if we are dealing with an occluder, the upper shadow planes are used to compute the number of penumbra wedges that intersect the clipped occluder edge. If the edge intersects at most one penumbra wedge, it is visible (this happens if there is no or only one “relevant” occluder for $O$) and we calculate the visible part of $O$ by clipping against the lower shadow planes. However if the edge intersects several wedges, the umbra is defined by a quadratic EEE event surface due to penumbra fusion (see Figure 5). Only in this case do we invoke the exact stabbing line computation described above.

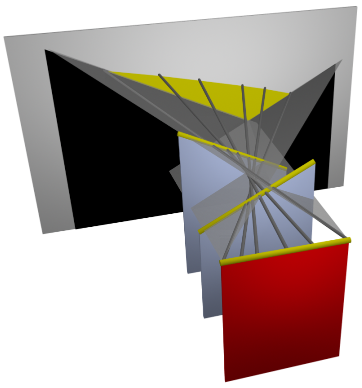

**Figure 5:** *A EEE event surface due to the cumulative effect of two occluders. The yellow region would be falsely classified as visible if only the penumbra wedges of the two occluders were used.*

## 6. Results and Discussion

We have implemented the presented algorithms and run some tests on a 2GHz Opteron PC. We will analyze the versatility, scalability and efficiency of our method on 4 different test scenes with different view cell sizes (test 1 and 2). We also compare aspects of our method against three different previous methods to show the effects of discretization [WWS00, LSCO03] (test 3) and to prove the performance advantage over exact 3D visibility evaluation [Bit02] (test 4).

### 6.1. Tests for different scenes

In a first test, we illustrate the *versatility* of our method by running it on four different test scenes with varying visibility properties: a model of the city of Vienna, which is a typical European city with dense occlusion; a model of the city of Atlanta, which has a much lower building density like most American cities, and therefore features visibility interactions that extend far from the view cell; a computer generated model of a large city with complex building shapes; and of a randomly generated 2.5D scene with slanting top edges, which makes the complex EEE events appear more often. All values in the following tables represent averages over a number of randomly placed view cells. Table 1 summarizes some model statistics. For the Vienna model, occludees consisted of bounding boxes containing a total of 8M polygons, whereas in the other models, the occluders also served as occludees. Note that in our experiments, we concentrate on the number of actual 2.5D occluders in the scene. As discussed in Section 3, these occluders typically represent significantly more complex 3D input scenes.

Snapshots of the PVS computed for one view cell in Atlanta, for view cells at different heights in the random scene, and a PVS at the City scene are depicted in Figures 7, 8, and 9. Figure 1 shows results for a very big scene, showing the scalability of the algorithm. The memory requirements for representing the line space subdivision for our the Atlanta scene are as follows: the subdivision consisted on average of 2312 cells for the larger view cells and 565 cells for the smaller view cells. There were 1.54 occluders associated per line space cell for the large view cells and 1.66 for the smaller view cells. This number expresses the average number of occluders visible above each other in a funnel (for a 2D scene it would equal 1.0).

| | Vienna | Atlanta | City | Random |
|---|---|---|---|---|
| Triangles | 8,123,675 | 181,678 | 2,153,382 | 42,801 |
| Occluders | 15,243 | 90,839 | 1,018,350 | 42,801 |
| Occludees | 7,731 | 90,839 | 1,018,350 | 42,801 |
| Area | 8 km$^2$ | 32 km$^2$ | 40 km$^2$ | 1 km$^2$ |

**Table 1:** *Model statistics.*

Table 2 shows the behavior of the algorithm when the view cell size varies. The algorithm is *output sensitive*, i.e., the running time increases when the complexity of visibility interaction increases for larger view cells (possibly faster than linearly), but is practically independent of the size of the scene. The output-sensitive behavior is mainly achieved due to the two hierarchies in the algorithm, the scene hierarchy and the hierarchical line-space subdivision.

| | view cell | | PVS size | time |
|---|---|---|---|---|
| | perim. | height | | $[s]$ |
| Vienna | 36 m | 2 m | 84 | 0.013 |
| | 535 m | 2 m | 235 | 0.060 |
| Atlanta | 17 m | 2 m | 279 | 0.055 |
| | 44 m | 2 m | 890 | 0.298 |
| | 89 m | 2 m | 1053 | 0.488 |
| | 177 m | 2 m | 1328 | 0.918 |
| City | 13m | 2 m | 1041 | 2.263 |
| | 569m | 2 m | 7050 | 70.2 |
| Random | 36 m | 2 m | 113 | 0.156 |
| | 36 m | 4 m | 194 | 0.438 |

**Table 2:** *Results for different view cells.*

The *efficiency* of the different stages of the algorithm is shown in Table 3. It can be seen that the penumbra acceleration succeeds in resolving visibility in the vast majority of cases, which explains the high speed of the algorithm. The remaining cases are run through the exact portal test, which classifies about 10-20% of the remaining occluders as visible, while not taking more than 25% of the total running time. This suggests that the EEE events—which are the most difficult visibility events to compute—are already very well approximated by the underlying 2D algorithm in conjunction with the penumbra wedge optimization.

| | Portal tests | Penumbra succ. | Stab. culls | Stab. time |
|---|---|---|---|---|
| Vienna | 11,293 | 99.7% | 8.7% | 30% |
| Atlanta | 51,704 | 99.99% | 6% | 5% |
| City | 12,083,009 | 99.99% | 10% | 7% |
| Random | 26,026 | 99.5% | 11.6% | 28% |

**Table 3:** *Algorithm efficiency, showing how often the penumbra wedge acceleration succeeded in determining visibility, how many of the remaining occluders were culled by the exact portal test, and the total time used by the exact portal test. For the statistics we used the most complex set of view cells defined for each model.*

## 6.2. Simulating conservative behavior

In a second test, we used our algorithm to simulate a typical conservative algorithm that only uses one penumbra wedge per occluder to approximate regions where EEE events occur. The resulting PVS increased by more than 47% for the Random scene for 2 m view cells, and more than 76% for 4 m view cells. This shows not only the effectiveness of our new exact algorithm especially in complicated visibility situations, but also its utility to analyze such situations.

## 6.3. Simulating discretization

In a third test, we used our method as a reference method to demonstrate the effects of the two kinds of *discretization* typically used in previous visibility algorithms: object space discretization [SDDS00, WWS00] and image [DDTP00] or line space discretization [KCCO01, LSCO03]. For *object space discretization*, we have applied Wonka et al.'s discrete hardware-accelerated approach to the Vienna model for the same view cells as our exact method. Occluders were conservatively discretized to a global 1400x1400 pixel grid (yielding cells of about 2x2m), which led to an overestimation of the PVS of 17% on average, while the running time was comparable. This method works well if the scene contains a sufficient number of larger occluders, while for the exact algorithm, the size of occluders is irrelevant. Note that for the Atlanta scene, the grid would already have to contain 4000x2000 pixels.

*Line space discretization* works differently: an occluder needs to block the whole funnel represented by a sample in the discretization in order to have any effect on the computation. This means that occluders near the view cell are treated with higher accuracy. We show this effect by simulating the distance-dependent occluder shrinking required by the line space method of Leyvand et al. [LSCO03]. Table 4 summarizes the results for several line-space resolutions with view cells similar to those used before.

Especially for the Atlanta scene, where many occluders are distant, a very high line-space resolution (more than 16000x16000) is required to achieve acceptable results. Another reason for these values is that due to the conservative discretization in line space, gaps appear between formerly connected occluders. These results indicate that there is a larger class of urban models where discretization-based methods are no longer competitive. For scenes where the occluders become smaller and move further away from the view cell the exact method will significantly outperform the discretization-based ones in terms of speed (due to high resolution requirements) and quality of the PVS (due to shrinking). .

| Grid res. | 512 | 1024 | 2048 | 16384 | exact |
|---|---|---|---|---|---|
| Vienna | 633 | 535 | 489 | 455 | 235 |
| Atlanta | 50744 | 25751 | 10082 | 2656 | 1152 |

**Table 4:** *PVS sizes for different line-space grid resolutions for Leyvand et al. [LSCO03] compared to the exact solution.*

## 6.4. Comparison to 3D method

In a fourth test, we prove the value of an exact 2.5D solution vs. a *full 3D implementation*. We ran an output-sensitive exact 3D visibility algorithm [Bit02] for the same view cells as our 2.5D algorithm. For the Vienna and Atlanta scenes, the 3D method took about 15 times as long to compute the same result as our 2.5D method, whereas in the Random scene it took almost 25 times as long. This means that for all applications that do not require a full 3D visibility solution (which are numerous), our exact 2.5D method offers all the advantages of an exact method, but at the low computational effort of a conservative 2.5D method.

We should also point out that even though our algorithm is exact, our current implementation is ultimately limited by floating point accuracy. Numerical robustness is an issue that is not trivial to solve. In our method, we have used normalization and epsilon-thresholds in order to deal with this issue, and found this to affect the visibility decision of a few polygons. The use of the BSP tree as the main data structure actually helps here, since the required binary splitting of 2D polygons can be implemented very robustly. The more problematic halfspace intersection is used primarily as an acceptance/rejection classifier, which significantly limits the propagation of numerical errors.

## 6.5. Complexity analysis

An accurate theoretical complexity analysis of the method is a difficult issue. First, it is hard to provide an accurate combinatorial model of real urban scenes; second, the method is based on several heuristics, the behavior of which is hard to analyze (kD-tree, hierarchical line space subdivision, penumbra wedges acceleration). We therefore present an analysis under an assumption about scene visibility which is also backed up by experimental measurements.

Let $n$ denote the total number of occluders and $k$ the number of visible occluders. A reasonable assumption about scene visibility in urban scenes is that the average number of occluders influenced (i.e., partially occluded) by each visible occluder is $O(1)$ (in both horizontal and vertical directions). Thus, each blocker polygon may be split $O(1)$ times due to the corresponding visibility events. Assuming that the resulting $O(k)$ blocker polygon fragments are uniformly distributed, the expected size of the line space BSP tree is $O(k \log k)$ (this follows from an analysis of randomized BSP construction for line segments in the plane [BKOS97]). Consequently, each leaf of the BSP tree is associated with $O(1)$ visible occluders. Inserting an occluder or testing its visibility is $O(\log k)$ in time (we traverse $O(1)$ paths from the root to the leafs).

To take into account the hierarchical visibility tests for all occluders (and occludees), let us assume that at each kD-tree leaf, there are $\Omega(1)$ occluders. It can be shown that the number of kD-tree nodes tested for visibility is between $O(k + \log n)$ and $O(k \log n)$ [Bit02]. The visibility test using the kD-tree node takes between $O(\log k)$ and $O(k)$. The first case corresponds to traversing a single path from the root to the leaf, the second case to the traversal of the whole tree. So in total, the algorithm complexity is between $O(k \log k + \log k \log n)$ and $O(k^2 \log n)$.

We provide experimental evidence about the derived complexity bounds by measuring the dependence of the the size of the line space BSP tree and the total number of funnel visibility tests on the number of visible occluders for the scenes and tests presented in Table 2. The measurements are depicted in Figure 6. The figure shows that both the size of the BSP tree and the number of funnel visibility tests roughly correspond to the derived complexity bounds. The only test which exhibits faster growth is the second City test (large view cells). We explain this mainly by the fact that the size of the view cells is so large that the assumption of partial occlusion of O(1) in horizontal direction does no longer hold.

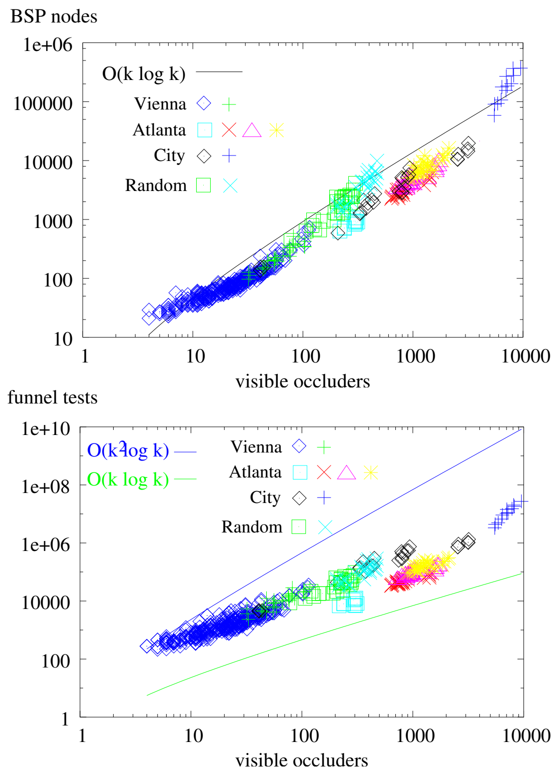

**Figure 6:** *Experimental evaluation of alghorithm complexity for all measured tests. (top) The size of the line space BSP tree in dependence on the number of visible occluders. (bottom) The total number of funnel visibility tests in dependence on the number of visible occluders. For better clarity both the graphs are shown in log-log scale.*

## 7. Conclusions and Future Work

In this paper, we have introduced the first exact solution to the 2.5D from-region visibility problem that achieves running times comparable to previous conservative methods. It combines a subdivision of 2D line space and an exact 3D portal visibility test in order to deal with all possible visibility interactions.

The main contribution of this paper is a solution to the 2.5D visibility problem that is fast and “just works”. It is exact and does not rely on any heuristic or simplification. In particular, it does not use discretization, neither of the scene nor of line space. We have shown possible consequences of such simplifications, where especially discretization methods have exhibited significant weaknesses in some situations. Also, no tuning of parameters is required, and the method works well for different sizes of occluders, no matter what distance they are located. All of this is important for practitioners from many fields who cannot easily judge how some simplification impacts their particular requirements for a visibility solution.

For future work, we plan to tackle the 3D visibility problem by decomposing ray space into subsets that can be analyzed with 2.5D visibility queries. The method could also be modified to efficiently handle non-urban 2.5D scenes, such as terrains.

## Acknowledgments

We would like to thank Pascal Müller for creating the City scene. This work was supported by the GameTools IST project no. IST-2-004363 and the Kontakt OE/CZ grant no. 2004-20.

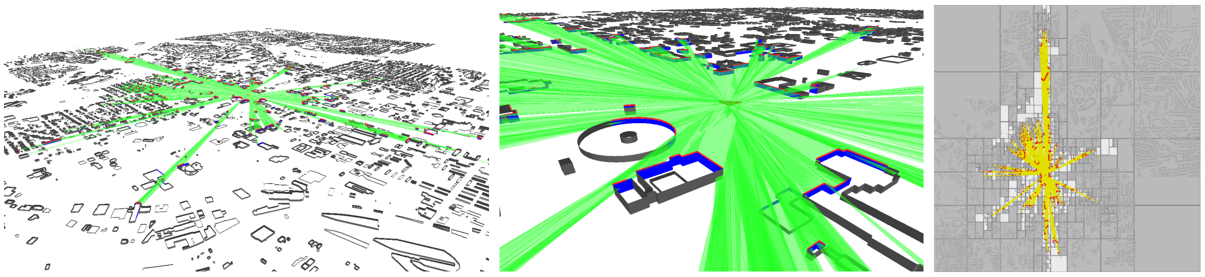

**Figure 7:** *(left) A PVS computed in the Atlanta scene. The buildings which form the PVS are shown in blue. The green planes correspond to lower penumbra wedges, the red line segments are visible parts of occluder top edges. (middle) A closeup of the view cell neighborhood. (right) Visualization of lines bounding the funnels of the line space subdivision (in yellow). Note the gray regions culled by hierarchical visibility tests.*

## References

[AF96] Avis D., Fukuda K.: Reverse search for enumeration. *Discrete Applied Math. 6* (1996), 21–46.

[ARB90] Airey J. M., Rohlf J. H., Brooks, Jr. F. P.: Towards image realism with interactive update rates in complex virtual building environments. In *1990 Symposium on Interactive 3D Graphics* (Mar. 1990), ACM SIGGRAPH, pp. 41–50.

[Bit02] Bittner J.: *Hierarchical Techniques for Visibility Computations*. PhD thesis, Czech Technical University in Prague, Oct. 2002.

[BKOS97] Berg M., Kreveld M., Overmars M., Schwarzkopf O.: *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, New York, 1997.

[BPS03] Bittner J., Přikryl J., Slavík P.: Exact regional visibility using line space partitioning. *Computers & Graphics 27*, 4 (2003), 569–580.

[BW03] Bittner J., Wonka P.: Visibility in computer graphics. *Environment and Planning B: Planning and Design 30*, 5 (sep 2003), 729–756.

[BWW01] Bittner J., Wonka P., Wimmer M.: Visibility preprocessing for urban scenes using line space subdivision. In *Proceedings of Pacific Graphics (PG'01)* (Tokyo, Japan, 2001), IEEE Computer Society, pp. 276–284.

[COCSD02] Cohen-Or D., Chrysanthou Y., Silva C., Durand F.: A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics.* (2002).

[DDTP00] Durand F., Drettakis G., Thollot J., Puech C.: Conservative visibility preprocessing using extended projections. In *Computer Graphics (Proceedings of SIGGRAPH 2000)* (2000), pp. 239–248.

[Dur99] Durand F.: *3D Visibility: Analytical Study and Applications*. PhD thesis, Universite Joseph Fourier, Grenoble, France, July 1999.

[KCCO01] Koltun V., Chrysanthou Y., Cohen-Or D.: Hardware-accelerated from-region visibility using a dual ray space. In *Proceedings of the 12th EUROGRAPHICS Workshop on Rendering* (2001).

[LSCO03] Leyvand T., Sorkine O., Cohen-Or D.: Ray space factorization for from-region visibility. *ACM Transactions on Graphics (TOG) 22*, 3 (2003), 595–604.

[NB04] Nirenstein S., Blake E.: Hardware accelerated aggressive visibility preprocessing using adaptive sampling. In *Rendering Technqiues 2004: Proceedings of the 15th symposium on Rendering* (2004), Eurographics Association, pp. 207–216.

[NBG02] Nirenstein S., Blake E., Gain J.: Exact From-Region visibility culling. In *Proceedings of EUROGRAPHICS Workshop on Rendering* (2002), pp. 199–210.

[SDDS00] Schaufler G., Dorsey J., Decoret X., Sillion F. X.: Conservative volumetric visibility with occluder fusion. In *Computer Graphics (Proceedings of SIGGRAPH 2000)* (2000), pp. 229–238.

[Tel92a] Teller S. J.: Computing the antipenumbra of an area light source. In *Computer Graphics (Proceedings of SIGGRAPH '92)* (July 1992), pp. 139–148.

[Tel92b] Teller S. J.: *Visibility Computations in Densely Occluded Polyhedral Environments*. PhD thesis, CS Division, UC Berkeley, Oct. 1992. Tech. Report UCB/CSD-92-708.

[TS91] Teller S. J., Séquin C. H.: Visibility preprocessing for interactive walkthroughs. In *Proceedings of SIGGRAPH '91* (July 1991), pp. 61–69.

[WWS00] Wonka P., Wimmer M., Schmalstieg D.: Visibility preprocessing with occluder fusion for urban walkthroughs. In *Proceedings of EUROGRAPHICS Workshop on Rendering* (2000), pp. 71–82.

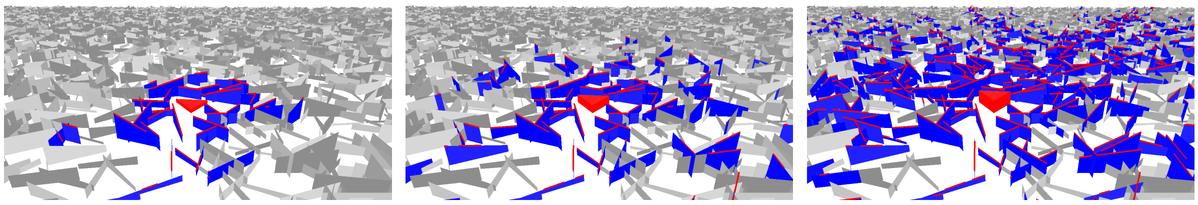

**Figure 8:** *A PVS computed in the random scene at different heights (left) 2 m, 72 visible objects, (middle) 4.6 m, 173 visible objects, (right) 7.2 m, 1554. visible objects.*

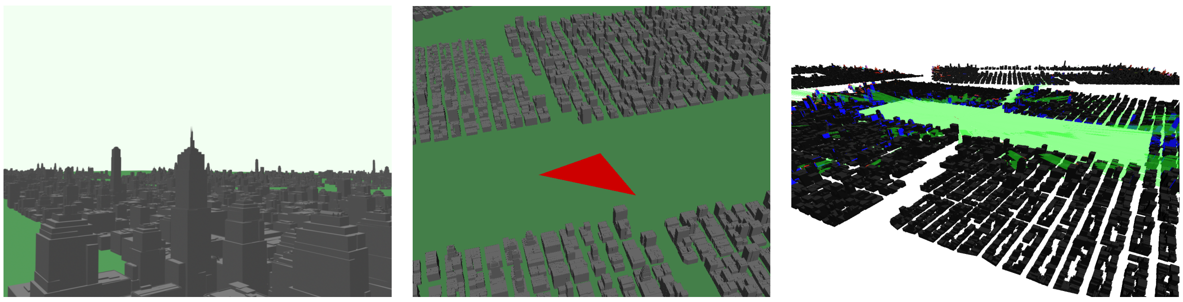

**Figure 9:** *(left) Overview of the City scene. (middle) A relatively large view cell inside the City scene used for our tests. Note that the view cell is located at a challenging position from which a large part of the city is visible. (right) Snapshot of a PVS computed in the City scene.*