Hardware-accelerated from-region visibility using a dual ray space

Vladlen Koltun[1], Yiorgos Chrysanthou[2], Daniel Cohen-Or[1]

[1] School of Computer Science, Tel Aviv University, Tel Aviv, Israel
{vladlen,dcor}@tau.ac.il
[2] Department of Computer Science, University College London, London, UK
y.chrysanthou@cs.ucl.ac.uk

Abstract. In this paper a novel from-region visibility algorithm is described. Its unique properties allow conducting remote walkthroughs in very large virtual environments, without preprocessing and storing prohibitive amounts of visibility information. The algorithm retains its speed and accuracy even when applied to large viewcells. This allows computing from-region visibility on-line, thus eliminating the need for visibility preprocessing. The algorithm utilizes a geometric transform, representing visibility in a two-dimensional space, the *dual ray space*. Standard rendering hardware is then used for rapidly performing visibility computation. The algorithm is robust and easy to implement, and can trade off between accuracy and speed. We report results from extensive experiments that were conducted on a virtual environment that accurately depicts 160 square kilometers of the city of London.

1 Introduction

In a remote walkthrough scenario, a large three-dimensional virtual environment is stored on a server. The client performs an interactive walkthrough, via a remote network connection, with no a priori information regarding the environment. The client is assumed to possess the computational resources equivalent to those of a personal workstation, not being able to render a significant portion of the environment in real time, nor even fit it into its memory.

This scenario brings about the need for *selective transmission*, the crux of which is that the server monitors the client's (virtual) location during the walkthrough, and transmits the relevant portions of the scene to the client. A selective transmission scheme must carefully balance between the necessity of not causing errors or visual artifacts on the client's side, and the desirability of real-time frame rates.

To keep the client's frame-rate high, the server has to ensure that the set of objects displayed by the client is as close to the set of visible objects as possible. Unfortunately, the ideal situation in which the client always renders only the visible objects is computationally infeasible. However, *conservative* visibility algorithms can be employed, which ensure that the client renders all the visible objects, as well as some occluded ones. Conservative visibility algorithms that perform well in practice have been the subject of intensive study for the past decade.

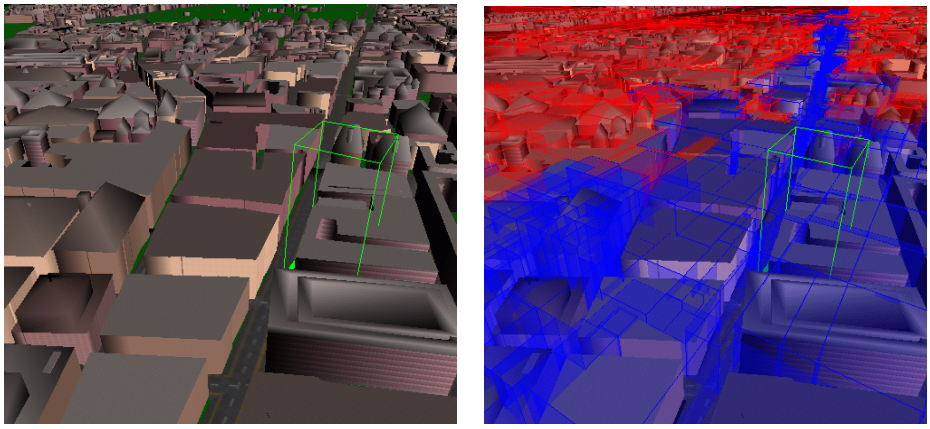


Fig. 1. The results of the algorithm for a viewcell of size 50x50x2 meters that partly lies on London's Oxford street. The model is shown on the left and the results are visualized on the right. The algorithm has detected that the whole length of Oxford street is visible (blue) from the viewcell (green), and was able to mark the immediately surrounding areas as occluded (red). The tall green wireframe box is drawn in order to mark the viewcell's location. Results for this viewcell are shown in Figure 6(a) as well.

From-point visibility algorithms [5, 12, 14, 30] are not suitable for remote walkthroughs, since they necessitate transmitting visibility changes every frame, giving rise to unacceptable communication lags, no matter how fast the visibility is computed. To avoid the problem of lag, from-region visibility [4, 8, 15, 22, 25, 29] can be used. The idea behind from-region visibility is to partition the space into a grid of viewcells. For each viewcell, the set of objects visible from at least one point inside the viewcell is conservatively estimated, yielding a *Potentially Visible Set* (PVS) that is associated with the viewcell.

While the client is in a certain viewcell, the server transmits the PVS of the adjacent viewcells. This alleviates lag, since the client does not have to wait every frame for updates from the server. Rather, the set of objects displayed by the client changes only when a viewcell boundary is crossed. By the time the client leaves a viewcell, the PVS of the next viewcell has already been transmitted, and the walkthrough proceeds smoothly.

This approach does not come without cost. Since the PVS of each viewcell contains information about the visibility of the whole scene (from that viewcell), and there is a large number of viewcells, the overall amount of visibility information that has to be stored on the server is enormous. Although compression schemes have been developed specifically to tackle this problem [11, 27], it is still very relevant and prohibitive for large scenes such as urban models.

The space problem is exacerbated by the fact that previous approaches to from-region visibility computation are efficient primarily when dealing with relatively small viewcells (with the exception of [9, 25, 26], which are dedicated to in-doors walk-

throughs). This implies that large scenes have to be partitioned into tens of thousands of viewcells, the Potentially Visible Sets of which have to be computed and stored.

Another bothersome aspect concerning small viewcells is the fact that the time it takes the client to cross one viewcell is short. If during that time the server cannot complete the transmission of the visibility change required for proceeding into an adjacent viewcell, visually disturbing errors occur. This is essentially the same problem of lag that occurs if the server uses from-point visibility computation. Note that the difference in visibility from two adjacent viewcells can be very significant, even if they are small.

In this paper, we introduce a different approach to computing from-region visibility, which eliminates the need for preprocessing and storing prohibitive amounts of visibility information, and does not introduce lag. The speed and accuracy of our from-region visibility algorithm are retained for very large viewcells. This allows utilizing from-region visibility computation on-line. While the client traverses one (large) viewcell, the server computes the visibility information for adjacent viewcells. The speed of the algorithm allows computing and transmitting this visibility information before the client reaches the next viewcell, and its accuracy (see Figure 1) ensures that the PVS is small enough to be displayed by the client in real time.

The next section surveys previous approaches to from-region visibility computation, and outlines the specific properties of our approach. Section 3 gives an overview of our algorithm, which is then developed in Sections 4 and 5. Implementation decisions and experimental results are reported in Section 6, and we conclude in Section 7.

2 From-region visibility

Detecting the objects that are visible from at least one point in a three-dimensional viewcell is inherently a non-linear four-dimensional problem [7, 24]. Exact solutions to the from-region visibility determination problem are considered impractical. In fact, no such solutions have explicitly appeared in computer graphics literature, with the exception of [7, 24]. Instead, researchers have concentrated on providing practical conservative algorithms that overestimate the set of visible objects.

For general scenes, conservative methods were introduced that take into account only the occlusion of individual large convex occluders [4, 21]. It was shown that such methods are only effective if the viewcells are smaller than the occluders [18]. For large viewcells, occlusion may arise out of the combined contribution of several objects. Often, a “cluster” of small objects occludes a large portion of the scene, while the individual occlusion of each object is insignificant.

New techniques have recently emerged that attempt to perform occlusion fusion, that is, capture occlusion caused by groups of objects [8, 15, 22, 29]. Durand et al. [8] perform from-region visibility computation by placing projection planes behind occluders, and projecting objects onto these planes. Koltun et al. [15] “slice” the scene and aggregate occlusion inside each slice to form large virtual occluders. Schaufler et al. [22] discretize the scene into a hierarchy of voxels and determine occlusion by testing the voxels against the umbrae of the occluders. Wonka et al. [29] prove that after the occluders are appropriately “shrunk”, sampling the visibility at discrete locations

on the boundary of the viewcell provides a conservative estimate of the visibility from the viewcell. Further discussion on these recent techniques can be found in [3].

The from-region visibility algorithm presented in this paper utilizes a geometric transform that maps rays in object-space to points in image-space. This alternative representation of visibility allows the problem to be conservatively discretized and solved rapidly, using standard rendering hardware. The resolution of the discretization can be adjusted to trade off between accuracy and speed. The algorithm has been successfully applied to viewcells that are much larger than individual occluders (see Figure 6(b)). Its accuracy and speed allow computing from-region visibility on-line, eliminating the need for visibility preprocessing. It thus provides inherent support for dynamic removal and addition of objects.

3 Overview

The algorithm processes a model that is represented by a hierarchical space subdivision; we use a kd-tree. Each node of this space subdivision is associated with an axis-aligned bounding box, and with the objects that are (perhaps partially) contained in this box. The bounding box associated with a node is completely contained in the bounding box associated with the node's parent.

For a given viewcell, our algorithm hierarchically traverses this subdivision in a top-down fashion. For each subdivision node, the algorithm determines whether the bounding box of the node is visible from the viewcell. When an occluded node is reached, the recursion terminates, since the children of that node are guaranteed to be occluded. This early termination strategy contributes to the speed of the algorithm, by allowing large portions of the scene to be culled after just one visibility determination query.

The cell-to-cell visibility determination algorithm is the core of the system. Denote the axis-parallel boxes corresponding to the viewcell and to some subdivision node by $\mathcal{A}$ and $\mathcal{B}$, respectively. Denote a collection of occluding objects by $\mathcal{S}$. The algorithm determines whether $\mathcal{A}$ and $\mathcal{B}$ are mutually visible among $\mathcal{S}$. $\mathcal{A}$ and $\mathcal{B}$ are said to be mutually visible if there exists a line segment with one end-point in $\mathcal{A}$ and another in $\mathcal{B}$ that is disjoint from all objects of $\mathcal{S}$. The cell-to-cell visibility determination algorithm operates by first reducing the problem to planar visibility determination as described in Section 4, and then solving this planar problem as described in Section 5.

Simplifying assumptions. There is no known rapid from-region visibility algorithm that is also exact. This does not seem likely to change due to the four-dimensional nature of the problem, and the complex geometric structure of visibility events [7]. Practical conservative algorithms [4, 8, 15, 22, 29] necessarily discretize or simplify the problem in some way. One common strategy, which is adopted here as well, is to assume that the input scene (more accurately, the set of occluders) is 2.5D [4, 15, 29].

Clearly, one can construct a scene in which this assumption is too restrictive for any significant occlusion to be detected. Such a scene may be, for example, a randomly generated "soup" of long and slim triangles. We consider such scenes to be of relatively little interest to the practical uses of the presented algorithm. A common type of input scenes for walkthrough systems is urban environments, where the most important occluders are buildings, large parts of which are 2.5D due to engineering constraints. In

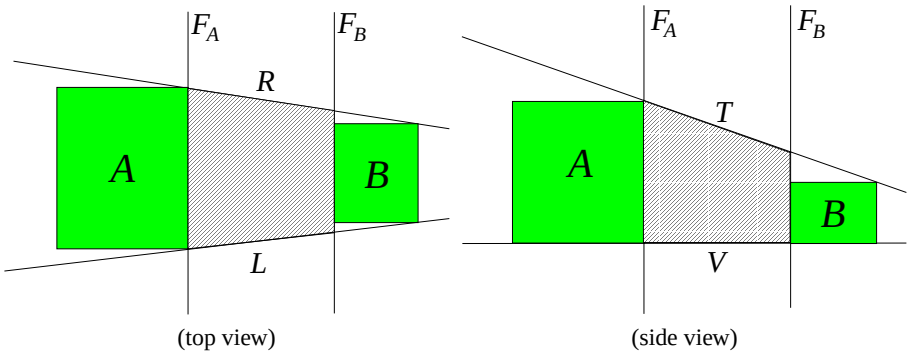


Fig. 2. The construction involved in the reduction to a planar problem. The shaft $\mathcal{AB}$ is shown filled.

architectural scenes, much of the occlusion is also caused by 2.5D objects. Even if a virtual environment is not specifically designed to model a city, a building, or a landscape, it is rare to come across a commercial virtual environment that is not largely 2.5D.

Moreover, "good" occluders can be synthesized from arbitrary input objects. Synthesis of large convex occluders was studied by Andujar et al. [1], and similar techniques can be applied to synthesize 2.5D occluders.

4 Reduction to a planar problem

The algorithm has to decide whether two axis-parallel boxes $\mathcal{A}$ and $\mathcal{B}$ are mutually visible. We now show how this problem can be reduced to *planar* visibility determination. We start with a simple observation. Denote the edges bounding the upper face of $\mathcal{A}$ ($\mathcal{B}$) by $\mathcal{A}_i$ (respectively, $\mathcal{B}_i$), for $1 \leq i \leq 4$.

Observation 1 *$\mathcal{A}$ and $\mathcal{B}$ are mutually visible with respect to $\mathcal{S}$ if and only if $\mathcal{A}_i$ and $\mathcal{B}_j$ are mutually visible with respect to $\mathcal{S}$, for some $1 \leq i, j \leq 4$. In other words, $\mathcal{A}$ and $\mathcal{B}$ are mutually visible if and only if their upper rims are.*

Proof. It is obvious that $\mathcal{A}$ and $\mathcal{B}$ are visible if their upper rims are. We thus concentrate on the "only if" part, stating that if $\mathcal{A}$ and $\mathcal{B}$ are mutually visible then their upper rims also are. $\mathcal{A}$ and $\mathcal{B}$ are mutually visible if and only if there is a visibility segment s between them. s is disjoint from $\mathcal{S}$ and has one end-point in the interior of $\mathcal{A}$ and another in the interior of $\mathcal{B}$. Consider the semi-infinite vertical "wall" defined as the union of upward vertical rays originating from s. Since the occluders are 2.5D and are disjoint from s, they are also disjoint from this wall. Also, the wall must contain at least one point belonging to the upper rim of $\mathcal{A}$ and another point belonging to the upper rim of $\mathcal{B}$. Since it is convex, it also contains the segment connecting these two points. Thus, there exists a segment connecting the upper rims of $\mathcal{A}$ and $\mathcal{B}$ that is disjoint from $\mathcal{S}$. □

We now define the term *shaft*; the definition is illustrated in Figure 2. Let L and R be the two vertical planes that support $\mathcal{A}$ and $\mathcal{B}$. Let F_A and F_B be two parallel vertical

planes that separate $\mathcal{A}$ and $\mathcal{B}$, such that F_A contains a face of $\mathcal{A}$ and F_B contains a face of $\mathcal{B}$. Let T be a plane that contains two parallel horizontal edges, one of $\mathcal{A}$ and one of $\mathcal{B}$, and supports $\mathcal{A}$ and $\mathcal{B}$ from above. (In general, there are two such planes.) Let V be the plane that supports $\mathcal{A}$ and $\mathcal{B}$ from below. The shaft $\mathcal{AB}$ is the area bounded by L, R, F_A, F_B, T, and V. (Notice that our definition of a shaft is different from that in [13].)

Observation 1 implies that there exists a visibility ray between $\mathcal{A}$ and $\mathcal{B}$ inside the shaft $\mathcal{AB}$ only if there exists a visibility ray on the "ceiling" of $\mathcal{AB}$. More precisely, let a_1 denote the point $T \cap F_A \cap L$, and let a_2 denote the point $T \cap F_A \cap R$. Similarly, $T \cap F_B \cap L$ is denoted by b_1, and $T \cap F_B \cap R$ is denoted by b_2. Also, denote by $\mathcal{S}_T$ the collection of intersections of the polygons of $\mathcal{S}$ with T, which is a collection of segments on T. $\mathcal{A}$ and $\mathcal{B}$ are mutually visible among $\mathcal{S}$ only if the segments (a_1, a_2) and (b_1, b_2), both lying on the plane T, are mutually visible among $\mathcal{S}_T$. We have thus reduced the problem to planar visibility determination on T.

5 Planar visibility determination

Given two segments s_1 and s_2 in the plane, and a collection of occluding segments O, we wish to determine whether s_2 is visible from s_1. We first provide a simple analytic algorithm for this problem, which is then converted into a rapid hardware-assisted one.

5.1 Exact analytic algorithm

We define a bounded two-dimensional space, the *dual ray space*, such that every ray originating on s_1 and intersecting s_2 corresponds to a point in this space. Our algorithm "marks" all points in the ray space that represent rays that pass through occluding segments. Visibility is then detected by checking whether there is at least one point that has not been "marked".

More precisely, parameterize s_1 and s_2 as $\{s_1(t) | \, 0 \leq t \leq 1\}$ and $\{s_2(t) | \, 0 \leq t \leq 1\}$, respectively. Let $\mathcal{RS}$ be the unit square $\{(x, y) | \, 0 \leq x, y \leq 1\}$, such that a point (x, y) in $\mathcal{RS}$ corresponds to the ray originating at $s_1(x)$ and passing through $s_2(y)$.

Define a mapping $\mathcal{T} : \mathbb{R}^2 \rightarrow \mathcal{RS}$ that maps each point $p \in \mathbb{R}^2$ to the collection of points in $\mathcal{RS}$ that correspond to rays passing through p. For any $p \in \mathbb{R}^2$, $\mathcal{T}(p)$ is a line segment in $\mathcal{RS}$. For a segment $v \in O$, parameterized as $\{v(t) | \, 0 \leq t \leq 1\}$, $\mathcal{T}(v)$ is defined to be the continuous collection of segments $\{\mathcal{T}(v(t)) | \, 0 \leq t \leq 1\}$. This collection is bounded by the segments $\mathcal{T}(v(0))$ and $\mathcal{T}(v(1))$ that correspond to the end-points of v. In general, it forms either a trapezoid (Figure 3(a)) or a double-triangle (Figure 3(b)), depending on whether the line containing v intersects the interior of s_1 or not.

This implies a simple exact algorithm for determining whether s_2 is visible from s_1: Map each segment $v \in O$ to $\mathcal{RS}$ and compute the union of the resulting trapezoids and double-triangles (i.e. $\bigcup_{v \in O} \mathcal{T}(v)$). This computation can be performed in worst-case optimal $O(n^2)$ time without employing complex data structures [6]. If s_1 and s_2 are mutually visible, there is a point $p_o \in \mathcal{RS}$ that is not contained in this union. The point p_o corresponds to a visibility ray (see Figures 3(c) and 3(d)).

The dual ray space mapping $\mathcal{T}$ bears similarities to other duality transforms, such as the standard duality transform in computational geometry [6] and the Hough transform,

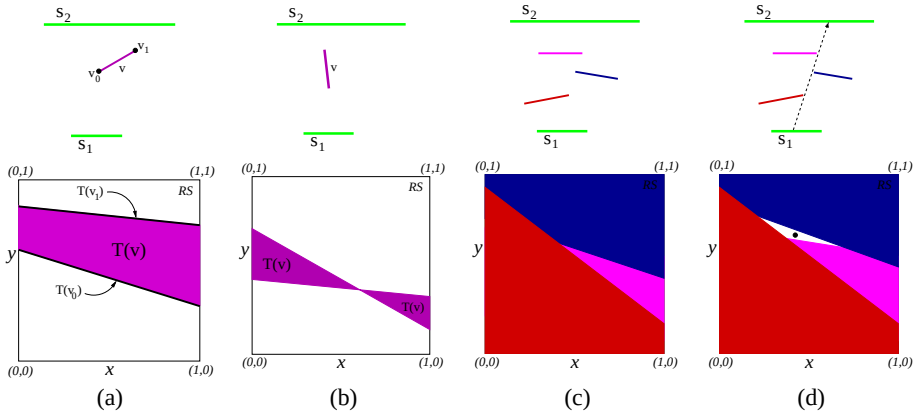


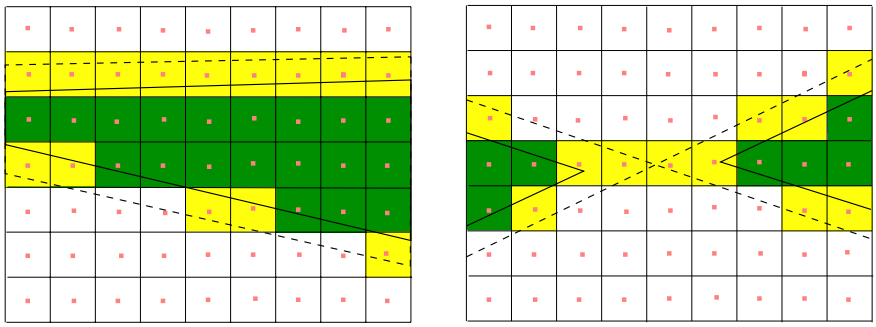
Fig. 3. Simple scenes (top) and their dual ray space (bottom). In this figure, each trapezoid in the ray space has the color of the segment it corresponds to. (a) and (b) each show an occluding segment. s_2 is occluded from s_1 in (c) and is visible in (d); the black point in the ray space of (d) corresponds to the dashed visibility ray.

which is used for line detection in image analysis [16]. However, the dual ray space is a bounded region (as opposed to the infinite dual planes of the above-mentioned transforms) that can be efficiently discretized. This is a crucial advantage that served as the main motivation for the current definition of the dual ray space. In this sense, the dual ray space is similar to the lumigraph [10] and the light field [17]. (Notice that Gortler et al. [10] also use the term "ray space".) In the context of related work in computational geometry, our exact algorithm corresponds to local computation of one face of the *visibility complex* [20]. The visibility complex has been previously applied to ray-tracing [2] and radiosity [19].

5.2 Hardware-accelerated algorithm

We wish to determine whether $\bigcup_{v \in O} \mathcal{T}(v)$ covers the unit square $\mathcal{RS}$. This can be accomplished conservatively by discretizing $\mathcal{RS}$ into a bitmap and rendering all $\mathcal{T}(v)$ onto this bitmap using graphics hardware. All $\mathcal{T}(v)$ are drawn in white, without z-buffering or shading, onto an initially black background. If a black pixel remains, the segments s_1 and s_2 are reported to be mutually visible. This algorithm avoids the complex analytic computation of the union and alleviates robustness problems common in geometric algorithms.

The default behavior of OpenGL is to color a pixel if its *center* is inside a drawn polygon. This may cause inaccuracy, since our algorithm is conservative provided that only the pixels that are *completely* covered by $\bigcup_{v \in O} \mathcal{T}(v)$ are colored white. This behavior is ensured by "shrinking" the double-triangles and trapezoids prior to rendering. Their edges are moved inward by $\sqrt{2}a$, where a is half the pixel size (see Figure 4). The center of a pixel is inside $\mathcal{T}(v)$ after shrinking, only if the pixel was completely covered by it prior to shrinking [28].



$\mathcal{T}(v)$ is a trapezoid $\mathcal{T}(v)$ is a double-triangle

Fig. 4. The green pixels are properly contained in $\mathcal{T}(v)$ (shown dashed), and their centers are contained in the "shrunk" $\mathcal{T}(v)$ (shown solid). The yellow pixels are only partially covered by $\mathcal{T}(v)$, but their centers are contained in $\mathcal{T}(v)$; shrinking prevents them from being colored.

The process of checking whether there are any black pixels left after rendering is accelerated using the OpenGL minmax operation, part of the OpenGL 1.2 standard. The minmax operation allows a quick determination of the minimal (or maximal) pixel color in the frame buffer. It can therefore be used for deciding whether the buffer is fully white or contains a black pixel.

The image-space nature of the described algorithm allows it to be applied hierarchically, in a manner similar to [30]. We can use a low-resolution bitmap to represent $\mathcal{RS}$, and rapidly render all $\mathcal{T}(v)$ onto it. If the resulting bitmap is not fully white, a bitmap of higher resolution is used to refine the test, yielding a less conservative PVS.

6 Results

The algorithms described in this paper were implemented, and tested on an IBM A20p laptop with a 750Mhz Pentium III CPU and an ATI Rage Mobility graphics card. The exact planar visibility algorithm (Section 5.1) was also implemented, for the sake of comparison. Our test model accurately depicts 160 sq. km. of London. It was created using Ordnance Survey data, and comprises of about 1.7 million polygons [23].

One goal of our experiments was to determine an effective resolution for the discretization of the dual ray space. There is a clear trade-off involved. Higher resolution yields a less conservative PVS, at the cost of computation speed. Another goal was determining an effective viewcell size. Small viewcells induce lag, but prohibitively large portions of the scene are visible from viewcells that are too large.

We have tested the algorithm with viewcells of sizes ranging from 50x50 meters to 300x300 meters, and with discretization resolutions ranging from 16x16 to 256x256. All the viewcells were 2 meters high. The average time to compute the PVS is shown in Table 1, for a sample of viewcell sizes and discretization resolutions. Table 1 also shows the impact of the discretization resolution on the overestimation of the PVS. The

Resolution	Time (sec.)			Overestimation (%)		
	Viewcell Size			Viewcell Size		
	50x50	100x100	300x300	50x50	100x100	300x300
16x16	0.4	0.5	1.8	0.391	0.518	1.192
32x32	0.6	0.7	2.7	0.335	0.436	0.901
64x64	0.9	1.1	4.1	0.185	0.240	0.533
128x128	2.1	2.5	7.8	0.012	0.026	0.106
256x256	6.6	7.9	24	0.001	0.005	0.019

Table 1. The effect of the discretization resolution on the speed and overestimation of the algorithm.

overestimation is given by

$$\frac{P_E - P_C}{P_E} \cdot 100\%,$$

where P_E is the size of the occluded areas, as computed using the exact planar algorithm, and P_C is the size of the occluded areas, as computed using the conservative planar algorithm. One of the advantages of this measure is its independence from the depth complexity of the model.

Table 1 shows that the speed of the algorithm directly depends on the discretization resolution. Hence, using better graphics hardware can further accelerate the visibility computation. This indicates that we have made the from-region visibility determination problem hardware intensive.

Based on our experiments, we have chosen to work with viewcells of size 100x100 meters and discretization resolution of 128x128. The PVS of a 100x100 viewcell in the London model consists of on average 8K polygons (0.5% of the model). This means that a client using a personal workstation can render the PVS in real time.

Before the walkthrough begins, the server computes and sends the PVS of the initial viewcell and the eight viewcells adjacent to it. During the walkthrough, the server ensures that the client receives the PVS of a viewcell before reaching it. Since the server does not know in advance which viewcell the client will enter next, this necessitates computing and transmitting the PVS of up to five adjacent viewcells while the client traverses a single viewcell, as shown in Figure 5. Assuming average walkthrough speed of 6km/h, such computation takes 12.5 seconds on average (Table 1), which leaves 47 seconds for transmitting the five visibility changes to the client, considering the fact that crossing a single 100x100 viewcell at this speed takes about one minute.

Even in the case of a slow network connection, this clearly shows that remote walkthroughs can be conducted unhindered by network lag. The large size of the viewcells gives the server enough time to compute and transmit the visibility information. The algorithm's speed allows performing visibility computation on-line. Finally, the accuracy of the algorithm ensures that the PVS is small enough to be displayed by the client in real time.

7 Conclusion

We have presented a novel from-region visibility algorithm. Its central idea is an alternative representation of visibility in the *dual ray space*, which allows utilizing standard

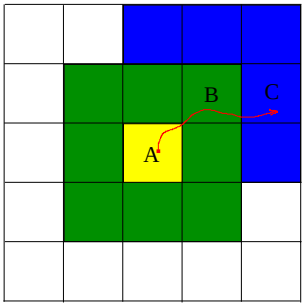


Fig. 5. A client's path in the model is shown in red. When the client commences the walkthrough at viewcell A, it has the PVS of A and the green viewcells. Upon the client's arrival at viewcell B, the server starts computing the PVS of the blue viewcells. Experiments show that the client will receive the PVS of all the blue viewcells before reaching any one of them. In similar fashion, the server ensures throughout the walkthrough that the client never reaches a viewcell before its PVS is received by the client.

rendering hardware for visibility determination. The algorithm is simple to implement, is robust due to working primarily in image-space, and can trade off between accuracy and speed. It was shown to be efficient even when applied to large viewcells. This allows remote walkthroughs to be conducted without preprocessing and storing a priori visibility information.

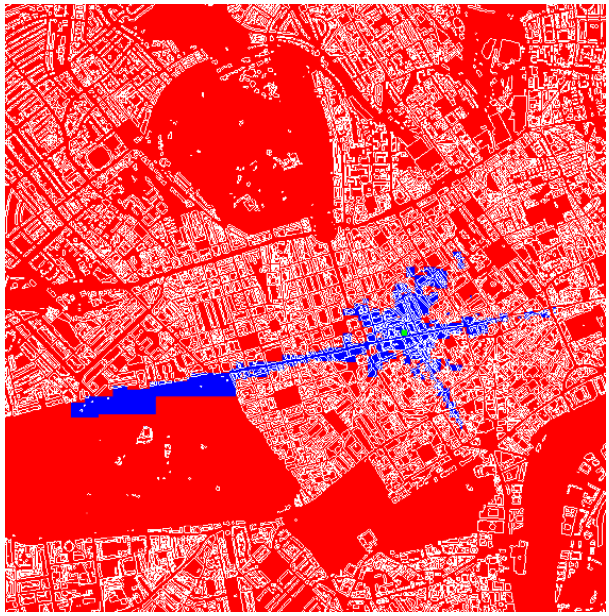
Acknowledgements

This work was supported by grants from the Israeli Academy of Sciences (center of excellence), and from the Israeli Ministry of Sciences. The London model was generated by the COVEN ACTS project on the Cities Revealed data from the GeoInformation Group.

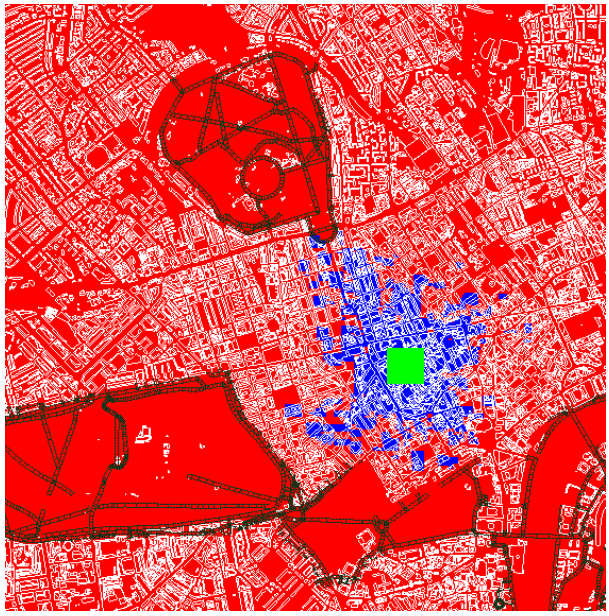
References

1. C. Andujar, C. Saona-Vazquez, and I. Navazo. LOD visibility culling and occluder synthesis. *Computer Aided Design*, 32(13):773–783, November 2000.
2. F. S. Cho and D. Forsyth. Interactive ray tracing with the visibility complex. *Computer & Graphics*, 23(5):703–717, 1999.
3. D. Cohen-Or, Y. Chrysanthou, and C. Silva. A survey of visibility for walkthrough applications. *SIGGRAPH 2000 'Visibility: Problems, Techniques, and Applications' course notes.*
4. D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum*, 17(3):243–254, 1998.
5. S. Coorg and S. Teller. Real-time occlusion culling for models with large occluders. *1997 Symposium on Interactive 3D Graphics*, pages 83–90, April 1997.
6. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

7. F. Durand. *3D Visibility: analytical study and applications*. PhD thesis, Université Joseph Fourier, Grenoble I, July 1999.
8. F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. *Proceedings of SIGGRAPH 2000*, pages 239–248, July 2000.
9. T. Funkhouser. Database management for interactive display of large architectural models. *Graphics Interface*, pages 1–8, May 1996.
10. S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. *Proceedings of SIGGRAPH 96*, pages 43–54, August 1996.
11. C. Gotsman, O. Sudarsky, and J. Fayman. Optimized occlusion culling. *Computer & Graphics*, 23(5):645–654, 1999.
12. N. Greene and M. Kass. Hierarchical z-buffer visibility. *Proceedings of SIGGRAPH 93*, pages 231–240, 1993.
13. A. E. Haines and J. R. Wallace. Shaft culling for efficient ray-traced radiosity. *2nd Eurographics Workshop on Rendering*, pages 122–138, 1994.
14. T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frusta. In *Proceedings of the 13th Symposium on Computational Geometry*, pages 1–10, June 1997.
15. V. Koltun, Y. Chrysanthou, and D. Cohen-Or. Virtual occluders: An efficient intermediate PVS representation. *11th Eurographics Workshop on Rendering*, pages 59–70, 2000.
16. V. F. Leavers. Which Hough transform? *Computer Vision and Image Understanding*, 58(2):250–264, 1993.
17. M. Levoy and P. Hanrahan. Light field rendering. *Proceedings of SIGGRAPH 96*, pages 31–42, August 1996.
18. B. Nadler, G. Fibich, S. Lev-Yehudi, and D. Cohen-Or. A qualitative and quantitative visibility analysis in urban scenes. *Computer & Graphics*, 23(5):655–666, 1999.
19. R. Orti, S. Riviére, F. Durand, and C. Puech. Radiosity for dynamic scenes in flatland with the visibility complex. *Computer Graphics Forum*, 15(3):237–248, August 1996.
20. M. Pocchiola and G. Vegter. The visibility complex. *International Journal on Computational Geometry and Applications*, 6(3):279–308, 1996.
21. C. Saona-Vazquez, I. Navazo, and P. Brunet. The visibility octree: A data structure for 3D navigation. *Computer & Graphics*, 23(5):635–644, 1999.
22. G. Schaufler, J. Dorsey, X. Decoret, and F. X. Sillion. Conservative volumetric visibility with occluder fusion. *Proceedings of SIGGRAPH 2000*, pages 229–238, July 2000.
23. A. Steed, E.Frecon, D. Pemberton, and G. Smith. The london travel demonstrator. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST-99)*, pages 150–157, Dec. 1999.
24. S. J. Teller. Computing the antipenumbra of an area light source. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):139–148, July 1992.
25. S. J. Teller. *Visibility cmputations in dnsely ocluded plyhedral evironments*. PhD thesis, Dept. of Computer Science, University of California, Berkeley, 1992.
26. S. J. Teller and P. Hanrahan. Global visibility algorithms for illumination computations. *Proceedings of SIGGRAPH 93*, pages 239–246, August 1993.
27. M. van de Panne and J. Stewart. Effective compression techniques for precomputed visibility. *10th Eurographics Workshop on Rendering*, pages 305–316, 1999.
28. P. Wonka and D. Schmalsteig. Occluder shadows for fast walkthroughs of urban environments. *Computer Graphics Forum*, 18(3):51–60, September 1999.
29. P. Wonka, M. Wimmer, and D. Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. *11th Eurographics Workshop on Rendering*, pages 71–82, 2000.
30. H. Zhang, D. Manocha, T. Hudson, and K. Hoff. Visibility culling using hierarchical occlusion maps. *Proceedings of SIGGRAPH 97*, pages 77–88, August 1997.



(a) A viewcell of size 50x50x2



(b) A viewcell of size 300x300x2

Fig. 6. Results of two experiments. Overviews of 25 square kilometers of the London model are shown, with outlines of the buildings in white. The algorithm has classified the red areas as occluded from the green viewcell, which is 50x50x2 meters large in (a), and 300x300x2 meters large in (b). Discretization resolution of 128x128 was used.