Exact Regional Visibility using Line Space Partitioning

Jiří Bittner [a,b,*] Jan Přikryl [b,d] Pavel Slavík [c]

[a] *Center for Applied Cybernetics, Czech Technical University in Prague, Karlovo náměstí 13, 121 35 Praha 2, Czech Republic*

[b] *Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11, A-1040 Wien, Austria.*

[c] *Department of Computer Science, Czech Technical University in Prague, Karlovo náměstí 13, 121 35 Praha 2, Czech Republic*

[d] *Department of Applied Mathematics, Czech Technical University in Prague Na Florenci 25, 110 00 Praha 1, Czech Republic.*

Abstract

We present a new method for computing visibility from a polygonal region in the plane considering a set of line segments as occluders. The proposed method provides a comprehensive description of visibility from the given region. We represent sets of occluded rays using a hierarchical partitioning of dual space (line space). The line space partitioning is maintained by a BSP tree that provides efficient operations on the sets of lines. The implementation shows that the method is suitable for computing potentially visible sets in large scenes with various visibility characteristics.

Key words: Visible surface algorithms, Geometric algorithms, Spatial hierarchies

1 Introduction

Visibility is essential for many techniques of image synthesis and therefore it has been studied intensively in the past. The traditional algorithms solve the problem of visibility *from a point* (e.g. z-buffer), or visibility *along a line* (ray shooting). The current research focuses on computing visibility *from a region* (also called regional or from-region visibility) and *global* visibility. Regional

[*] Corresponding author. Favoritenstrasse 9-11, A-1040 Wien, Austria. Tel: +43 (1) 58801-18688, fax +43 (1) 58801-18698, email: bittner@cg.tuwien.ac.at

visibility algorithms capture visibility along each ray intersecting the given region, global visibility algorithms capture visibility along all rays intersecting the scene. These methods are crucial for soft shadow generation, global illumination, and real-time walkthroughs.

Our method is primarily targeted at walkthrough applications where visibility preprocessing by means of regional visibility is commonly used to accelerate rendering of large densely occluded scenes. In a preprocessing step the scene is subdivided into regions (view cells) and for each view cell we determine a *potentially visible set* of objects (PVS). In real time only the objects from the PVS are rendered for any view point inside the current view cell.

Many complex real world scenes used in computer graphics have largely 2D or 2.5D nature. Consider for example a building interior where the structure of the scene is predominantly given by the ground plan. Outdoor urban scenes can often be considered a 2D height function [1]. Consequently, the corresponding visibility computations can either be reduced to a set of 2D visibility problems [2] or solved by specialized algorithms tailored to the scene structure [3,1]. Figure 1 shows a PVS computed for a view cell in a 2D footprint of a large urban scene.

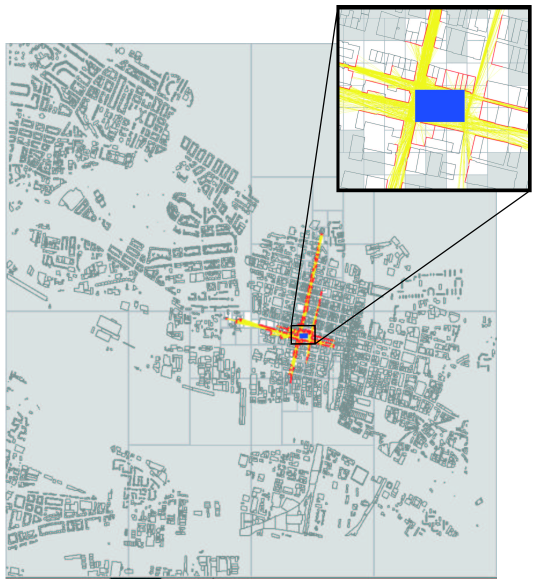


Fig. 1. Illustration of the from-region visibility in 2D. The scene represents a 2D footprint of 25km^2 of Glasgow city containing 94460 line segments. The PVS formed by the occluders visible from the given region is shown in red. The yellow lines depict visibility discontinuities.

Recently, Bittner et al. [4] presented an efficient visibility preprocessing method for 2.5D urban scenes that is based on a 2D regional visibility algorithm. In this paper we thoroughly discuss formal aspects of the 2D visibility algorithm used as a core of the 2.5D method [4]. The presented method determines visibility from a polygonal region in the plane considering a set of line segments as occluders. We provide an analysis of 2D regional visibility using dual space analogy and discuss an efficient hierarchical method for representing occlusion and testing visibility. In contrast to conservative visibility preprocessing algorithms [5,1] our technique is exact and yields comprehensive description of visibility from a given region. The algorithm accounts for all types of occluder fusion [1,6] and it can easily handle large regions. The algorithm can be applied to scenes with degenerate occluder configurations (overlapping and intersecting occluders) and does not require explicit knowledge of occluder connectivity. The method exhibits output sensitive behavior in practice [7], i.e. its running time is proportional to the number of visible objects.

2 Related work

Visibility problems are studied in computer graphics, computer vision, robotics, and other research areas. A comprehensive interdisciplinary survey was published by Durand [8]. More recently, Bittner and Wonka [9] reviewed visibility problems and algorithms in computer graphics, and Cohen-Or et al. [10] surveyed visibility algorithms for walkthrough applications. We first discuss visibility algorithms for 3D scenes and then we review related 2D visibility methods.

Computing exact regional visibility in 3D scenes is a very demanding task. Plantinga and Dyer [11] partitioned the scene into regions with topologically equivalent views (aspects). Teller [12], Drettakis and Fiume [13], and Stewart and Ghali [14] dealt with the 3D regional visibility in the context of computation of shadow boundaries. Durand et al. [15] proposed the visibility skeleton that captures all critical visual events. The currently known exact methods are not directly applicable to large scenes due to their computational complexity and numerical robustness problems.

Visibility culling techniques have been introduced to speedup rendering of large scenes where only a fraction of the scene is actually visible [10]. Generally, we can distinguish between from-region and from-point visibility culling algorithms. The from-point visibility algorithms require recomputation of visibility for each change of the view point. On the contrary, the from-region visibility algorithms precompute visibility for each view point of the given spatial region.

Greene et al. [16] proposed the hierarchical z-buffer that is a general discrete from-point visibility algorithm. Zhang et al. [17] developed a similar method that uses a hierarchical occlusion map and a depth estimation buffer. Continuous from-point visibility algorithms were proposed by Luebke and Georges [18], Coorg and Teller [19], Manocha et al. [20], and Bittner et al. [21].

Airey et al. [22] partitioned an indoor scene into cells and portals and for each cell they computed an approximate PVS by identifying objects visible through portal sequences. Teller and Séquin [3] developed a conservative variant of the cell/portal visibility algorithm. The cell/portal techniques are restricted to scenes with a natural cell/portal subdivision and cannot be applied for outdoor urban scenes. Cohen-Or et al. [23] used ray shooting to sample occlusion due to single convex occluder. Schaufler et al. [6] used blocker extensions to handle occluder fusion, i.e. occlusion due to combined effect of multiple occluders. Durand et al. [24] handled occluder fusion by extended occluder projections and occlusion sweep. Wonka et al. [1] used occluder shadows for visibility preprocessing in 2.5D scenes.

2D visibility was studied intensively in computational geometry as well as in computer graphics. The visibility graph [25] is a well known structure for capturing visibility in 2D scenes. Vegter [26] introduced the visibility diagram containing more information than the visibility graph. Later, Pocchiola and Vegter [27] proposed a similar structure called the visibility complex. The visibility complex for polygonal scenes was studied by Riviere [28]. Hinkenjann and Müller [29] developed the hierarchical blocker trees, i.e. a discrete structure similar to the visibility complex.

Our method is mostly related to the visibility diagram [26] and the visibility complex [27,28]. We construct a hierarchical representation of a 2D cross-section of the visibility complex involving objects visible from the given region. In terminology of Vegter [26] our method provides a hierarchical representation of the visibility function.

3 Algorithm overview

To compute visibility from a convex polygonal region we incrementally build the *occlusion tree* that associates with each occluded ray emerging from the region an occluder (line segment) that it first intersects.

The algorithm uses a kD-tree to organize the occluders; each leaf of the kD-tree stores a list of occluders that intersect the corresponding region. The kD-tree is used to generate an approximate front-to-back order of occluders with respect to the given region. For each occluder, a line space *blocker polygon* is constructed that represents rays intersecting the region and the occluder. The blocker polygon is then inserted in the occlusion tree. The insertion yields parts of the occluder that are currently visible. If the occluder is invisible, the tree remains unmodified. The traversal of the kD-tree is interleaved with testing visibility of its nodes using the current occlusion tree. If a node is classified invisible, its whole subtree and the corresponding occluders are culled.

The rest of the paper is organized as follows: Section 4 discusses lines intersecting a set of occluders and shows the corresponding configurations in dual space. Section 5 describes the occlusion tree and algorithms for its construction. Section 6 presents visibility tests using the occlusion tree. Section 7 outlines the complete hierarchical visibility algorithm. In Section 8 we evaluate our implementation of the proposed methods. Finally, Section 9 concludes.

4 Sets of lines

Visibility is a phenomenon that can be defined by means of mutually unoccluded points: Two points are mutually visible if the *line segment* connecting them is unoccluded. From this definition we can observe that visibility is carried by lines. Visibility from a given set of points is given by visibility along lines intersecting these points. Our algorithm uses mapping of oriented 2D lines to points in dual space, the *line space.* Such a mapping allows to handle sets of lines much easier than in the primary space [27].

4.1 Parametrization of lines

To parametrize oriented 2D lines we use a 2D projection of Plücker coordinates [30]. This parametrization corresponds to an "oriented homogeneous form" of the duality between points and lines in 2D [30]. Let l be an oriented line in E^2 and let $u = (u_x, u_y)$ and $v = (v_x, v_y)$ be two distinct points lying on l. Line l oriented from u to v can be described by the following matrix:

$$M_l = \begin{pmatrix} u_x & u_y & 1 \\ v_x & v_y & 1 \end{pmatrix}$$

Plücker coordinates l^* of l are minors of M_l:

$$l^* = (u_y - v_y, v_x - u_x, u_x v_y - u_y v_x).$$

l^* can be interpreted as homogeneous coordinates of a point in 2D *oriented projective space* P^2. Two oriented lines are equal if and only if their Plücker coordinates differ only by a positive scale factor. l^* also corresponds to coefficients of the implicit description of a line: l' expressed as $ax + by + c = 0$ induces two oriented lines l_1, l_2, with Plücker coordinates $l_1^* = (a, b, c)$ and $l_2^* = -(a, b, c)$.

Homogeneous coordinates are often normalized, e.g. $l_N^* = (a/b, 1, c/b)$. The normalization introduces a singularity: In our example vertical lines map to points at infinity. To avoid singularities we embed P^2 in E^3. Consequently, l^* represents a half-line in E^3 emerging from the origin. All points in E^3 lying on the half-line l^* represent the same oriented line l.

To sum up: An oriented line in 2D is mapped to a half-line beginning at the origin in 3D. An example of the concept is depicted in Figures 2-(a) and 2-(b). For the sake of clarity we will use 2D illustrations of line space (such as in Figure 2-(c)). We will talk about planes and half-lines, but they will be depicted as lines and points, respectively.

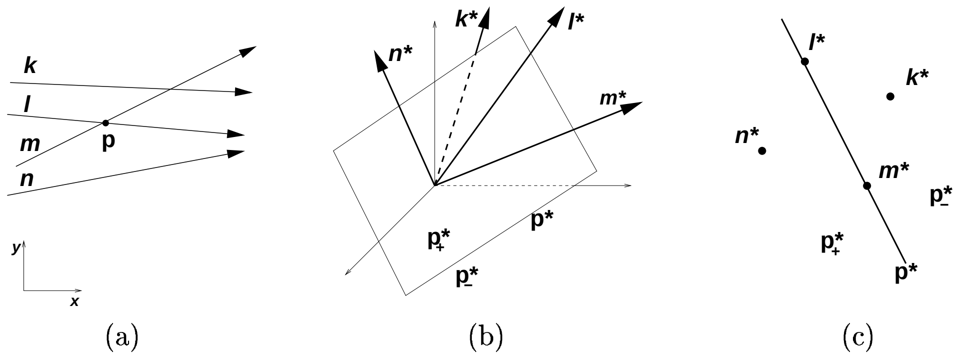


Fig. 2. (a) Four oriented lines in primal space. (b) Mappings of the four lines and point p. Lines intersecting p map to plane p^*. Lines passing clockwise (counterclockwise) around p, map to half-spaces p^*_- (p^*_+). (c) The situation after projection to a plane perpendicular to p^*.

4.2 Lines intersecting a point

A *pencil* of oriented lines intersecting a point $p = (p_x, p_y)$ maps to an oriented plane p^* in line space that is expressed as

$$p^* = \Big\{(x, y, z) \in E^3 \;\Big|\; p_x x + p_y y + z = 0\Big\}.$$

This plane subdivides line space in two half-spaces p^*_+ and p^*_-. Points in p^*_- correspond to oriented lines passing clockwise around p (see Figure 2). Points in p^*_+ correspond to oriented lines passing counterclockwise around p. We denote $-p^*$ an oriented plane opposite to p^* that can be expressed as

$$-p^* = \Big\{(x, y, z) \in E^3 \;\Big|\; -p_x x - p_y y - z = 0\Big\}.$$

4.3 Lines intersecting a line segment

Oriented lines intersecting a line segment can be decomposed into two sets depending on their orientation. Consider a configuration depicted in Figure 3.

The supporting line l_S of a line segment S partitions the primal space into half-spaces S^+ and S^-. Denote a and b the two endpoints of S and a^* and b^* their dual mappings. Lines that intersect S and are oriented from S^- to S^+ can be expressed as an intersection of half-spaces $a^*_+ \cap b^*_-$ in line space. The opposite oriented lines intersecting S can be expressed as $a^*_- \cap b^*_+$.

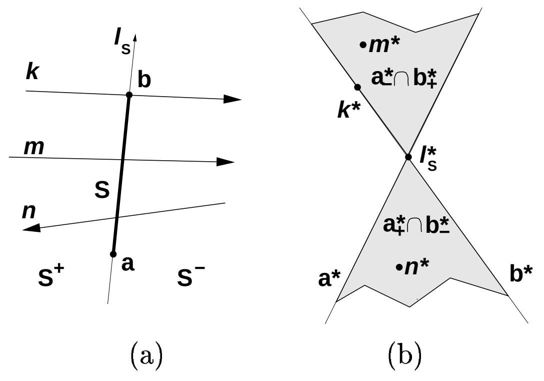


Fig. 3. (a) A line segment S and three oriented lines that intersect S. (b) The situation in line space: The projection of two wedges corresponding to lines intersecting S. The supporting line l_S of S maps to two half-lines that project to the point l^*_S. The line k intersects the point b and therefore its mapping lies in the plane b^*. Lines m and n map to the wedge corresponding to their orientation.

4.4 Lines intersecting two line segments

Consider two disjoint line segments such as those depicted in Figure 4-(a). The set of lines intersecting the two line segments in a given order can be described as an intersection of four half-spaces in line space. The intersection of these half-spaces is a pyramid with the apex at the origin of line space. The half-spaces are defined by mappings of the end-points of the two line segments. The boundary half-lines of the pyramid correspond to mappings of the four *extremal lines* induced by the two segments. Denote $P(S,O)$ a line space pyramid corresponding to lines intersecting line segments S and O in this order. We represent the pyramid by a *blocker polygon* $B(S,O)$ (see Figure 4-(b)). The blocker polygon $B(S,O)$ only represents the pyramid $P(S,O)$ and thus it need not be planar, i.e. its vertices may lie anywhere on the boundary half-lines of $P(S,O)$. We normalize the vertices of the blocker polygon to lie on the unit sphere centered at the origin of line space.

In Figure 5-(a), the supporting line of cd intersects ab at point x. The set of lines intersecting ab and cd consists of lines intersecting ax and cd, and lines intersecting xb and cd. Lines intersecting ax and cd map to a pyramid described by intersection of three half-spaces induced by mappings of a, x, and d. Lines intersecting xb and cd can be described similarly. The configuration in line space is depicted in Figure 5-(b).

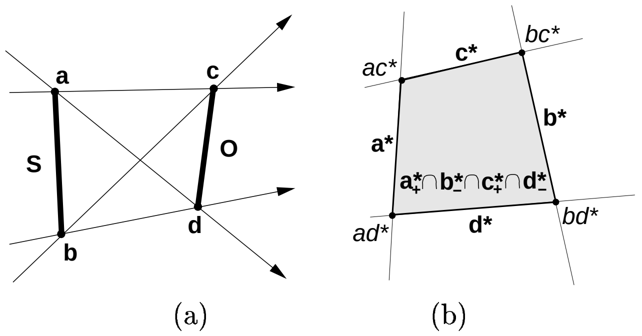


Fig. 4. (a) Two line segments and the corresponding four extremal lines oriented from S to O. The separating lines ad and bc bound the region of partial visibility of S behind O (penumbra). The supporting lines ac and bd bound region where S is invisible (umbra). (b) The blocker polygon $B(S, O)$ representing the pyramid $P(S, O)$.

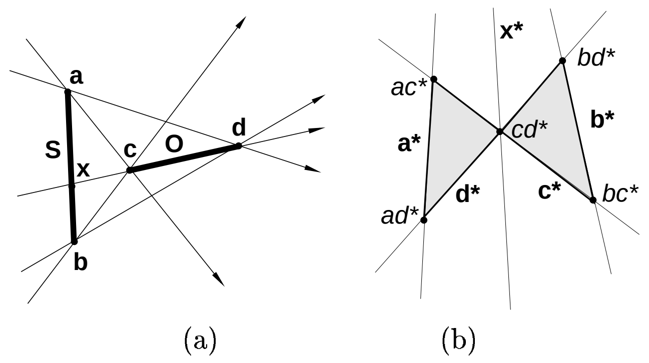


Fig. 5. (a) Degenerate configuration of line segments: The supporting line of cd intersects ab at point x. There are five extremal lines. Note, that there is no umbra region. (b) In line space the configuration yields two pyramids sharing a boundary that is a mapping of the oriented line cd.

4.5 Lines intersecting a set of line segments

Consider a set of $n + 1$ line segments. We call one line segment the *source* (denoted by S) and the other n segments we call occluders (denoted by O_k, $1 \leq k \leq n$). Further in the paper we will use the term *ray* as a representative of an oriented line that is oriented from the source towards the occluders.

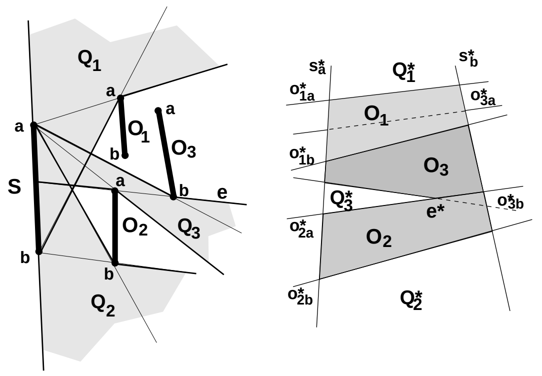
Assume that we can process all occluders in a strict front-to-back order with respect to the given source. We have already processed k occluders and we continue by processing O_{k+1}. O_{k+1} can be visible through rays that correspond to the pyramid $P(S, O_{k+1})$. However, some of these rays can be blocked by combination of already processed occluders O_x $(1 \leq x \leq k)$. To determine if O_{k+1} is visible we subtract all $P(S, O_x)$ from $P(S, O_{k+1})$:

$$\mathcal{V}(S, O_{k+1}) = P(S, O_{k+1}) \; - \bigcup_{1 \leq x \leq k} P(S, O_x)$$

$\mathcal{V}(S, O_{k+1})$ is a set of pyramids representing rays through which O_{k+1} is visible from S. Consequently, all rays corresponding to $\mathcal{V}(S, O_{k+1})$ are blocked behind O_{k+1}. If $\mathcal{V}(S, O_{k+1})$ is an empty set, occluder O_{k+1} is invisible. This suggests an incremental construction of an arrangement of pyramids $\mathcal{A}_k$ that corresponds to rays blocked by the k processed occluders. We determine $\mathcal{V}(S, O_{k+1})$ and $\mathcal{A}_{k+1}$ ($\mathcal{A}_0$ is empty):

$$\mathcal{V}(S, O_{k+1}) = P(S, O_{k+1}) \ - \ \mathcal{A}_k,$$
$$\mathcal{A}_{k+1} = \mathcal{A}_k \cup P(S, O_{k+1}) = \mathcal{A}_k \ \cup \ \mathcal{V}(S, O_{k+1}).$$

Figures 6-(a,b) depict a projection of an arrangement $\mathcal{A}_3$ of a source and three occluders. Note that the shorter the source line segment the narrower are the pyramids $P(S, O_k)$ (s_a^* and s_b^* get closer).



(a) (b)

Fig. 6. (a) The source line segment S and three occluders. Q_{1-3} denote unoccluded funnels. (b) The line space subdivision. For each cell, the corresponding visible occluder is depicted. Note the cells Q_1^*, Q_2^* and Q_3^* corresponding to unoccluded funnels.

Recall that the pyramid $P(S, O_k)$ is represented by the blocker polygon $B(S, O_k)$. The construction of the arrangement $\mathcal{A}_k$ resembles the from-point visibility problem, more specifically the hidden surface removal applied to the blocker polygons with respect to the origin of line space. However there is no notion of depth in our definition of line space. The priority of blocker polygons is either completely determined by the processing order of occluders or their depths must be compared in primal space.

5 Occlusion tree

The occlusion tree is a *binary space partitioning* tree (BSP tree) maintaining the arrangement $\mathcal{A}_k$. Each node N of the tree represents a subset of line space $\mathcal{Q}_N^*$. The root of the tree represents the whole line space. If N is an interior node, it is associated with a plane π_N. The left child of N represents $\mathcal{Q}^*_N \cap \pi^+_N$, the right child $\mathcal{Q}^*_N \cap \pi^-_N$, where π^+_N and π^-_N are half-spaces induced by π_N.

Leaves of the tree are classified *in* or *out*. If N is an *out*-leaf, $\mathcal{Q}^*_N$ represents unoccluded rays emerging from the source. If N is an *in*-leaf it is associated with an occluder O_N that blocks the corresponding set of rays $\mathcal{Q}^*_N$. Further N stores an intersection of the blocker polygon $B(S, O_N)$ and $\mathcal{Q}^*_N$, denoted B_N. $\mathcal{Q}^*_N$ represents a *funnel* $\mathcal{Q}_N$ in primal space that is bound by points corresponding to planes of the tree on the path from the root to N. N also contains a line segment I_N that is an intersection of the occluder O_N and the funnel $\mathcal{Q}_N$.

Consider a source and a single occluder such as in Figure 4-(a). The corresponding occlusion tree has four interior nodes that represent the endpoints of the two line segments. We call this tree the elementary occlusion tree for a blocker polygon $B(S, O)$, denoted e-OT($B(S, O)$) (see Figure 7-(a)). A more complex situation is depicted in Figure 7-(b).

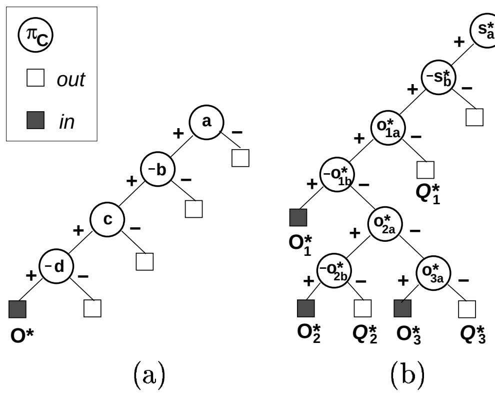


Fig. 7. (a) An elementary occlusion tree for a configuration shown in Figure 4-(a). (b) An occlusion tree for the three occluders depicted in Figure 6-(a). The three *in*-leaves correspond to rays blocked by the relevant occluders. The *out*-leaves Q_i represent three funnels of unoccluded rays emerging from S in the direction of occluders.

We need to perform polyhedra set operations on the arrangement $\mathcal{A}_k$ to obtain both the arrangement $\mathcal{A}_{k+1}$, and $\mathcal{V}(S, O_{k+1})$ that describes visibility of occluder O_{k+1}. In particular we must determine the set difference of $P(S, O_{k+1})$ and $\mathcal{A}_k$ to obtain $\mathcal{V}(S, O_{k+1})$ and the union of $\mathcal{A}_k$ and $\mathcal{V}(S, O_{k+1})$ to obtain $\mathcal{A}_{k+1}$. The set difference operation can be performed easily using the BSP tree by "filtering" $P(S, O_{k+1})$ down the tree [31,32]. The filtering identifies the desired set $\mathcal{V}(S, O_k + 1)$ that is then used to extend the tree so that it represents the arrangement $\mathcal{A}_{k+1}$.

Assume that we can determine a strict front-to-back order of occluders with respect to the source S. When we process occluder O_k all occluders O_j $(1 \leq j < k)$ that can block visibility of O_k have been already processed before. On the other hand O_k cannot block any occluder O_j $(1 \leq j < k)$.

The occlusion tree construction algorithm proceeds as follows: For an occluder we construct a pyramid $P(S, O_k)$ corresponding to rays blocked by this occluder. The pyramid is represented by a blocker polygon $B(S, O_k)$, that is then filtered down the tree. The algorithm maintains two variables: The current node N_c and the current blocker polygon B_c. Initially, N_c equals to the root of the tree and B_c equals to $B(S, O_k)$.

If N_c is not a leaf, we determine the position of B_c and the plane π_{N_c} associated with N_c. If B_c lies in the positive half-space induced by π_{N_c}, the algorithm continues in the left subtree. Similarly if B_c lies in the negative half-space induced by π_{N_c}, the algorithm continues in the right subtree. If B_c intersects both half-spaces, it is split by π_{N_c} into two parts B_c^+ and B_c^- and the algorithm proceeds in both subtrees of N_c with appropriate fragments of B_c.

If N_c is a leaf node then we make a decision depending on its classification. If N_c is an *in*-leaf, all rays represented by B_c are blocked by the occluder O_{N_c} referred in N_c. If N_c is an *out*-leaf, all rays represented B_c are unoccluded. Thus B_c is a part of $\mathcal{V}(S, O_k)$. We replace N_c by e-OT(B_c) representing the pyramid induced by B_c.

5.1.1 Handling approximate front-to-back order

Until now we have assumed that a front-to-back order of occluders with respect to the source is determined. In practice it can be advantageous to determine only an approximate front-to-back order. In the latter case we cannot guarantee the position of the currently processed occluder O_k with respect to the already processed occluders. When reaching an *in*-leaf of the occlusion tree we have to check the depth of the occluder O_k with respect to the occluder O_{N_c} associated with the leaf. We use the line segments I_k and I_{N_c} that are intersections of O_k and O_{N_c} with the funnel $\mathcal{Q}_{N_c}$ (see I_2 and I_1 on Figure 8-(a)). Denote an open half-space defined by I_{N_c} that contains the source H^+ and the opposite open half-space H^-. The two line segments can be in four mutual positions:

(1) I_k behind I_{N_c}: I_k lies completely in H^-.
(2) I_k in front of I_{N_c}: I_k lies completely in H^+.
(3) I_k intersects I_{N_c}: I_k lies in both H^+ and H^-.

(4) I_k on I_{N_c}: I_k does not intersect H^+ nor H^-.

In the first case all rays represented by B_c are blocked by I_{N_c}. Thus O_k is not visible through these rays and no modification to the tree is necessary.

In the second case all rays represented by B_c are not blocked by I_{N_c}. Thus O_k is visible and O_{N_c} is not visible through these rays. We construct e-OT(B_c) and filter the "old" blocker polygon B_{N_c} down this tree (see Figure 8). Finally, we replace the N_c by the constructed tree.

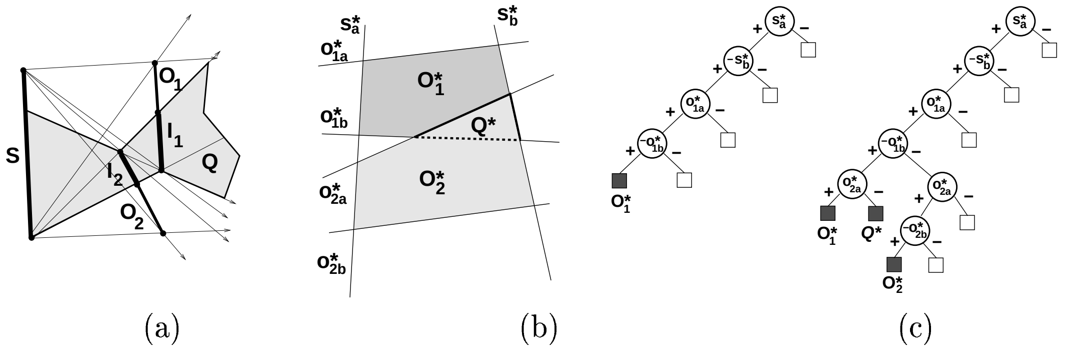


Fig. 8. (a) Occluder O_2 partially hides occluder O_1 that is already in the tree. The funnel $\mathcal{Q}$ corresponds to the lines through which O_2 hides O_1. Note that O_1 is still completely visible from some points on S. (b) Situation in line space. (c) The original tree and the tree after inserting O_2. Note, the lower left node has both descendants classified as *in*.

In the third case some lines of B_c are blocked by I_{N_c} and some block I_{N_c}. We determine the intersection point X of the line segments I_k and I_{N_c}. The point X maps to the plane X^* in line space. We split B_c by X^* obtaining B_c^+ and B_c^-. Depending on the position of the endpoints of I_k we decide which of the two fragments corresponds to lines blocked by I_{N_c}. This fragment can be deleted as in the case 1. The other fragment is treated as in the case 2.

The solution of the fourth case depends on the application. For the sake of simplicity we assume that O_k is invisible through rays represented by B_c.

5.2 Visibility from a region

The occlusion tree captures visibility with respect to a given line segment. Visibility from a convex polygonal region can be determined by computing visibility from its boundaries. A separate tree can be built to capture visibility from each boundary line segment S_i of source region R_S. Alternatively we can build a single occlusion tree that captures visibility from all boundaries. Inserting rays blocked by occluder O_k into the tree involves insertion of a blocker polygon $B(S_i, O_k)$ for each boundary line segment S_i of the source region.

6 Visibility tests

In this section we describe algorithms that use the occlusion tree to test visibility of a line segment or a region with respect to the already processed occluders.

6.1 Visibility of a line segment

To determine visibility of a line segment L we can use the algorithm from Section 5.1, while avoiding the modifications to the occlusion tree. We collect the set $\mathcal{V}(S, L)$ of visible fragments of the blocker polygon $B(S, L)$. If this set is empty, L is not visible from S. Otherwise, it is visible through the set of rays that correspond to $\mathcal{V}(S, L)$.

6.2 Visibility of a region

Visibility of a region can be determined by checking visibility of it boundaries. Visibility of each boundary is determined as described in the Section 6.1. In the next section we describe a faster conservative visibility tests that sacrifices accuracy for speed.

6.3 Conservative visibility of a region

The conservative visibility test aims to avoid testing each boundary of the region separately. Instead it determines visibility of the region using a single traversal of the occlusion tree.

The conservative visibility test proceeds as follows: Given a source region R_S and a polygonal region R_X we find supporting and separating lines of the source region R_S and R_X (see Figure 9-(a,b)). We construct a blocker polygon $B(R_S, R_X)$ using mappings of the supporting and separating lines as its vertices. This blocker polygon generally represents a superset of rays intersecting R_S and R_X (see Figure 10-(a,b)). $B(R_S, R_X)$ is filtered down the occlusion tree as described in Section 5.1. Reaching an *out*-leaf we can conclude that at least part of R_X is visible and terminate the algorithm. If the filtering procedure reaches an *in*-leaf L we classify visibility of the current part of R_X by testing the depth order of O_L and R_X.

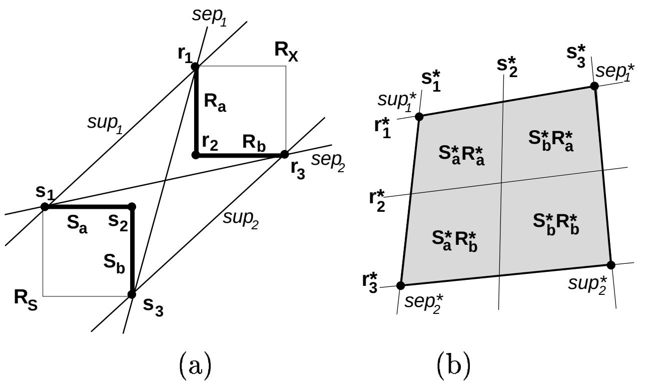


Fig. 9. (a) Two rectangular regions and the corresponding extremal lines. (b) In line space there are four blocker polygons corresponding to four combinations of the mutually visible boundaries of R_S and R_X. The vertices of the union of these blocker polygons correspond to supporting (sup$_1$, sup$_2$) and separating (sep$_1$, sep$_2$) lines of R_S and R_X.

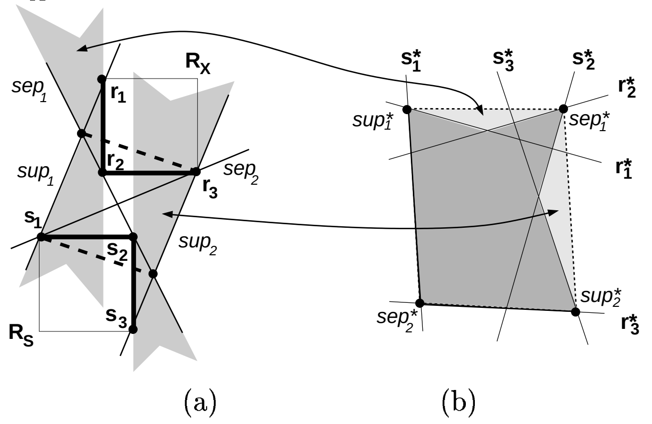


Fig. 10. (a) A configuration of R_S and R_X that leads to a conservative blocker polygon. The blocker polygon represents lines intersecting the “virtual” line segments show dashed. (b) The blocker polygon constructed from mappings of the supporting and separating lines represents a superset of the rays intersecting R_S and R_X.

6.4 Maximal visibility distance

We have assumed that the occlusion tree contains no information about the depth of the occluders and the depth tests are conducted at the leaf nodes using the occluders themselves. In this section we show how to include a conservative hierarchical depth representation of occluded rays. This representation can be used for quick visibility tests that eventually terminate near the root of the tree.

Each node N of the occlusion tree is associated with the *maximal visibility distance* (MVD) denoted d_N. If N is an *out*-leaf, $d_N = \infty$. If N is an *in*-leaf,

d_N is the maximal distance of the source region R_S and the part I_N of occluder O_N intersecting the funnel $\mathcal{Q}_N$. If N is an interior node of the tree, d_N is a maximum of MVDs of its children. MVDs are updated by propagating their changes up the tree after each insertion of an occluder. The MVD of node N can be used to quickly determine that an occluder O_x is invisible with respect to N if the minimal distance of O_x from R_S is greater than d_N.

The described depth representation is similar to the hierarchical z-buffer [16] used for the from-point visibility culling in 3D. The MVDs are conservative, hierarchical, and piecewise constant representation of the depth map with respect to the given region. In leaves of the tree the depth is represented exactly by the associated occluder fragments I_N.

7 Hierarchical visibility algorithm

The above mentioned methods are used within a hierarchical visibility algorithm. Occluders are organized in a kD-tree, in which a node N corresponds to rectangular region R_N. The root of the kD-tree corresponds to the bounding box of the whole scene. Leaves of the tree contain links to the occluders that intersect the corresponding cells. The kD-tree is used for two main purposes: Firstly, it allows to determine the approximate front-to-back order of occluders. Secondly, pruning of the kD-tree by hierarchical visibility tests leads to output-sensitive behavior of the algorithm.

The occluders are processed in an approximate front-to-back order with respect to the source region. We have used two ordering algorithms. The first processes the kD-tree using a strict front-to-back ordering of kD-tree nodes with respect to the center of the source region. The second uses a priority queue, where the priority of node N is given by the minimal distance of R_N from the source region. In both cases occluders stored within a leaf node are processed in random order. The occlusion tree is constructed incrementally by inserting blocker polygons $B(S_i, O_k)$ corresponding to the currently processed occluder O_k and the i-th boundary of the source region that faces O_k.

The occlusion tree construction is interleaved with visibility tests of region R_N corresponding to the currently processed kD-tree node N. If R_N intersects the source region it is classified visible. Otherwise the visibility test classifies visibility of R_N with respect to the already processed occluders. If R_N is invisible, the subtree of N and all occluders it contains are culled. If R_N is visible, we process the descendants of N recursively.

7.1 Achieving exact visibility classification

The hierarchical visibility tests provide a useful information about visibility of whole scene regions. Nevertheless the visibility classification of hierarchy nodes as described above is only conservative. Due to the approximate depth ordering of the occluders and the kD-tree nodes it is possible that there is an unprocessed region R_l containing occluders which might occlude (or at least partially occlude) the currently processed region R_k. R_k can be classified visible, although in fact it is invisible due to the influence of unprocessed occluders from R_l. For an exact classification of hierarchy nodes we have to perform additional series of visibility tests on all leaves of the kD-tree that were previously classified visible. This test eventually decides that the corresponding region is invisible if all relevant occluders are considered.

8 Results

We have evaluated the presented algorithms on three types of scenes: A city plan, a building interior, and random line segments. The scenes are depicted in Figures 1, 11-(a), and Figure 11-(b), respectively. On each figure the blue rectangle represents the source region. Yellow lines correspond to extremal lines that bound the visible funnels ($\mathcal{Q}_{L_x}$). Red line segments depict visible parts of occluders (I_{L_x}). Gray regions were culled by hierarchical visibility tests. Table 1 summarizes measurements for several source regions.

Figure 1 depicts a 2D cut through the scene representing the city of Glasgow [33]. For the selected source region the majority of the scene was culled by the hierarchical visibility tests and only few occluders corresponding to the neighboring streets were found visible. Figure 11-(a) depicts a ground plan of the Soda Hall of the University of Berkeley, Figure 12-(b) shows the corresponding blocker polygons.

Scenes with random line segments allowed us to study the dependence of the algorithm on the complexity of the scene. We could observe that the size of the occlusion tree is proportional to the total number of funnels $\mathcal{Q}_{N_x}$ through which occluders are visible from R_S. The behavior of the algorithm also depends on the properties of the kD-tree. The more occluders are referred in each leaf of the kD-tree the less precise front-to-back order is determined. Consequently, the occlusion tree is slightly larger due to late insertions of visible occluders. Additionally, more occluders are inserted in the tree, although they could be culled by the hierarchical visibility test. Figure 12-(a) depicts the dependence of the size of the occlusion tree on the number of inserted blocker polygons. All the curves were measured for a scene with 10000 random line

scene	occ.	kD-tree nodes	R_S size	tested kD-nodes	blocker polygons	OT nodes	visible occluders	time
	[−]	[−]	[%]	[%]	[−]	[−]	[%]	[s]
Glasgow	94460	13711	0.017	4.2	4400	1371	0.32	1.2
		13711	0.21	6.8	6919	3233	0.75	3.5
		13711	0.64	16	13142	7545	1.5	9.2
Soda	873	335	2.5	50	759	1045	19	0.31
		2169	2.5	34	619	671	19	0.63
		2169	6	65	1078	2605	44	2.2
Random	1000	61	5.1	62	1682	8639	54	1.3
		1767	5.1	51	1018	4385	54	1.3
Random	20000	7003	0.21	3.3	964	3185	2	0.61
		7003	3.4	8.2	2602	19179	13	5.2

Table 1
Summary of the results of the visibility algorithm. The table contains the total number of kD-tree nodes, the relative size of the source region with respect to the bounding box of the scene, the number of nodes of the kD-tree on which the hierarchical visibility test was applied, the total number of blocker polygons constructed for occluders, the total number of occlusion tree nodes, the percentage of occluders that are visible, and the total running time of the visibility algorithm. Measured on a PC with 950MHz Athlon CPU and 256MB RAM.

segments. The curves A and B were measured for the kD-tree with 50 occluders per leaf. The curves C and D correspond to the kD-tree with 5 occluders per leaf. For the curves A and C the front-to-back ordering with respect to the center of R_S was used, whereas for the curves B and D the priority queue front-to-back ordering was applied. Note that the “stairs” in graphs A and C occur due to the processing of the kD-tree using a depth-first search. It follows from the graphs that the best results were obtained using method D, i.e. the finer kD-tree and the priority queue front-to-back order. The corresponding curve also shows that once the occlusion tree represents enough blocked rays its size does not grow.

9 Conclusion and Future work

We have presented an algorithm for computing visibility from a given region in a 2D scene consisting of a set of line segments. The algorithm solves visibility by performing set operations in line space. The sets of occluded rays are represented using hierarchical partitioning of line space maintained by an occlusion tree. This technique provides an exact solution to regional visibility in 2D. The method requires an implementation of only a few simple geometrical algorithms. We have applied the method to computing a PVS in three different types of scenes including a scene representing a footprint of a large part of a city. The algorithm exhibited output-sensitive behavior for all tested scenes.

The presented method extends to 2.5D; it was already used as a core of a visibility preprocessing algorithm for urban scenes [4]. We currently investigate an approximate variant of the algorithm that exploits graphics hardware. The algorithm could also be extended to compute a measure of visibility in line space [34] corresponding to a form factor between the source region and the occluder.

Acknowledgments

Many thanks to Vlastimil Havran, Andreas Traxler, Tomáš Pajdla, and Václav Hlaváč for their helpful comments. This research was supported by the Czech Ministry of Education under Project LN00B096 and the Austrian Science Foundation (FWF) contract no. p-13876-INF.

References

[1] P. Wonka, M. Wimmer, D. Schmalstieg, Visibility preprocessing with occluder fusion for urban walkthroughs, in: Rendering Techniques (Proceedings of Eurographics Workshop on Rendering '00), 2000, pp. 71–82.

[2] V. Koltun, Y. Chrysanthou, D. Cohen-Or, Hardware-accelerated from-region visibility using a dual ray space, in: Rendering Techniques (Proceedings of Eurographics Workshop on Rendering '01), 2001, pp. 205–216.

[3] S. J. Teller, C. H. Séquin, Visibility preprocessing for interactive walkthroughs, in: Computer Graphics (SIGGRAPH '91 Proceedings), 1991, pp. 61–69.

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

[5] V. Koltun, Y. Chrysanthou, D. Cohen-Or, Virtual occluders: An efficient intermediate pvs representation, in: Rendering Techniques (Proceedings of Eurographics Workshop on Rendering '00), 2000, pp. 59–70.

[6] G. Schaufler, J. Dorsey, X. Decoret, F. X. Sillion, Conservative volumetric visibility with occluder fusion, in: Computer Graphics (SIGGRAPH '00 Proceedings), 2000, pp. 229–238.

[7] O. Sudarsky, C. Gotsman, Output-sensitive visibility algorithms for dynamic scenes with applications to virtual reality, Computer Graphics Forum 15 (3) (1996) C249–C258.

[8] F. Durand, 3D visibility: Analytical study and applications, Ph.D. thesis, Universite Joseph Fourier, Grenoble, France (Jul. 1999).

[9] J. Bittner, P. Wonka, Visibility in computer graphics, to appear in Journal of Environmental Planning (Feb. 2003).

[10] D. Cohen-Or, Y. Chrysanthou, C. Silva, F. Durand, A survey of visibility for walkthrough applications, To appear in IEEE Transactions on Visualization and Computer Graphics. .

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

[12] S. J. Teller, Computing the antipenumbra of an area light source, Computer Graphics (SIGGRAPH '92 Proceedings) 26 (2) (1992) 139–148.

[13] G. Drettakis, E. Fiume, A Fast Shadow Algorithm for Area Light Sources Using Backprojection, in: Computer Graphics (SIGGRAPH '94 Proceedings), 1994, pp. 223–230.

[14] A. J. Stewart, S. Ghali, Fast computation of shadow boundaries using spatial coherence and backprojections, in: Computer Graphics (SIGGRAPH '94 Proceedings), 1994, pp. 231–238.

[15] F. Durand, G. Drettakis, C. Puech, The visibility skeleton: A powerful and efficient multi-purpose global visibility tool, in: Computer Graphics (SIGGRAPH '97 Proceedings), 1997, pp. 89–100.

[16] N. Greene, M. Kass, G. Miller, Hierarchical Z-buffer visibility, in: Computer Graphics (SIGGRAPH '93 Proceedings), 1993, pp. 231–238.

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

[18] D. Luebke, C. Georges, Portals and mirrors: Simple, fast evaluation of potentially visible sets, in: Proceedings of Symposium on Interactive 3D Graphics '95, ACM SIGGRAPH, 1995, pp. 105–106.

[19] S. Coorg, S. Teller, Real-time occlusion culling for models with large occluders, in: Proceedings of the Symposium on Interactive 3D Graphics, ACM Press, 1997, pp. 83–90.

[20] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, H. Zhang, Accelerated occlusion culling using shadow frusta, in: Proceedings of ACM Symposium on Computational Geometry, 1997, pp. 1–10.

[21] J. Bittner, V. Havran, P. Slavík, Hierarchical visibility culling with occlusion trees, in: Proceedings of Computer Graphics International '98 (CGI'98), IEEE, 1998, pp. 207–219.

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

[23] D. Cohen-Or, G. Fibich, D. Halperin, E. Zadicario, Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes, in: Computer Graphics Forum (Eurographics '98 Proceedings), 1998, pp. 243–253.

[24] F. Durand, G. Drettakis, J. Thollot, C. Puech, Conservative visibility preprocessing using extended projections, in: Computer Graphics (SIGGRAPH '00 Proceedings), 2000, pp. 239–248.

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

[26] G. Vegter, The visibility diagram: a data structure for visibility problems and motion planning, in: In Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory (SWAT '90), Vol. 447 of Lecture Notes in Computer Science, Springer, 1990, pp. 97–110.

[27] M. Pocchiola, G. Vegter, The visibility complex, in: Proceedings of ACM Symposium on Computational Geometry, 1993, pp. 328–337.

[28] S. Rivière, Dynamic visibility in polygonal scenes with the visibility complex, in: Proceedings of ACM Symposium on Computational Geometry '97, ACM Press, New York, 1997, pp. 421–423.

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

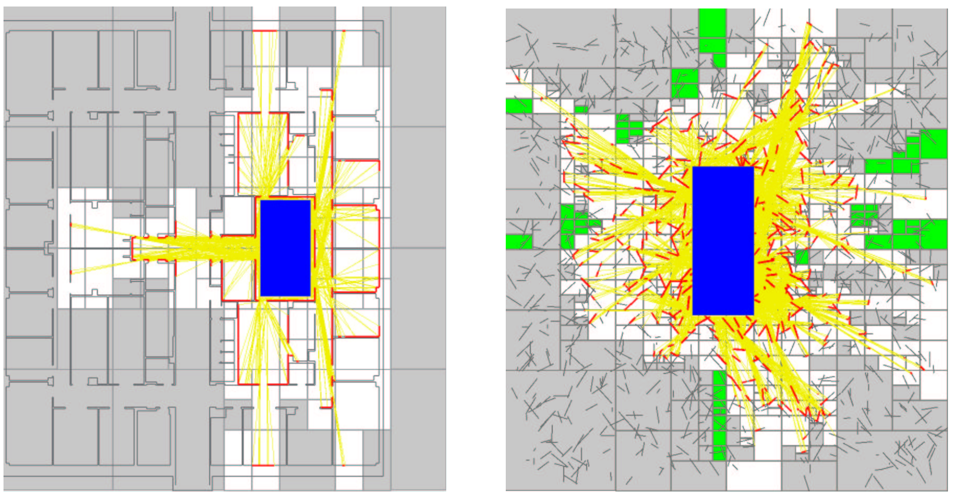
[30] J. Stolfi, Oriented Projective Geometry: A Framework for Geometric Computations, Academic Press, 1991.

[31] N. Chin, S. Feiner, Near real-time shadow generation using BSP trees, in: Computer Graphics (Proceedings of SIGGRAPH '89), 1989, pp. 99–106.

[32] B. Naylor, J. Amanatides, W. Thibault, Merging BSP trees yields polyhedral set operations, Computer Graphics (SIGGRAPH '90 Proceedings) 24 (4) (1990) 115–124.

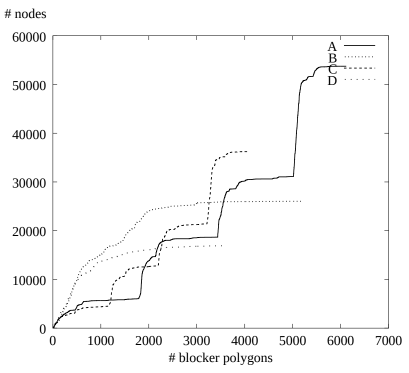
[33] Abacus Ltd., VR Glasgow project, `http://www.vrglasgow.co.uk/` (2003).

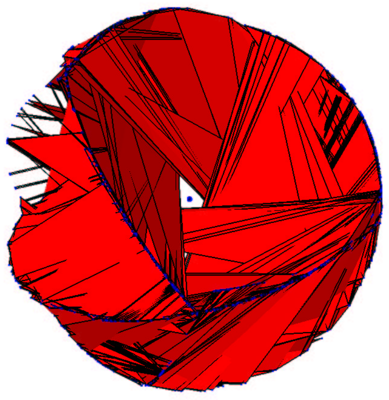
[34] R. Orti, S. Riviere, F. Durand, C. Puech, Using the Visibility Complex for Radiosity Computation, in: Lecture Notes in Computer Science (Applied Computational Geometry: Towards Geometric Engineering), Vol. 1148, Springer-Verlag, Berlin, Germany, 1996, pp. 177–190.



(a) (b)

Fig. 11. (a) The ground plan of the building of the Soda hall (873 occluders). 159 occluders are visible, the occlusion tree has 745 nodes. Computation took $190ms$. (b) A scene with 1000 random line segments. 540 segments are visible, the occlusion tree has 8607 nodes. The green regions were reclassified invisible by the second pass of visibility tests. The computation took $1.6s$.





(a) (b)

Fig. 12. (a) Dependency of the number of occlusion tree nodes on the number of processed blocker polygons for various settings of the algorithm (see Section 8). (b) Visualization of blocker polygons of processed occluders. Note the spiky triangles on the top-left. These triangles are blocker polygons of line segments supporting line of which intersects R_S. The chains of blue vertices lay in planes corresponding to mappings of vertices of R_S.