Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/070502

Visibility Resolution with Polar Diagrams

Lidia Ortega[*] and M. Dolores Robles-Ortega

Departamento de Informtica, Universidad de Jaén, Paraje Las Lagunillas A3-140, 23071 Jaén, Spain

Received: 16 Dec. 2012, Revised: 23 Apr. 2013, Accepted: 26 Apr. 2013
Published online: 1 Sep. 2013

Abstract: Visibility determination is an essential topic in computer graphics when visualizing large scenes. This problem can be addressed using many different techniques, but most of them discard obtaining an exact visible set because it is more time consuming compared with the solution provided by the graphics hardware. However the problem remains if the scenes must be visualized in a mobile device and the visible scene must be transmitted via the Internet. In this paper we introduce a new approach based on ray shooting for obtaining an exact visibility set in polygonal scenes in $\mathbb{R}^2$ and prism-shaped objects in $\mathbb{R}^3$. In both cases the scene is divided into disjoint regions using the *polar diagram*, a plane partition based on angle preprocessing. The polar diagram allows to improve computation times while ensuring accurate results in these scenes.

Keywords: Computer Graphics, Visibility Map, Ray-shooting technique

1 Introduction and state of the art

Two points in $2D$ $p=(x_p,y_p)$ and $q=(x_q,y_q)$ located in a scene E are visible if the line segment pq does not intersect with any other object in E. This concept can be extended to scenes defined in R^n. The visibility computation for a pair of points is indispensable to solve more complex visibility problems in several application areas such as computer graphics, robotics, simulation of wireless communications, etc.

To address the problem of visibility from the point of viewpoint of computational geometry, the scene E becomes an abstraction of reality. For example, a polygon with holes in R^2 may represent a furnished room. An urban environment may be represented by means of the footprints of their buildings, etc. In R^3 this abstraction process is lower, but the algorithms to solve visibility are more complex or require more computation time.

Visibility from a viewpoint p that observes the scene E in a specific viewing angle is also referred as the *visibility map* from point p. This is a key problem in path planning and walk-through applications. From the point of view of a mobile object or robot, visibility determines the free configuration space to move without collision. In computer graphics, visibility determination is important to reduce the amount of geometry that the hardware graphics must process.

In this paper we study an extension of the simplest visibility test between two points pq: the shooting query of the ray $r(t)=p+t\overrightarrow{d}$, $\overrightarrow{d}=\overrightarrow{q}-\overrightarrow{p}$, $t\subset[0,\infty]$, considering q and p as vectors with the coordinates of these points, [22]. Once solved this visibility problem, a selected set of rays starting from p can determine the visibility in a viewing angle. This ray shooting problem can be considered as a special instance of the geometric range-searching problem. In a preprocessing phase, a data structure divides the scene E into small regions to avoid performing the expensive ray-object intersection tests. Afterwards a ray-traversal algorithm searches the set of objects hit by this set of rays.

There are many spatial data structures supporting ray shooting algorithms. A tree-based data structure maintaining a partition of the scene may support efficient range searches, especially on balanced structures. The root represents the whole scene and each hierarchical level represents a decomposition of the level just above. The ray shooting test begins with the location of p into the tree data structure. The ray may intersect with the object or objects associated with this node. Otherwise the ray crosses successively to adjacent regions according to the ray direction $\overrightarrow{d}$, while the intersection is not found or the ray does not leave the scene. The concept of adjacency is implemented in the tree structure itself by accessing to positions of neighboring nodes. This decomposition is compatible with scenes in $\mathbb{R}^2$ or $\mathbb{R}^3$, and with closed or

[*] Corresponding author e-mail: lidia@ujaen.es

© 2013 NSP
Natural Sciences Publishing Cor.

open areas. The most used data structures in ray shooting are BSP-trees, [2,12], Kd-trees [15], octrees [11,24].

Flat space-oriented partitioning such as uniform grids are also widely used for ray traversal problems [9]. The advantages of this type of plane or space partition are the simplicity and efficiently of their implementation. However regular regions are not adapted to the distribution or size of the objects in the scene. The data structures described so far show their strength in some cases but not in all of them. Thus, they can be combined to work together in hybrid data structures [16].

Once a spatial subdivision is constructed using the most appropriate criterion, the ray shooting process consists in the traverse of adjacent regions. The data structure must help to make this process efficiently by means of its own configuration (uniform grids) or by defining topological relations such as the TIN (Triangulated Irregular Network).

Nevertheless, ray shooting in computer graphics is specially used in related problems to visibility, such as global illumination. These techniques named ray-casting and ray-tracing are image based algorithms of heuristic nature. In computer graphics, visibility determination is oriented to speed up the rendering phase by discarding as much non-visible objects as possible, in order to release the hardware of processing scene primitives that are invisible from the viewer position. Exact visibility is usually discarded for being more expensive in time than the solution given by the Z-buffer [8]. In fact, the most desired characteristic is to identify as much occluded objects as possible without forgetting any visible primitive. Thus, these techniques called *visibility culling*, search a potentially visible set (PSV) with the property of being conservative (considering all visible objects) even if it contains non-visible primitives. The Z-buffer is responsible of hidden those objects of the PSV that are finally non-visible.

The visibility culling techniques discard in a straightforward way those objects outside the viewing frustum (*view-frustum culling)* and also those primitives that face away from the observer *(back-face culling*). Finally, the *occlusion culling* techniques go beyond by rejecting primitives occluded behind of some other visible objects. These methods have been widely studied, we find some of them summarized in [1,6,14].

Occlusion culling techniques can be classified attending to different strategies for addressing the problem as well as for different types of scenes. It can be distinguished between occlusion culling *from a viewpoint* or *from a region*, this last studying all those visible objects from at least a point-position of a certain area. Point-based methods can be classified in *object* and *image-precision*, the first ones use the scene geometry, while image-precision techniques operate during the rasterization process. The order in which the PVS is obtained differentiates the techniques that remove candidates from the global set of primitives from those which start from an empty set that increases its size in the process. This method becomes more advisable if the percent of the visible scene is reduced, for example in densely occluded environments.

In Section 2 we highlighted the benefits of using an exact visibility culling method, above all in web-based systems. Section 3 introduces the polar diagram as a tessellation that divides the plane into regions with similar angular characteristics. In Section 4 several visibility problems related to ray shooting are defined and solved using the polar diagram as preprocessing. These algorithms run efficiently in dense scenes such as urban environments. Section 5 shows performance results for applying these algorithms with prism-shaped scenes representing city models in web-based systems. Finally in Section 6 we summarize the contributions of this paper and discuss possible improvements and applications.

2 Motivation

Despite the many methods in the literature to solve visibility, none of them is especially focused on web-based systems. In recent times web-based systems are extensively used due to the services offered through Internet. The demand grows specially for web 3D services such as mobile navigation in urban scenes, [7]. Mobile devices do not have the graphics capabilities of desktops computers, requiring a huge reduction of geometry to allow an accurate client-server interaction. The network must transfer at each time the information that the client device visualizes. This is an example in which exact visibility is important for accurate results since non-visible buildings transferred through the network is ineffective to maintain real-time interaction in the web system.

This paper introduces a ray shooting method for exact visibility determination from a viewpoint in polygonal scenes in $\mathbb{R}^2$, and prism-shape objects in $\mathbb{R}^3$. In both cases a plane tessellation called *polar diagram* [13], is used as preprocessing of the scene geometry. This plane partition makes the process of calculating ray intersections more efficient by reducing the set of possible intersecting objects. The resulting visible set is conservative because all the visible objects are finally found. The best performance case is given in densely occluded scenes such as city models where the visible set is expected to be reduced.

3 The polar diagram

The polar diagram [13], as any other plane tessellation, constructs a partition of the scene that makes some processes work faster by reducing the number of primitives to be tested in a specific instant. For instance, the Voronoi diagram, [19], finds the nearest site to a given point position in logarithmic time. The polar diagram can

© 2013 NSP
Natural Sciences Publishing Cor.

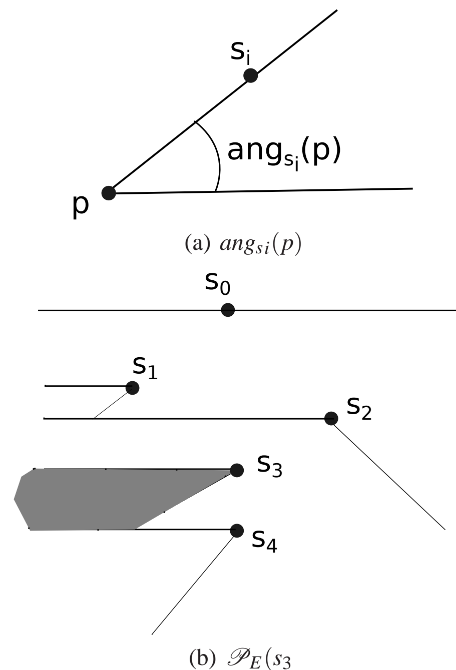
be defined in terms of the Voronoi diagram but replacing the minimum Euclidean distance as rule of construction with the *minimum angular distance*, that is, the minimum *polar angle* of any point p with respect to a specific object o_i, denoted as $ang_{o_i}(p)$. This angle is defined by the positive horizontal line of p (the starting angle) and the straight line linking p and o_i in the range $[0,\pi]$. The locus of points with smallest positive polar angle with respect to the object $o_i \in E$ is the *polar region* of o_i, denoted as $\mathscr{P}_E(o_i)$. Thus, $\mathscr{P}_E(o_i) = \{(x,y) \in \mathbb{R}^2 \mid ang_{o_i}(x,y) < ang_{o_j}(x,y), \forall j \neq i\}$. Figure 1.a) shows the polar angle of p with respect to the site s_i denoted as $ang_{s_i}(p)$. The shaded area of Figure 1.b) represents the polar region of the site s_3. Each point located in this region is angularly closer to object s_3 than to any other site.

Given a set E of n two-dimensional objects, $E = \{o_1, o_2, ..., o_n\}$, the *polar diagram* of E, denoted as $\mathscr{P}(E)$, is a half-plane division in polar regions. Because the polar angle is defined in the range $[0,\pi]$, it partitions the lower half-plane defined by the horizontal line containing the vertex (or site) with greater y-coordinate. Each generator object o_i creates a polar region $\mathscr{P}_E(o_i)$ representing the locus of points with common angular characteristics in a given angular criterion. Any point in this half-plane belongs to one polar region which determines its angular situation with respect to the rest of generator objects in the scene. More specifically, if point p lies in the polar region of object o_i, $p \in \mathscr{P}_E(o_i)$, we know that o_i is the first object found after performing an angular scanning in counter-clockwise starting at zero angle. Figure 1.b) is the polar diagram of a set of sites in the plane, and the horizontal line containing s_0 the frontier tessellation.

The polar diagram can be constructed in $\Theta(n \log n)$ for a set on n sites in the plane or a set of N polygonal objects with n vertices, using the Divide and Conquer or the Incremental methods, [13]. The strength of using this tessellation as preprocessing is avoiding any angular sweep by locating a point into a polar region in logarithmic time.

The polar diagram can be used as preprocessing to solve efficiently some angle-based problems as the convex hull of points and objects, [13], the path planning for robotics applications, [21], as well as Collision Detection, [20].

The polar diagram can be constructed for a set of static and dynamic objects in the plane as in [3]. Polygonal objects may represent an abstraction of any real object. In generalized cases, complex-geometry objects are replaced with some others formed by simple geometric items, whose complexity is increased hierarchically only when necessary. In all these cases a 2D representation of the environment, a city map for instance, can be used for some precalculations, as locating an observer in the scene or finding the visibility map from its position. Polar diagrams carry out a multi-purpose plane partition that can be useful as a



(a) $ang_{si}(p)$

(b) $\mathscr{P}_E(s_3$

Fig. 1: Polar diagram

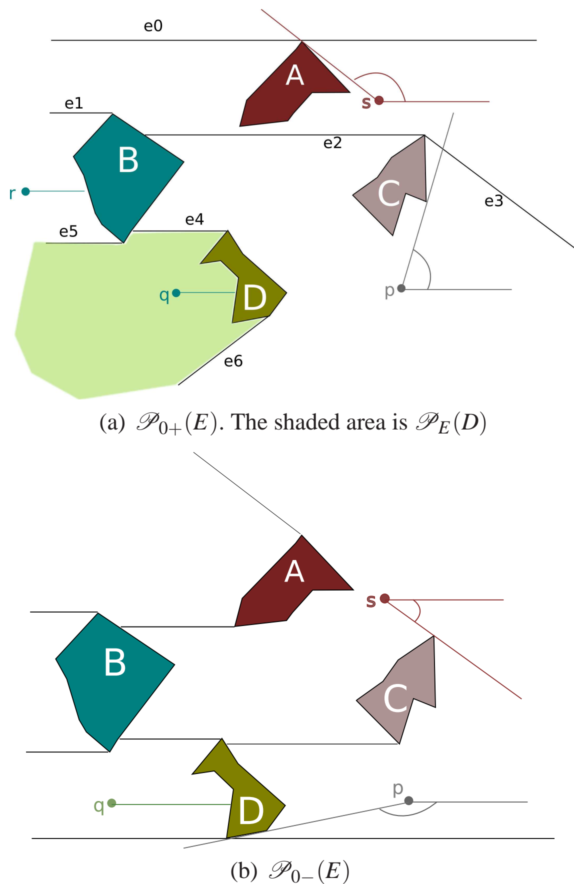
visibility culling technique, with important improvements in the final visualization time.

The definition of polar diagram, $\mathscr{P}(E)$, considers the counter-clockwise direction and the angle zero as criterion of construction. In certain applications, it is possible that the counter-clockwise criterion should be replaced by the clockwise or by another starting angle. Observe Figure 2.a) a counter-clockwise scanning from point s finds object A. If the rotation is clockwise, then the object found is C, as depicted in Figure 2.b).

Therefore, the criterion of construction can vary considering different starting angles or rotation directions. The previous definition of polar diagram, $\mathscr{P}(E)$ can be denoted as $\mathscr{P}_{0+}(E)$, considering the angle zero as starting angle, and the scanning in positive direction (counter-clockwise). The polar diagram which rotates clockwise or negative direction is denoted as $\mathscr{P}_{0-}(E)$, as the example of Figure 2.b). In the general case, the polar diagram can be computed considering the starting angle defined by the vector $\overrightarrow{d} = \overrightarrow{q} - \overrightarrow{p}$, considering $\overrightarrow{p}$ the vector with coordinates equal to the point p, the origin of the angular scanning. Thus, in general $\mathscr{P}_{d+}(E)$ is considered the polar diagram performed using the angle of $\overrightarrow{d}$ (the angle formed with the horizontal) and rotating in counter-clockwise. Similarly $\mathscr{P}_{d-}(E)$ denotes the polar diagram constructed using the same angle but scanning in clockwise.

The polar diagram data structure is key for obtaining efficient results in visibility. It is defined in terms similar to the Voronoi diagram, a winged-edge data structure

© 2013 NSP
Natural Sciences Publishing Cor.



(a) $\mathscr{P}_{0+}(E)$. The shaded area is $\mathscr{P}_E(D)$

(b) $\mathscr{P}_{0-}(E)$

Fig. 2: Polar diagram of polygonal objects using two different criteria; p lies in the region of C in the $\mathscr{P}_{0+}(E)$ polar diagram and in the polar region of D in $\mathscr{P}_{0-}(E)$

preserving topological information. This data representation is also used in other plane tessellation such us the Triangulated Irregular Network (TIN), used for terrain representation. Neighborly relations are important for grouping adjacent regions and therefore to speed up the process of traversal polar regions during the ray shooting. The Voronoi diagram uses these neighborly relations to make efficient the path planning problem resolution in [10] or to compute the drainage network in a triangulated terrain as in [17].

The boundary of a polar region $\mathscr{P}_E(o_i)$ is defined by a set of edges denoted as $PE_E(o_i)$. If o_i is a polygonal object, then this boundary is defined by polar edges, and also by edges of polygons, as depicted in Figure 2. Each polar edge $e_k \in PE_E(o_i)$, $e_k \in PE_E(o_j)$ divides two adjacent polar regions $\mathscr{P}_E(o_i)$ and $\mathscr{P}_E(o_j)$. For instance, e_6 divides $\mathscr{P}_E(D)$ and $\mathscr{P}_E(C)$. Thus, if the trajectory of a mobile object touches e_k after crossing the polar region of o_i, then it arrives to the adjacent polar region of o_j after crossing edge e_k, and avoiding linear searches.

Table 1: Polar diagram data structure of the example of Figure 2.a)

Object	Edges	Adjacent Object
A	e0, e1, e2, e3	A, B
B	e1, e5	B
C	e2, e3, e4, e6	C, B, D
D	e4, e5, e6	D, B

Edges	Adjacent regions
e0	⊘, A
e1	A, B
e2	A, C
e3	A, C
e4	C, D
e5	B, D
e6	D, C

Table ?? shows the topological relations of the polar diagram of Figure 2.a). For instance, the first row of the upper table states that $\mathscr{P}_E(A)$ is bounded by the polar edges e_0, e_1, e_2, e_3. In the second table, edge e_0 divides two adjacent polar regions, $\mathscr{P}_E(A)$ and the upper half-plane.

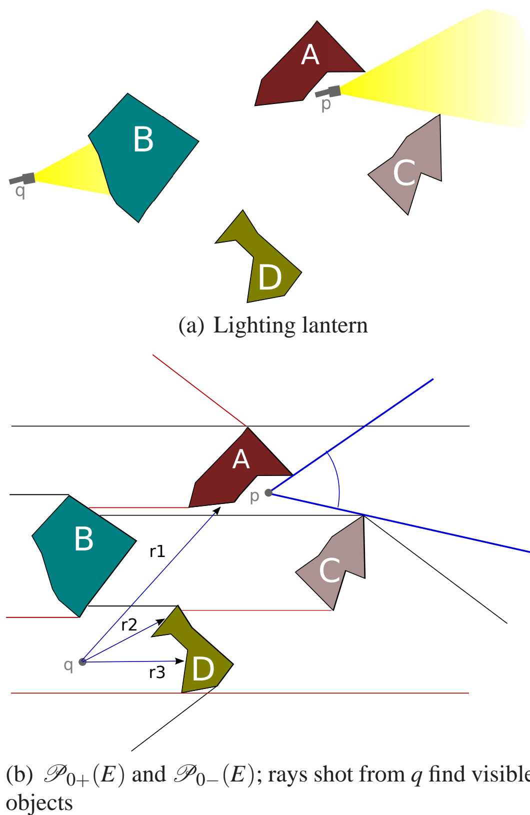
4 Visibility using the polar diagram as preprocessing

Polar diagrams can solve visibility problems due to its capability of determining the nearest angular neighbors. As stated before, this plane tessellation avoids angular searches by locating a point p in a polar region. This process requires $O(logn)$ instead of linear time.

The foundations for solving visibility problems using polar diagrams consist of defining p as the viewpoint, and considering the virtual scene composed by a set of polygons in $\mathbb{R}^2$. For 3D scenes represented by prism-shaped objects with polygonal base, the visibility problem can be defined as an extension of the 2D case. These objects resulting of the extrusion of a polygon are also called as 2.5D objects in computer graphics literature. They are usually used for representing buildings in virtual urban environments [4] or as bounding boxes that enclose other geometrically complex objects as in [1].

In this section we define several related visibility problems. The first determines the maximum visibility angle (if any) from a viewpoint p in a given direction. An extension of this problem is to find the first visible object from p in an arbitrary direction $\overrightarrow{d}$, also defined as ray shooting query. The solution to this simple problem is the basis for obtaining the visibility map from p in $\mathbb{R}^2$ or $\mathbb{R}^3$. The two dimensional case considers a scene of polygons, and the three dimensional case considers prism-shaped objects.

© 2013 NSP
Natural Sciences Publishing Cor.



(a) Lighting lantern

(b) $\mathscr{P}_{0+}(E)$ and $\mathscr{P}_{0-}(E)$; rays shot from q find visible objects

Fig. 3: Maximum visibility angle

4.1 Maximum visibility angle in a given direction

The maximum visible angle from point p in a given direction $\vec{d}$ is considered the maximum obstacle-free angle from the viewer position p when the scene E is being observed in direction $\vec{d}$. This is a problem related to illumination in computer graphics. The incident light beam between the gap formed by objects A and C is the maximum angle of visibility from the lantern position p, as depicted in Figure 3.a). However the opening angle from point q is null because the ray $r(t) = q + t\vec{d}$, $t \subset [0, \infty)$ intersects with object B, as seen in the picture. In this case there is not an open angle, but it is determined the specific illuminated object.

The maximum visibility angle from point p or the problem of determining the intersecting object of the ray $r(t)$ in a scene with n objects, are linear time problems using brute force algorithms. However the problem can be studied using the polar diagram as preprocessing.

Let us consider the scene $E = \{o_1, o_2, \ldots, o_n\}$ with n polygonal objects, the viewpoint p and the direction of visibility $\vec{d}$. If the polar diagrams $\mathscr{P}_{d+}(E)$ and $\mathscr{P}_{d-}(E)$ are pre-computed, then the maximum visibility angle is determined according to Lemma 1.

Lemma 1.*Let be $\mathscr{P}_{(E)d+}(o_i)$ the polar region in which the viewpoint p is located in the polar diagram $\mathscr{P}_{d+}(E)$ of the scene E, $p \in \mathscr{P}_{(E)d+}(o_i)$, and let be $\mathscr{P}_{(E)d-}(o_j)$ the polar region in which p is located in the polar diagram $\mathscr{P}_{d-}(E)$, $p \in \mathscr{P}_{(E)d-}(o_j)$. There exists an open visibility angle from p in direction $\vec{d}$ iff $o_i \neq o_j$. Otherwise, if $o_i = o_j$, then the visibility angle is null and the ray $r = p + t\vec{d}$, $t \subset [0, \infty)$, intersects with o_i.*

*Proof.*Let us consider $\vec{d}$ the direction of an horizontal vector with angle 0 (East direction). An open visibility angle exists if the ray $r(t) = p + t\vec{d}$, $t \subset [0, \infty)$, does not intersect with any object. Then if $p \in \mathscr{P}_{(E)0+}(o_i)$, it is necessary a non-zero angular sweep in counter-clockwise starting from direction $\vec{d}$ to find o_i. Likewise, an angular sweep in clockwise direction is required to find o_j. But object o_j could never be found sweeping counter-clockwise by definition of polar angle because it only sweeps in the angular range $[0, \pi)$, and o_j could never be reached, consequently $o_i \neq o_j$.

The implication in the reverse is straightforward considering that if two different objects are found using opposite angular scans from the angle 0 direction, then by definition there exists an open visibility angle.

Finally, if the ray $r(t)$ intersects with o_i, then o_i is the first object found when looking at $\vec{d}$ direction (East in this case). No angular scans in clockwise or counter-clockwise direction are necessary because o_i is found before starting any angular scans. Therefore this object must be the same in both cases.

In the example of Figure 3.b) the polar edges of $\mathscr{P}_{0+}(E)$ in black color and the edges of $\mathscr{P}_{0-}(E)$ in red are superimposed. Point p lies in the polar regions of objects C and D in $\mathscr{P}_{0+}(E)$ and $\mathscr{P}_{0-}(E)$ respectively. The maximum angle of visibility is obtained by throwing tangent lines to C and D as the figure 3.b) shows.

In the same example, point q lies in both cases in the polar region of the same object D, consequently object D is the first object that intersects the horizontal positive ray $r(t) = q + t\vec{d}$ according to Lemma 1. However, computing linear time processes to find intersections can be avoided by locating point p into a polar region in logarithmic time.

Theorem 1.*The maximum angle of visibility from point p in a given direction $\vec{d}$ in a scene with n polygonal objects can be calculated in $O(\log n)$ time using a pair of polar diagrams as preprocessing.*

*Proof.*Lemma 1 guarantees that it is possible to find the maximum visibility or the first visible object in direction $\vec{d}$ using the pairs of polar diagrams $\mathscr{P}_{d+}(E)$ and $\mathscr{P}_{d-}(E)$ by locating the viewpoint p into both polar diagrams. Each of these location processes requires a logarithmic time. The rest of operations such as throwing tangent lines need only constant time.

© 2013 NSP
Natural Sciences Publishing Cor.

4.2 Visible object in an arbitrary direction

Lemma 1 provides a method to find the first visible object in a given direction $\overrightarrow{d}$ by means of pre-computing a pair of polar diagrams $\mathscr{P}_{d+}(E)$ and $\mathscr{P}_{d-}(E)$, using $\overrightarrow{d}$ as criterion of construction.

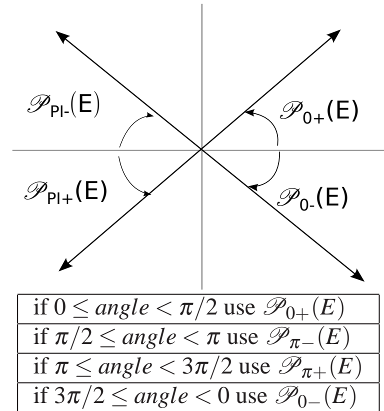
This method is efficient for calculating visibility (or related problems such as intersections) from any site p in the plane in $\overrightarrow{d}$ direction. However in real problems, the ray $r(t) = p + t\overrightarrow{d}$ may represent the movement direction in a specific instant, and this trajectory may vary along the execution. Suppose a robot moving in a scene, or an avatar navigating in a virtual environment. In these cases $\overrightarrow{d}$ can take any value, and the construction of an arbitrary number of polar diagrams in each direction requiring $O(n\log n)$ time, is neither efficient nor feasible. Nevertheless, it is possible to provide an efficient solution to this visibility problem by using the topological relations implemented in the data structure of the polar diagram, as well as its angular characteristics.

According to the definition of polar diagram, the polar region of the site o_i, $\mathscr{P}_{(E)0+}(o_i)$, is the locus of points with smaller positive angle with respect to o_i in the angular range $[0,\pi)$. This means that if point $p \in \mathscr{P}_{(E)0+}(o_i)$, the ray $r(t) = p + t\overrightarrow{d}$, with $\overrightarrow{d}$ defining any angle in the range $[0,\pi)$ is angularly close to object o_i, even if r does not intersect with o_i. This statement can be emphasized if we search only in the range $[0,\pi/2]$, which implies that if $\overrightarrow{d}$ is framed in the first quadrant, we search angularly close objects using the $\mathscr{P}(E) = \mathscr{P}_{0+}(E)$ polar diagram.

See again Figure 3.b), the rays r_2 and r_3, collides with object D, however ray r_1 does not. In any case r_1 is angularly close to o_i and this proximity may be used for searching angularly close objects. If ray r_1 does not intersect with D, the search can be focused on angularly close objects, which can be found efficiently using the this topological data structure. This characteristic benefits polar diagrams with respect to similar plane tessellation such as Voronoi diagrams. The approach is similar, a ray crosses adjacent regions to find the first intersecting object with the ray describing the trajectory, however the minimum Euclidean distance criterion is less efficient than using angular preprocessing.

If $\mathscr{P}_{0+}(E)$ is able solve angular proximity in the range $[0,\pi/2]$, a combination of polar diagrams constructed with different criteria can scan the angular spectrum $[0,2\pi)$. In fact, we describe $\mathscr{P}_{0-}(E)$ as the polar diagram constructed using the starting angle 0 and sweeping the plane in clockwise direction. The remaining $[\pi/2, 3\pi/2]$ range can be angularly covered with the $\mathscr{P}_{\pi+}(E)$ and $\mathscr{P}_{\pi-}(E)$, sweeping the third and second quadrant respectively.

These four orthogonal polar diagrams used as preprocessing can solve visibility in an arbitrary $\overrightarrow{d}$ direction. The visibility problem from point p in the direction $\overrightarrow{d}$ or ray shooting query, is defined as the process to find the intersecting object with the ray $r(t) = p + t\overrightarrow{d}$. The direction $\overrightarrow{d}$ is framed in one quadrant and one polar diagram is selected according to the scheme of Figure 4.



if $0 \le angle < \pi/2$ use $\mathscr{P}_{0+}(E)$
if $\pi/2 \le angle < \pi$ use $\mathscr{P}_{\pi-}(E)$
if $\pi \le angle < 3\pi/2$ use $\mathscr{P}_{\pi+}(E)$
if $3\pi/2 \le angle < 0$ use $\mathscr{P}_{0-}(E)$

Fig. 4: The polar diagram is chosen according to the $\overrightarrow{d}$ direction

Algorithm 1 RayShootingQuery($\mathscr{PD}(E)$, $r(t) = p + t\overrightarrow{d}$):$l_i, o_i$

Input: The four polar diagrams E: $\mathscr{PD}(E) = \{\mathscr{P}_{0-}(E), \mathscr{P}_{0+}(E), \mathscr{P}_{\pi+}(E), \mathscr{P}_{\pi-}(E)\}$
- The ray $r(t) = p + t\overrightarrow{d}$ defining the visibility direction.
Output: The visible object $o_i \in E$ or $\emptyset$
- The list of crossed rays l_i
BEGIN
1. Select the polar diagram $\mathscr{P}(E) \in \mathscr{PD}(E)$ according to the table of Figure 4
2. Locate p in the polar region of object o, $p \in \mathscr{P}_{(E)}(o)$
3. RayCollision ($\mathscr{P}(E)$, $r(t)$, o): $\{o_i, l_i\}$
4. **RETURN** o_i and l_i
END

Algorithm 1 describes the process to find the object colliding with $r(t)$ using the polar diagrams as pre-processing. The input data are the set of polar diagrams $\mathscr{PD}(E) = \{\mathscr{P}_{0-}(E), \mathscr{P}_{0+}(E), \mathscr{P}_{\pi+}(E), \mathscr{P}_{\pi-}(E)\}$ and $r(t)$, the output are the colliding object o_i and the list of crossed polar edges l_i. This algorithm determines the polar diagram to be used and the polar region in which p is located $p \in \mathscr{P}_{(E)}(o)$, then it calls to the Algorithm 2 (RayCollision) that implements the traversal process and finds the colliding object o_i.

Now we describe the RayColliison process considering the example of Figure 5. Firstly, point p is located in the polar region of object o_{14}. The polar region

© 2013 NSP
Natural Sciences Publishing Cor.

Algorithm 2 RayCollision($\mathscr{P}(E)$, $r(t) = p + t\vec{d}$, o):$\{o_i, l_i\}$

Input: - The polar diagram $\mathscr{P}(E) \in \mathscr{PD}(E)$
- The ray $r(t) = p + t\vec{d}$ defining the visibility direction.
- The object o such that $p \in \mathscr{P}_{(E)}(o)$

Output: The tuple (o_i, l_i): the visible object $o_i \in E$ or $\emptyset$,
- The list of crossed edges l_i

BEGIN

1. **BOOL** terminate $\leftarrow$ **false**
2. Initialize $l_i \leftarrow \emptyset$ and $o_i \leftarrow o$
3. **WHILE** (**NOT** terminate) **DO**

(a) Determine the edge e intersecting with $r(t)$ in $\mathscr{P}_{(E)}(o_i)$
(b) IF $e = 0$ *(out of the scene)* OR e is a polygon edge *(collision detected)*
(c) THEN terminate $\leftarrow$**true**
(d) **ELSE** *(go to an adjacent region)*
i. Insert edge e in l, $l \leftarrow l + \{e\}$
ii. Let be o_j the object whose polar region is adjacent to edge e
iii. $o_i \leftarrow o_j$
(e) END_WHILE

4. **IF** $e = 0$ **THEN** $o_i = 0$
5. **RETURN** o_i and l_i

END

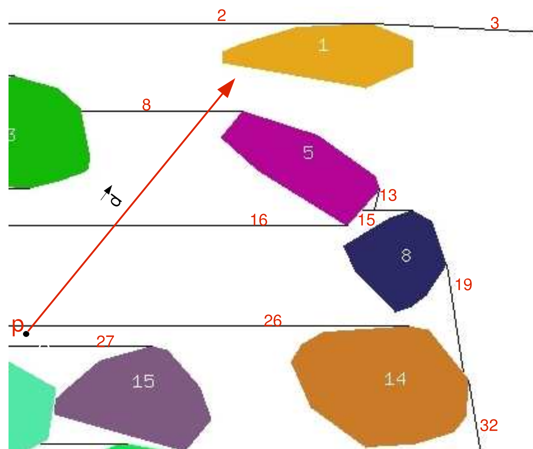


Fig. 5: Ray traverse of $r(t) = p + td$

$\mathscr{P}_{(E)}(o_{14})$ is in fact a free configuration space in which $r(t)$ can move freely without colliding with any obstacle except its boundary. The frontier of a polar region is defined by either polar edges or generator polygons. If $r(t)$ intersects with a polar edge, then it crosses to the polar region of an adjacent object. In the example the ray crosses towards the polar region of o_8. This circumstance is computed in constant time because polar edges store adjacency information. Again the polar region $\mathscr{P}_{(E)}(o_8)$ provides a free space to move until a new intersection if performed. In this case $r(t)$ enters in the polar region of a new object o_5 before arriving to $\mathscr{P}_{(E)}(o_1)$. There, $r(t)$ finally intersects with o_1 completing the ray shooting query.

Theorem 2. *Given a scene with n polygonal objects, the visible ray shooting query from the viewpoint p in an arbitrary direction $\vec{d}$ (if any), can be solved using the four polar diagrams: $\mathscr{PD}(E) = \{\mathscr{P}_{0+}(E), \mathscr{P}_{0-}(E), \mathscr{P}_{\pi+}(E), \mathscr{P}_{\pi-}(E)\}$ in $O(\log n + n)$, being n the number of polygons in the scene.*

Proof. The time for locating p into a polar region is $O(\log n)$. According to Algorithm 1, in the worst case the ray $r(t)$ must cross n regions to find an intersection or to leave the scene. The processing time for determining that this ray crosses each regions depends on the number of edges defining the scene. The number of polar edges of the scene E is $O(2n)$, with an average number of two edges per region. The rest of the frontier edges are the scene polygons.

Theorem 2 considers the worst case scenario in which the scene is probably composed of small polygonal objects and the ray $r(t)$ does not collide with any of them. Thus, the ray crosses even n polar regions computing intersections before leaving the scene without finding collisions. However in dense scenes this computation time can be considered as logarithmic, the location time, since it is assumed that the ray will intersect in only several loops of the algorithm.

Then, the visibility resolution comes from the same principle which allows the Voronoi diagram to solve collisions in [18]. However Voronoi diagrams for polygonal objects are more complex to compute, generating curved edges. In addition polar diagrams provide conservativity when determining visibility in a view frustum, as described in the next section.

4.3 Visibility map determination in 2D scenes

In practice, the ray shooting query that finds the visible object from p in direction $\vec{d}$ is closely related to obtain the intersecting object with the ray $r(t) = p + t\vec{d}$. Visibility is more oriented to find a set of visible objects from a view frustum or viewing angle. In any case, if a ray $r(t)$ is able to find a visible object, on the same principle, a set of selected rays may obtain the visible set from p in an angular range. This is the basis for ray-tracing techniques in computer graphics for visibility culling. Illumination also uses ray-tracing techniques to consider reflexion, refraction or dispersion of the light when rendering complex scenes.

© 2013 NSP
Natural Sciences Publishing Cor.

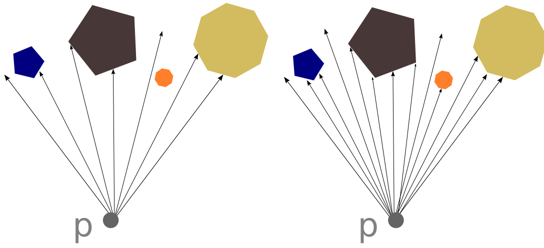


Fig. 6: Different density of rays for visibility detection.

A simple technique of ray shooting for visibility may consist of computing the intersection of an uniform fan of rays, as Figure 6 shows, for the 2D version of this problem. Each time that a ray starting from the viewpoint p intersects with an object, it is considered as visible. Although this approach is simple, two issues arise: (1) which number of rays must be shot in order to obtain a *conservative visibility set*? and (2) how to speed up this process to avoid sequential searches? The first property ensures that all visible objects are detected. However, if the fan of rays is small as in the left example of the figure, a visible object may not be reached by any ray. If the number of rays is increased as in the example on the right, then more computation time is required without ensuring conservativity.

Therefore, the solution should be adaptive to the number and disposition of the objects in the scene. The ideal configuration is a fan of k rays to find k visible objects, that is $O(c \cdot k)$ rays, considering c as a constant. The goal is to find the visibility map as Definition 1 states.

Definition 1. *Let be $VM_p(r_L, r_R)$ the visibility map from point p looking at the scene with the viewing angle $[r_L, r_R]$, considering r_L and r_R the left and right rays defining this angular sector respectively. The set $VM_p(r_L, r_R)$ contains those objects that are visible from p at least partly.*

The definition of $VM_p(r_L, r_R)$ is valid in computer graphics because the hidden portions of the objects are detected by the depth map implemented by the Z-buffer. In fact, the Z-buffer does not require any additional processing to find the visible set. However, a complex scene with many hidden primitives needs too much time for rendering objects that will never be visible. Therefore it is preferable to reduce the number of primitives sent to the Z-buffer, allowing this visibility culling process works faster. This reasoning is further justified when the scene is to be transferred over the network.

The polar diagram helps to speed up the visibility determination process in all these phases:

– the ray shooting query of the ray $r(t) = p + t\overrightarrow{d}$ can be solved efficiently (Algorithm 1),
– the fan of rays is adaptive to the visible set so that only $c \cdot k$ rays are needed to be shot,
– the conservativity of the visible set is also guaranteed.

The first issue, studied in Section 4.2, allows to perform efficient intersections specially in densely occluded scenes. The rest of features, studied below, also contributes to an efficient and accurate method.

4.3.1 Determining the fan of rays

Consider again the situation of Figure 3.b) in which the maximum visibility angle is obtained by means of tangent lines to objects A and C. Tangent lines determine the dividing lines between the visible and non-visible portion of the scene. Tangent lines and polar diagrams are the basis of the methodology used in this section to obtain the visibility map.

This ray shooting process denoted as *Visbility_Map* is described in Algorithm 3. In a first time, the set of polar diagrams that match with the viewing angle $[r_L, r_R]$ is selected. If $\mathbf{r_L}$ and $\mathbf{r_R}$ are located in different quadrants, the angular interval $[\mathbf{r_L}, \mathbf{r_R}]$ is divided into the resulting sub-intervals from the intersection with the coordinate axes. The visibility determination process is performed independently in each sub-intervals $[\mathbf{r_l}, \mathbf{r_r}] \subseteq [\mathbf{r_L}, \mathbf{r_R}]$, using a different polar diagram according to Figure 4, and combining the partial results at the end of the process. For each sub-interval, p is located in a polar region and the visibility map is obtained as described in this algorithm.

The tree-based data structure T, maintains the fan of rays angularly sorted from left to right (clockwise). T is initialized with rays $\mathbf{r_l}$ and $\mathbf{r_r}$. The rays inserted in T are new tangent lines from p toward any object considered as visible. Each time that a ray r_i is shot using Algorithm 2, it is obtained the list l_i of polar edges crossed by r_i, as well as the collided object o. Each time that a new visible object is obtained is inserted into V, the set of visible objects from p.

In summary, if the ray r_i intersects with object A, then A is inserted in V and its left and right tangent lines from p are inserted in T waiting to be processed in angular order. However, a new tangent ray may not add new visibility information because it is located in the so called *adjacent rays*. In order to check adjacency, the set $R = \{(r_l, l_l, o_l), (r_{l+1}, l_{l+1}, o_{l+1}), ..., (r_{r-1}, l_{r-1}, o_{r-1}), (r_r, l_r, o_r)\}$ maintains the fan of rays shot, as well as the lists of edges that they cross in the form $l_i = \{ei_1, ei_2, ...ei_s\}$. Between two adjacent rays there are no new visible objects. Therefore, when the algorithm finishes, each pair of consecutive tuples $(r_i, l_i, o_i) - (r_{i+1}, l_{i+1}, o_{i+1})$ in R must represent adjacent rays (Definition 2). Otherwise Algorithm 4 searches for potential visible objects, as explained in the following section.

Definition 2. *Two shot rays r_i and r_j represented by the tuples (r_i, l_i, o_i) and (r_j, l_j, o_j), are adjacent if the angular sector that they define visualizes as much a single object.*

© 2013 NSP
Natural Sciences Publishing Cor.

Table 2: Steps followed by Algorithm 3 in the scene of Figure 7.

shoot ray	collide	tangent to	R	T	V	$Adjacency$
			$\emptyset$	$\{r_l, r_r\}$	$\emptyset$	-
r_l	A	$-$	$\{(r_l, l_l = e_7), A)\}$	$\{r_1, r_r\}$	$\{A\}$	$[l_l]$
r_1	B	A	$\{(r_l, l_l = \{e_7\}, A),$ $(r_1, l_1 = \{e_7, e_4, e_2\}, B)\}$	$\{r_2, r_r\}$	$\{A, B\}$	$[l_l, l_1]$
r_2	D	B	$\{(r_l, l_l = \{e_7\}, A),$ $(r_1, l_1 = \{e_7, e_4, e_2\}, B),$ $(r_2, l_2 = \{e_7\}, D)\}$	$\{r_3, r_4, r_r\}$	$\{A, B, D\}$	$[l_l, l_1][l_2]$
r_3	B	D	$\{(r_l, l_l = \{e_7\}, A),$ $(r_1, l_1 = \{e_7, e_4, e_2\}, B),$ $(r_3, l_3 = \{e_7, e_4, e_2\}, B),$ $(r_2, l_2 = \{e_7\}, D)\}$	$\{r_4, r_r\}$	$\{A, B, D\}$	$[l_l, l_1, l_3, l_2]$
r_4	E	D	$\{(r_l, l_l = \{e_7\}, A),$ $(r_1, l_1 = \{e_7, e_4, e_2\}, B),$ $(r_3, l_3 = \{e_7, e_4, e_2\}, B),$ $(r_2, l_2 = \{e_7\}, D),$ $(r_4, l_4 = \emptyset, E)\}$	$\{r_r\}$	$\{A, B, D, E\}$	$[l_l, l_1, l_3, l_2, l_4]$
r_r	E	$-$	$\{(r_l, l_l = \{e_7\}, A),$ $(r_1, l_1 = \{e_7, e_4, e_2\}, B),$ $(r_3, l_3 = \{e_7, e_4, e_2\}, B),$ $(r_2, l_2 = \{e_7\}, D),$ $(r_4, l_4 = \emptyset, E),$ $(r_r, l_r = \emptyset, E)\}$	$\{\}$	$\{A, B, D, E\}$	$[l_l, l_1, l_3, l_2, l_4, l_r]$

The fun of rays represented in Figure 7 given in clockwise order $\{r_l, r_1, r_3, r_2, r_4, r_r\}$ are adjacent:

–(r_l, l_l, o_l) - (r_1, l_1, o_1) visualizes A
–(r_1, l_1, o_1)-(r_3, l_3, o_3) visualizes B
–(r_3, l_3, o_3)-(r_2, l_2, o_2) visualizes D
–(r_2, l_2, o_2)-(r_4, l_4, o_4) visualizes D
–(r_4, l_4, o_4)-(r_r, l_r, o_r) visualizes E

Lemma 2. *Given the pair of tuples (r_i, l_i, o_i) and (r_j, l_j, o_j), if $l_i = l_j$ and $o_i = o_j$, then r_i and r_j are adjacent.*

Proof. If $l_i = l_j$ and $o_i = o_j$, then r_i and r_j cross exactly the same polar regions before intersecting. If there were a different object between r_i and r_j, then necessarily $l_i \neq l_j$.

In the example of Figure 7, Lemma 2 determines that r_1 and r_3, as well as r_4 and r_r are adjacent rays. The rest of pairs ensures adjacency by means of Lemma 3, which ensures adjacency in the first three cases depicted in Figure 8, the last one corresponds to Lemma 2.

Lemma 3. *Given the pair of tuples (r_i, l_i, o_i) and (r_j, l_j, o_j), defined in a 2D scene, $l_i \neq l_j$, the rays r_i and r_j are adjacent if $\exists o_k$ such that:*

–*if $o_k = o_i$ then r_j is a tangent ray to o_k or, (first example of Figure 8)*
–*if $o_k = o_j$ then r_i is a tangent ray to o_k or, (second example of Figure 8)*
–*if $o_k \neq o_i \neq o_j$ then r_i and r_j are tangent rays to o_k (third example of Figure 8)*

and the sequence of edges before touching o_k must be equal; otherwise let be ES the set of edges in which they differ before touching o_k; r_i and r_j are adjacent if for each $e_s \in ES$ then:

–*e_s is a polar edge of object o_k, $e_s \in PE(o_k)$, or*
–*e_s is a polar edge of a different object o_t, $o_t \neq o_k$ that is not located in the angular range $[r_i, r_j]$.*

Proof. Lists l_i and l_j begin with the same sequence because rays r_i and r_j start from point p. If both lists are equal before touching object o_k, then both rays cross the same polar regions and no other object can be located between r_i and r_j; otherwise l_i and l_j could not be equal according to Lemma 2.

If both lists are different before touching object o_k and the polar edge $e_s \in ES$ belongs to o_k, it means that no other object can lie between r_l and r_j. Otherwise if $e_s \in PE(o_t)$, necessarily o_t must be out of this angular range to ensure adjacency.

In the example of Figure 7, $r_l = \{e_7\}$ and $r_1 = \{e_7, e_4, e_2\}$ are adjacent rays according to Lemma 3. In this case $o_k = A$ as the first example of Figure 8. If r_l is considered as r_i and r_1 as r_j, then the first different edge is $e_4 \in PE(D)$, a polar edge that belongs to D, an object out of range. The cases of r_3-r_2 and r_2-r_4 are similar.

The steps that Algorithm 3 follows to find the visibility map from point p in the scene of Figure 7 are summarized in Table 2. Firstly the rays defining the viewing angle r_l and r_r are inserted in T waiting to be shot in clockwise order. The ray r_l is thrown reaching object A, which becomes visible. Only the right tangent to

© 2013 NSP
Natural Sciences Publishing Cor.

Algorithm 3 $VivisilityMap(E, p, r_L, r_R)$

Input: The four polar diagrams of $E: \mathscr{PD}(E) = \{\mathscr{P}_{0-}(E), \mathscr{P}_{0+}(E), \mathscr{P}_{\pi+}(E), \mathscr{P}_{\pi-}(E)\}$
-The rays $r_L = p + t\overrightarrow{d_L}$ and $r_R = p + t\overrightarrow{d_R}$ defining the viewingangle.

Output: $VM_p(r_L, r_R)$

Var: T: a tree-based data structure of angularly sorted rays
- R: set of 3-tuplas containing rays, lists of crossed edges and the collided object o, $R = \{(r_L, l_L, o_L), (r_{L+1}, l_{L+1}, o_{L+1}), ..., (r_{R-1}, l_{R-1}, o_{R-1}), (r_R, l_R, o_R)\}$

BEGIN
1. Initialize V, $V \leftarrow \emptyset$
2. **FOR EACH** angular sector in a quadrant $[r_l, r_r] \subset [r_L, r_R]$ **DO**
(a) Select the polar diagram $\mathscr{P}(E) \in \mathscr{PD}(E)$ according to Figure 4
(b) Initialize T and R, $T \leftarrow \emptyset$, $R \leftarrow \emptyset$
(c) Locate p in the polar region of object q, $p \in \mathscr{P}_{(E)}(q)$
(d) Insert r_l and r_r clockwise sorted in T, $T \leftarrow T + \{r_l, r_r\}$
(e) bool $adjacency \leftarrow$ false
(f) **REPEAT**
i. **WHILE** (T is not empty) **DO**
A. Get the first ray r of T, $T \leftarrow T - \{r\}$
B. RayCollision($\mathscr{P}(E)$, $r(t)$, q):(o, l) *(Shoot ray $r(t)$ using Algorithm 2)*
C. **IF** $o \neq \emptyset$ **AND** $o \notin V$ *(if the there is a new visible object)*
D. **THEN**
-insert o in V, $V \leftarrow V + \{o\}$
-let be rt_l the left tangent ray to o
-IF $rt_l \subset [r_l, r_r]$ AND is not within a pair of adjacent rays
-THEN insert rt_l clockwise sorted in T, $T \leftarrow T + \{rt_l\}$
-let be rt_r the right tangent ray to o
-IF $rt_r \subset [r_l, r_r]$ AND is not within a pair of adjacent rays
-THEN insert rt_r clockwise sorted in T, $T \leftarrow T + \{rt_r\}$
E. Insert (r, l, o) into R, $R \leftarrow R + \{(r, l, o)\}$
ii. **END_WHILE**
iii. **IF** all consecutive pair of tuples $(r_i, l_i, o_i) - (r_{i+1}, l_{i+1}, o_{i+1})$ are adjacent
iv. **THEN** $adjacency \leftarrow$true
v. **ELSE** $T \leftarrow T + find_potential_VO$ $((r_i, l_i, o_i), (r_{i+1}, l_{i+1}, o_{i+1}))$
(g) **UNTIL** $adjacency$ =true
3. **END_FOREACH**
4. **RETURN** V
END

Algorithm 4 $find_Potential_VO((r_i, l_i, o_i), (r_j, l_j, o_j))$: r_A, r_B

Input: The tuples $((r_i, l_i, o_i)$ and $(r_{i+1}, l_{i+1}, o_{i+1})$, $l_i = \{ei_0, ei_1, ..., ei_m\}$, $l_j = \{ej_0, ej_1, ..., ej_q\}$
Output: The left and right tangents to the potentially visible object o

BEGIN
1. **FOREACH** $ei_k \in l_i$ **DO**
(a) **IF** $ei_k \notin l_j$ *(if this edge is not in l_j it may belong to a visible object)*
(b) **THEN**
i. let o be the object such that $ei_k \in PE_E(o)$
ii. **IF** o is in the range $[r_i, r_j]$
iii. **THEN RETURN** left and right tangents to o r_A and r_B
2. **END_FOR**

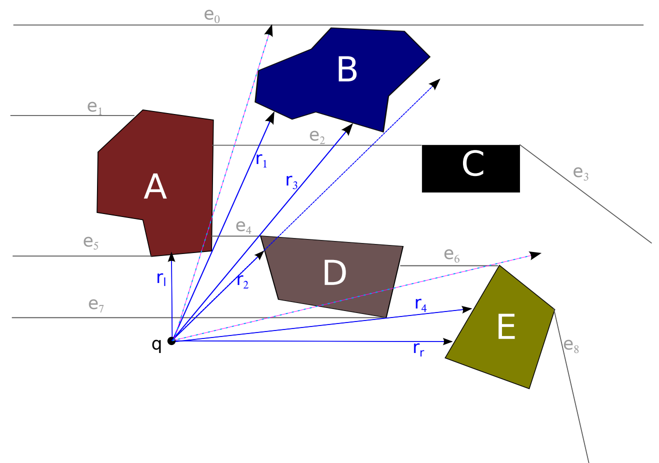


Fig. 7: Example of execution of Algorithm 3

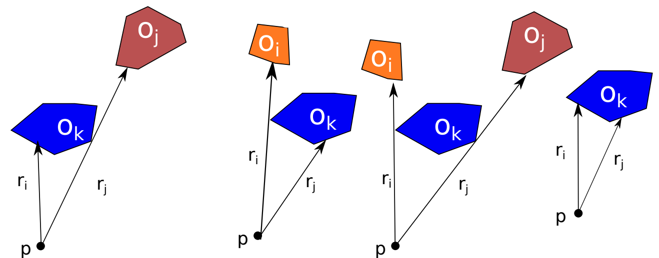


Fig. 8: Different adjacent situations associated with Lemma 3.

© 2013 NSP
Natural Sciences Publishing Cor.

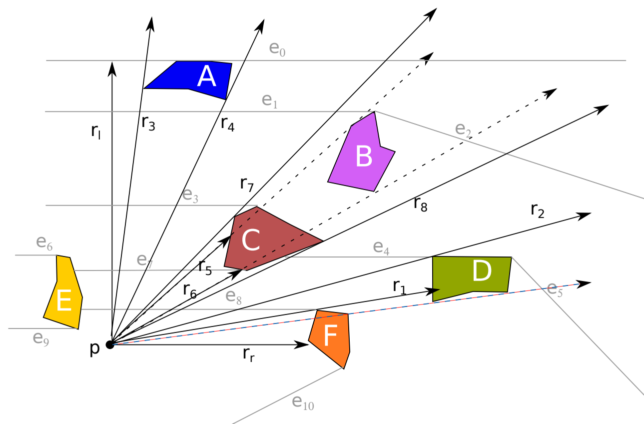


Fig. 9: Example of execution of Algorithm 3 in a sparse scene.

object A, ray r_1 is inserted in T because the left one if out of the first quadrant. When r_1 is shot, it reaches to object B, and its right and left tangent lines processed. However, the left tangent is not inserted in T since the angular range in which it lies does not provide new visibility information. It is formed by $[r_l - r_1]$, a pair of rays already considered as adjacent according to Lemma 3 (in the figure this ray is depicted with dotted lines). When a pair of rays are adjacent, their lists of crossed edges appear into the same bracket in the column *adjacency*.

The right tangent to B, the ray r_2 is shot intersecting with object D. Left and right tangents to D, r_3 and r_4 are then inserted in T to be processed later. R maintains the tree rays shot so far. The next ray to shot in angular order is r_3, tangent to object D. It collides with B, previously processed, thus its tangent lines are not inserted in T again. Then r_4 is thrown intersecting with object E, a new visible object that is inserted in V. Again the left tangent line to object E is an adjacent ray to r_2 (Lemma 2) and to r_4 (Lemma 3), then it does not provide new visible objects and is not inserted in T. The right tangent line to object E is out of range and is neither processed. The last ray shot, r_r finds object E too, which is adjacent to its neighbor r_4 in R. The set of rays T becomes empty and Algorithm 3 finishes. The algorithm checks that all the pairs of tuples in R are adjacent, as in this example. Each angular sector defined by these pairs of tuples represents the view-frustum towards a single visible object. The set of angular sectors R represents the information associated with the visibility map of point p in the angular range $[r_L, r_R]$, that is $VM_p(r_L, r_R)$.

4.3.2 Adjacency check for preserving conservativity

In the example developed in Figure 7, all rays intersect with any object in the scene. In these cases each ray is able to find new visible objecs, and the fan of rays maintained in R after the iterative process has the property of being adjacent. However, if one or several tangent rays does not intersect with any object, what happens in sparse scenes, adjacency is not guaranteed. In this section we specify the process to get a conservative set of visible objects.

If all the pairs of consecutive tuples inserted in R are not adjacent using the method proposed in Algorithm 3, then there may be visible objects not detected. In case that the tuples $(r_i, l_i, o_i) - (r_{i+1}, l_{i+1}, o_{i+1})$ are not adjacent, Algorithm 4 is responsible of finding a possible visible object to follow with the same process. For this purpose the list l_i is checked to find an edge $e \in l_i$, such that e belongs to the polar region of an object o, that has not yet been considered visible, $o \notin V$ and is in the angular range $[r_i, r_{i+1}]$. This candidate may be visible or not, therefore tangent rays towards object o are inserted in T. The inner loop WHILE of Algorithm 3 will cycle again because in these cases T becomes non-empty. The REPEAT-UNTIL loop finishes when all pairs of tuples become adjacent.

Figure 9 represents a scene with scattered objects representing small occluders. This means that many tangent rays may not intersect with any objects. The sequence of steps that this algorithm performs is described in Table 3. The column $Adjacency$ represents into the same bracket those lists corresponding to adjacent rays. The adjacency must be searched between the pair of rays without satisfying this property.

The first steps are similar to the example of Figure 7 because T maintains rays to process. However when it becomes empty after shooting r_2, which does not find any intersecting object, the lists $l_l = \{e_8, e_7, e_3, e_1, e_0\}$ and $l_2 = \{e_8, e_4\}$ are not adjacent. Then the routine described in Algorithm 4, $find_potential_VO((r_l, l_l, o_l)(r_2, l_2, o_2))$ determines that $e_0 \in l_l$, $e_0 \notin l_r$, can be associated with a visible object in the angular range $[r_l, r_2]$. The edge e_0 is a polar edge of object A, $e_0 \in PE_E(A)$, and if its tangent rays r_3 and r_4, lies in the angular range $[r_l, r_2]$, then A may be visible, thus both rays are inserted in T to be processed. Because r_3 touches object A, then it is inserted in V, and lists r_l and r_3 are adjacent according to Lemma 2. When r_4 is shot it also touches object A in the way that l_3 and l_4 become adjacent too.

Again T is empty and l_4 and l_2 are not adjacent at all. Then list l_4 is checked again in Algorithm 4 to find that edge e_1 belongs to an object B in the range $[r_4, r_2]$ which generates two tangent rays toward B, the rays r_5 and r_6. However B is never reached by r_5 or r_6 which means that is hidden by the visible object C. When tangent rays to C, r_7 and r_8 are shot, all the lists become adjacent.

Rays r_6 and r_8 are also adjacent according to Lemma 3 even when they differ in several objects $l_6 = \{e_8, e_7\}$, $l_8 = \{e_8, e_4, e_2, e_0\}$. Both rays r_6 and r_8 touch object C (first example of Figure 8). The list ES has the edges in which they differ before touching C, $ES = \{e_7, e_4\}$. This pair of edges is analyzed independently: e_7 is a polar edge of object $C = o_k$ and e_4 is a polar edge of object D that is out of the range $[r_6, r_8]$. Therefore these rays are adjacent because both situations are referred in this lemma.

© 2013 NSP
Natural Sciences Publishing Cor.

Table 3: Steps followed by Algorithm 3 in the scene of Figure 9.

ray	collide	R	T	V	$Adjacency$
		$\emptyset$	$\{r_l, r_r\}$	$\{\emptyset\}$	-
r_l	$\emptyset$	$\{(r_l, l_l = \{e_8, e_7, e_3, e_1, e_0\}, \emptyset)\}$	$\{r_r\}$	$\{\emptyset\}$	$[l_l]$
r_r	F	$\{(r_l, l_l),$ $(r_r, l_r = \emptyset, F)\}$	$\{r_1\}$	$\{F\}$	$[l_l][l_r]$
r_1	D	$\{(r_l, l_l),$ $(r_1, l_1 = \{e_8\}, D),$ $(r_r, l_r)\}$	$\{r_2\}$	$\{F, D\}$	$[l_l][l_1 l_r]$
r_2	$\emptyset$	$\{(r_l, l_l),\ (r_2, l_2 = \{e_8, e_4, e_2, e_0\}, \emptyset),$ $(r_1, l_1),\ (r_r, l_r)\}$	$\{\}$	$\{F, D\}$	$[l_l]$ $[l_2, l_1, l_r]$
execute $find_potential_VO((r_l, l_l, o_l)(r_2, l_2, o_2))$:$A$			$\{r_3, r_4\}$		
r_3	$\emptyset$	$\{(r_l, l_l),$ $(r_3, l_3 = \{e_8, e_7, e_3, e_1, e_0\}, \emptyset),\ (r_2, l_2),$ $(r_1, l_1),\ (r_r, l_r)\}$	$\{r_4\}$	$\{F, D, A\}$	$[l_l, l_3]$ $[l_2, l_1, l_r]$
r_4	$\emptyset$	$\{(r_l, l_l),\ (r_3, l_3),$ $(r_4, l_4 = \{e_8, e_7, e_3, e_1, e_0\}, \emptyset),\ (r_2, l_2),$ $(r_1, l_1),\ (r_r, l_r)\}$	$\{\}$	$\{F, D, A\}$	$[l_l, l_3, l_4]$ $[l_2, l_1, l_r]$
execute $find_potential_VO((r_4, l_4, o_4)(r_2, l_2, o_2))$:$B$			$\{r_5, r_6\}$		
r_5	C	$\{(r_l, l_l),\ (r_3, l_3),$ $(r_4, l_4),\ (r_5, l_5 = \{e_8, e_7\}, C),$ $(r_2, l_2),$ $(r_1, l_1),\ (r_r, l_r)\}$	$\{r_7, r_6, r_8\}$	$\{F, D, A, C\}$	$[l_l, l_3, l_4]$ $[l_5]$ $[l_2, l_1, l_r]$
r_7	$\emptyset$	$\{(r_l, l_l),\ (r_3, l_3),$ $(r_4, l_4),\ (r_7, l_7 = \{e_8, e_7, e_3, e_1, e_0\}, \emptyset),$ $(r_5, l_5),\ (r_2, l_2),$ $(r_1, l_1),\ (r_r, l_r)\}$	$\{r_6, r_8\}$	$\{F, D, A, C\}$	$[l_l, l_3, l_4, l_7, l_5]$ $[l_2, l_1, l_r]$
r_6	C	$\{(r_l, l_l),\ (r_3, l_3),$ $(r_4, l_4),\ (r_7, l_7),$ $(r_5, l_5),\ (r_6, l_6 = \{e_8, e_7\}, C),$ $(r_2, l_2),$ $(r_1, l_1),\ (r_r, l_r)\}$	$\{r_8\}$	$\{F, D, A, C\}$	$[l_l, l_3, l_4, l_7, l_5, l_6]$ $[l_2, l_1, l_r]$
r_8	$\emptyset$	$\{(r_l, l_l),\ (r_3, l_3),$ $(r_4, l_4),\ (r_7, l_7),$ $(r_5, l_5),\ (r_6, l_6),$ $(r_8, l_8 = \{e_8, e_4, e_2, e_0\}, \emptyset),\ (r_2, l_2),$ $(r_1, l_1),\ (r_r, l_r\}$	$\{\}$	$\{F, D, A, C\}$	$[l_l, l_3, l_4, l_7, l_5,$ $l_6, l_8, l_2, l_1, l_r]$

4.3.3 Efficiency

The worst case of Algorithm 3 arises when all the scene is visible from point p, what happens in sparse scenes with small objects. In this case it is required to throw $c \cdot n$ rays, and each of them needs to cross n polar regions.

Lemma 4.*Algorithm 3 throws $O(4k)$ rays to find k visible objects from point p in a scene with n polygonal objects.*

*Proof.*Each visible object $o \in V$ is obtained by shooting tangent rays until each pair of rays in R becomes adjacent. In dense scenes, if each tangent ray reaches to any other visible object, the fan of rays in R is adjacent without running Algorithm 4. In this case only $2 \cdot k$ rays are required. The rays that lies between adjacent rays are not thrown.

The worst case is found in sparse scenes in which tangent rays are thrown to the potentially visible object o_i. If o_j is reached instead of o_i, then new tangent rays to o_j are thrown. In theses cases four tangent rays are shot to find one visible object. Again, no new rays are thrown in angular ranges defined by adjacent rays.

In the example of Figure 9, object A is found throwing only two rays, r_3 and r_4. However object C has required four rays to ensure adjacency in its angular spectrum. Rays r_5 and r_6 are tangent rays to a potentially visible object B, that is finally hidden by C. Two new tangent rays to object C, r_7 and r_8, are then required.

Theorem 3.*Given a scene E with n polygonal objects, the visibility map from point p, $VM_p(r_L, r_R)$ can be found in $O(k \cdot n)$ for k visible objects, using the four polar diagrams $\mathscr{P}_{0+}(E)$, $\mathscr{P}_{0-}(E)$, $\mathscr{P}_{\pi+}(E)$ and $\mathscr{P}_{\pi-}(E)$, as preprocessing.*

*Proof.*Lemma 4 asserts that at most each visible object requires four rays to be found. According to Lemma 2 a ray requires $O(\log n + n)$ to find its intersecting object;

© 2013 NSP
Natural Sciences Publishing Cor.

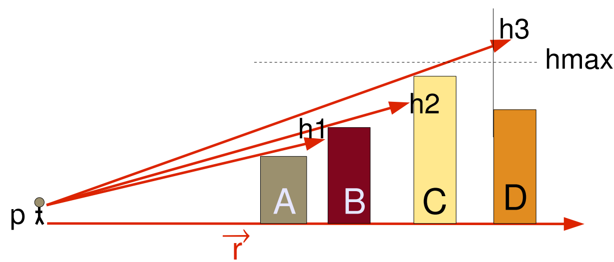


Fig. 10: Throwing a $2.5D$ ray.

however the logarithmic time is performed only once, then it requires only linear time. Algorithm 4 needs as much linear time to find a candidate. In the worst case, to find k visible objects $O(\log n + k.n + n)$ is required to find the visibility map.

4.4 Visibility map in 3D prism-shape scenes

Visibility determination in 2D scenes is a key topic in computational geometry and robotics. However, in computer graphics visibility must be solved for 3D scenes. As stated in the Introduction section, the hardware graphics acceleration option can solve this problem directly, however visibility remains crucial in large scenes with thousands of objects in which most of them are occluded, and specially in mobile devices with limited graphics capabilities and connected to the Internet.

The ray shooting technique described in Section 4.3 can be extended to prism-shaped objects constructed by extruding 2D polygons. These simple models, also called $2.5D$ objects, are very useful to represent buildings in virtual cities, and as bounding boxes that enclose other models with complex geometry.

The ray shooting process described in Algorithm 3 can also be used for $2.5D$ scenes, since only the geometry of the base and the height of the $2.5D$ object are considered. The main difference is the behavior of the ray shooting query, that has been adapted in Algorithm 5. While in $2D$ scenes the ray stops when the first obstacle is found, in prism-shaped objects there can be partially visible objects behind. In contrast, the ray in $2.5D$ scenes must follow its trajectory to detect half-occluded objects, as in the situation of Figure 10. Initially the ray is shot in the XZ plane (at the observer height), but it takes the gradient of each object found. In fact the ray projection in the XZ plane compute intersections as in the $2D$ case, but it also considers the height of each intersected object. As observed in the same figure, the ray first reaches object A, then it takes the gradient considering the height of A, that also intersects with object B at height $h1$. As $height(B) > h1$, then B is considered as visible. This process can follow until the ray leaves the scene or until it takes the maximum height of the scene $hmax$, a value

Algorithm 5 RayShootingQuery2.5D($\mathscr{P}(E)$, $r(t) = p + t\vec{d}$, $hmax$, o):$\{lp, V\}$

Input: The polar diagram $\mathscr{P}(E) \in \mathscr{PD}(E)$ in which $r(t)$ lies
- The ray $r(t) = p + t\vec{d}$ defining the visibility direction.
- The maximum height of the scene $hmax$
- The object o such that $p \in \mathscr{P}_{(E)}(o)$

Output: The list lp defining the polar regions crossed by $r(t)$;
-V the set of visible objects that finds $r(t)$

BEGIN
1. **BOOL** terminate ← **false**
2. Initialize $lp \leftarrow \emptyset$ and $o_i \leftarrow o$
3. Initialize $V \leftarrow \emptyset$
4. **WHILE** (**NOT** terminate) **DO**

(a) Determine the edge e intersecting with $r(t)$ in $\mathscr{P}_{(E)}(o_i)$
(b) **IF** e is a polygon edge
(c) **THEN**
 i. determine h the height of $r(t)$ when reaching to o_i
 ii. **IF** $h < height_max(o_i)$ AND $h > height_min(o_i)$
 iii. **THEN** $V \leftarrow V + \{o_i\}$ AND $r(t)$ adopts the gradient of $height_max(o_i)$
 iv. **IF** $h > hmax$ terminate←**true** *(out of height)*
 v. Insert o_i in lp, $lp \leftarrow lp + \{o_i\}$
(d) **ELSE** *(go to an adjacent region, open a new subset)*
 i. Insert edge e in lp, $lp \leftarrow lp + \{e\}$
 ii. Let be o_j the object whose polar region is adjacent to edge e
 iii. $o_i \leftarrow o_j$
 iv. **IF** $e = 0$ *(out of the scene)*
 v. **THEN** terminate ←**true** *(o_i is null)*

5. **RETURN** lp and V

END

which ensures that this ray will not find any other visible object. On the other hand, if the scene is not settled on a flat surface, each $2.5D$ object has associated two different heights, $height_min(A)$ and $height_max(A)$ as Algorithm 5 refers.

The second difference is the way in which adjacency is checked. The tuples (r_i, l_i, o_i) defined in Section 4 assume that the ray r_i only intersects with object o_i, however in prims-shaped scenes the ray can cross several polar regions and objects, and compute several intersections. In this case the tuple is replaced by the pair (r_i, lp_i), the ray r_i and the list of crossed polar regions lp_i, which also contains the objects intersected. Thus, a ray crosses objects and polar edges, which is to say polar

© 2013 NSP
Natural Sciences Publishing Cor.

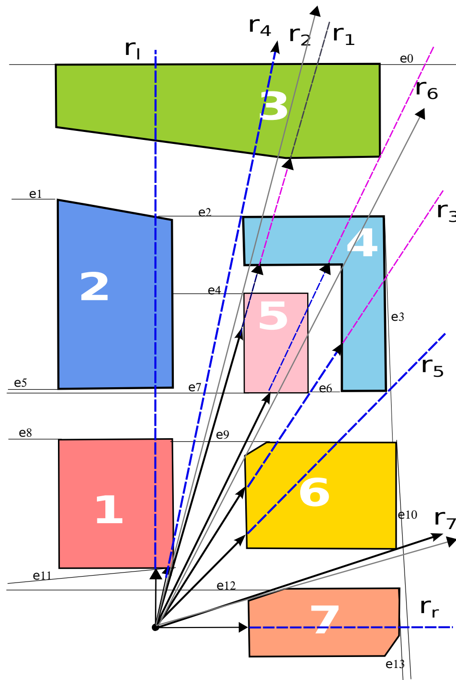


Fig. 11: Visibility map in $2.5D$ scenes.

Algorithm 6 AdjacentRays2.5D((r_i, lp_i) , (r_j, lp_j)):*boolean*

Input: the pair of tuples (r_i, lp_i) and (r_j, lp_j), defined in a $2.5D$ scene
Output: true if r_i and r_j are adjacent; false otherwise

BEGIN
1. **IF** $lp_i = lp_j$
2. **THEN RETURN true**
3. **IF** $lp_i \neq lp_j$,
4. **THEN** let be ES the set of different polar regions in which they differ

(a) **FOREACH** $pr_k \in ES$,
  i. **IF** the object defining the polar region pr_k is out of the range defined by $[r_i, r_l]$,
  ii. **THEN** $ES \leftarrow ES - \{pr_k\}$

(b) **END_FOREACH**

5. **IF** $|ES| = \emptyset$ or $|ES| = 1$
6. **THEN RETURN true**
7. **ELSE RETURN false**

END

regions. In this case we define a set of of subsets, each of them representing a crossed polar region. The case of r_l in Figure 11 is defined as: $(r_l, lp_l = \{\{e_{12}\}, \{e_{11}, o_1\}, \{e_7\}, \{o_2\}, \{o_3, e_0\}\})$. Each subset into lp_l represents a polar region by means of its polar edges and/or its generating polygon. Algorithm 5 describes the process for generating this ray shooting query.

The definition of adjacency differs of Definition 2 because an angular sector may contain many objects. However adjacency is guaranteed if both rays crosses the same polar regions, or if they differ as much in one object or polar region.

Definition 3. *Two rays r_i and r_j, represented by the tuples (r_i, lp_i) and (r_j, lp_j) in a $2.5D$ scene are adjacent if the polar regions that they cross differ as much in one object.*

Adjacency determination is then a similar process to the defined in Lemma 3.

Lemma 5. *Given the pair of tuples (r_i, lp_i) and (r_j, lp_j), defined in a $2.5D$ scene, r_i and r_j are adjacent:*

1. if they cross the same polar regions: $|lp_i| = |lp_j|$ and $\forall k$ such that $\mathscr{P}_{(E)}(o_k) \in lp_i$, then $\mathscr{P}_{(E)}(o_k) \in lp_j$, or
2. if there is one or several polar regions $\mathscr{P}_{(E)}(o_k)$ such that $\forall k$, $\mathscr{P}_{(E)}(o_k) \in lp_i$ and $\mathscr{P}_{(E)}(o_k) \notin lp_i$, then o_k must be out of the range $[r_i, r_j]$ or/and
3. if there is only one polar region such that $\mathscr{P}_{(E)}(o_k) \in lp_i$ and $\mathscr{P}_{(E)}(o_k) \notin lp_i$ and o_k lies is in the range $[r_i, r_j]$.

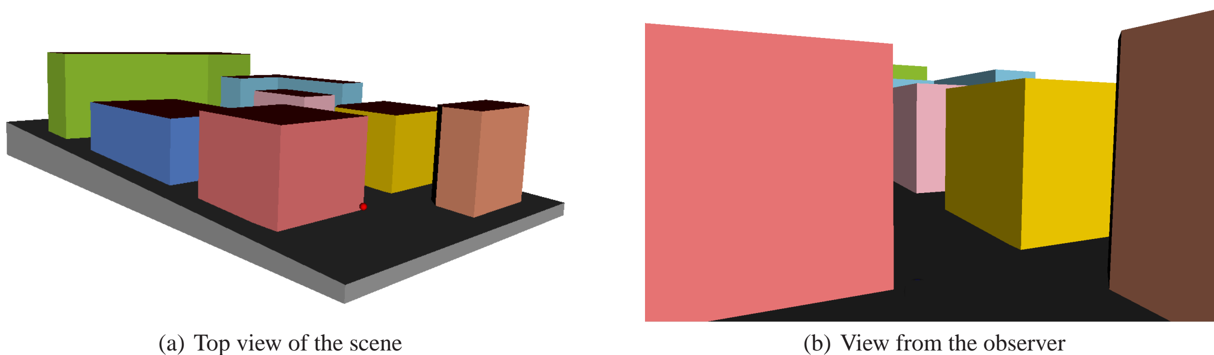
Proof. If both rays crosses the same polar regions, then no other object can be inside the range $[r_i, r_l]$ becoming adjacent rays. If they differ in several polar regions but they are associated with objects out of range, then they are adjacent as well; otherwise they can only differ in one polar region whose generating object is into the angular range $[r_i, r_l]$. In all these cases the range ensures as much a different object inside each adjacent pair of rays.

Algorithm 6 provides the sequence of steps to detect if two rays are adjacent according to Lemma 5. In the example of Figure 11, we observe that all the rays shot are adjacent. Polar regions are represented with the edges or polygons crossed by these rays. For instance $(r_l, lp_l = \{\{e_{12}\}, \{e_{11}, o_1\}, \{e_7\}, \{o_2\}, \{o_3, e_0\}\})$ and $(r_4, lp_4 = \{e_{12}\}, \{o_1\}, \{e_9\}, \{e_7, e_4\}, \{e_2\}, \{o_3, e_0\})$ are not similar but they represent adjacent rays. They differ in edge, $e_9 \in lp_3$ and $e_9 \notin lp_l$ but its associated prism-shaped object o_6 is out of the range $[r_l, r_1]$. The case of e_2 is similar, $e_2 \in lp_3$, $e_2 \notin lp_l$ but it belongs to the object o_4 that is out of range. There is only one allowed element in this angular range in which they differ for being adjacent, o_2, $o_2 \in lp_l$ and $o_2 \notin lp_3$.

Algorithm 7 is the revised version of Algorithm 3 for $2.5D$ scenes. They are essentially the same except for two main differences cited above:

– the ray shooting query in $2D$ is replaced by the *RayShootingQuery2.5D( )* of Algorithm 5, which generates lists of crossed polar regions.
– the adjacency checking process is replaced by the *AdjacentRays2.5D( )* of Algorithm 6.

© 2013 NSP
Natural Sciences Publishing Cor.



(a) Top view of the scene (b) View from the observer

Block 1		Block 2		Block 3		Block 4		Block 5		Block 6		Block 7	
Ground	Roof	Ground	Roof	Ground	Roof	Ground	Roof	Ground	Roof	Ground	Roof	Ground	Roof
5	70	10	60	10	90	10	80	10	70	5	65	0	75

Fig. 12: $3D$ visualization of the scene of Figure 11.

Figure 11 shows the topview of the 3D scene represented in Figure 12.a). It also shows the set of rays that the modified Algorithm 3 generates to determine visibility. Each object in the scene has different heights and also is located at different heights from the ground surface. After applying this visibility determination process from the viewer position (point in red), the resulting set of visible objects is shown in Figure 12.b).

Table 4 summarizes the steps for determining visibility in this scene:

1. r_l is initially shot in parallel to the XZ plane; then it intersects with o_1 that is visible, then r_l takes the gradient of o_1, it intersects with o_2 and o_3 but the height of the ray is greater when reaching to these objects and are considered as non visible. Then, tangent rays to the visible object o_1 are shot, but its left tangent is out of range. The ray r_1, right tangent to o_1, is inserted in T waiting to be processed.
2. The tangent ray to o_1, r_1, hits with o_5 that becomes visible. When the ray takes the gradient of o_5, then it collides with the object behind o_4; again its gradient is modified to the height of o_4 determining that o_3 is also visible. New tangent rays to o_5 (r_2 and r_3), to o_4(r_4 and r_5) as well as the right tangent to o_3 (r_6) are inserted in T in angular order. The lists l_l and l_1 appear in separate brackets $[l_l]\,[l_1]$ indicating that rays r_l and r_1 are non-adjacent rays.
3. The next ray to process in angular order is r_4, that also intersects with o_1. The ray computes in l_4 the crossed regions. Lists l_l and l_4 represent two adjacent rays, as stated before, since this angular sector only contains one different object. On the other hand, l_4 and l_1 also represents to adjacent rays because they cross exactly the same polar regions. Adjacency is represented now with these lists inside the same bracket $[lp_l, lp_4, lp_1]$.
4. The next ray r_2, tangent to r_5, is not finally shot because it lies inside a pair of adjacent rays r_4 and r_1, then no new visibility information is going to be obtained.
5. Next, r_6, the right tangent to o_3 is shot. Since it crosses the same polar regions that its neighbor $r_{1,}$then they become adjacent too.
6. Ray r_3 intersects with o_6 that is considered as visible. Then o_4 is also intersected, but it was already inserted in V and not processed again. The list l_3 is checked for adjacency with l_6, becoming adjacent too. New tangent rays to o_6 should be inserted in T but the left one lies inside an adjacent interval, so it is discarded. The right tangent r_7 is inserted in T.
7. Ray r_5 collides with o_6 and becomes also adjacent to r_3.
8. Ray r_7, that does not hit with any object, is also adjacent to r_3.
9. Finally r_r is shot colliding with o_7. Because it is adjacent to r_3, its left tangent ray is not necessary to be shot.

The list T is empty and the process finishes because all the pairs of lists remain adjacent.

Theorem 4. *Given a scene E with n prism-shaped objects, the visibility map from point p, $VM_p(r_L, r_R)$ can be found in $O(k\,.\,n)$ for k visible objects, using the four polar diagrams $\mathscr{P}_{0+}(E)$, $\mathscr{P}_{0-}(E)$, $\mathscr{P}_{\pi+}(E)$ and $\mathscr{P}_{\pi-}(E)$, as preprocessing.*

Proof. The sequence of steps and the performance of Algorithm 7 are similar to the Algorithm 3. The RayShootingQuery2.5D described in Algorithm 5 does not stop at the first object found and it is expected that crosses several polar regions. However these cases are similar to those in which no intersecting objects are found in Algorithm 7. Each visibility object detected requires as much four rays, each of them shot in $O(n)$ time. The

© 2013 NSP
Natural Sciences Publishing Cor.

Table 4: Steps followed by Algorithm 3 revised for $2.5D$ in the scene of Figure 11.

ray	collide	R	T	V	$Adjacency$
		$\emptyset$	$\{r_l, r_r\}$	$\{\emptyset\}$	-
r_l	o_1	$\{(r_l, lp_l = \{e_{12}\}, \{e_{11}, o_1\}, \{e_7\}, \{o_2\}, \{o_3, e_0\})\}$	$\{r_1, r_r\}$	$\{o_1\}$	$[lp_l]$
r_1	o_5, o_4, o_3	$\{(r_l, lp_l),$ $(r_1, lp_1 = \{e_{12}\}, \{o_1\}, \{e_9\}, \{e_7, o_5\}, \{o_4\}, \{o_3, e_0\})\}$	$\{r_4, r_2, r_6,$ $r_3, r_5, r_r\}$	$\{o_1, o_5, o_4, o_3\}$	$[lp_l]\,[lp_1]$
r_4	o_1	$\{(r_l, lp_l),$ $(r_4, lp_4 = \{e_{12}\}, \{e_{11}, o_1\}, \{e_9\}, \{e_7, e_4\}, \{e_2\}, \{o_3, e_0\}),$ $(r_1, lp_1)\}$	$\{r_2, r_6,$ $r_3, r_5, r_r\}$	$\{o_1, o_5, o_4, o_3\}$	$[lp_l, lp_4, lp_1]$
r_2 is not shot, it lies in the range $[r_4, r_1]$ that is adjacent			$\{r_6, r_3,$ $r_5, r_r\}$		
r_6	o_5, o_4	$\{(r_l, lp_l), (r_4, lp_4), (r_1, lp_1),$ $(r_6, lp_6 = \{e_{12}\}, \{e_9\}, \{o_5\}, \{o_4\}, \{o_3, e_0\}\}$	$\{r_3, r_5, r_r\}$	$\{o_1, o_5, o_4, o_3\}$	$[lp_l, lp_4,$ $lp_1, lp_6]$
r_3	o_6, o_4	$\{(r_l, lp_l), (r_4, lp_4), (r_1, lp_1),$ $(r_6, lp_6), (r_3, lp_3 = \{e_{12}\}, \{o_6\}\{e_6, o_4, e_3\}, \{e_0\})\}$	$\{r_5, r_7, r_r\}$	$\{o_1, o_3, o_5,$ $o_4, o_6\}$	$[lp_l, lp_4, lp_1,$ $lp_6, lp_3]$
r_5	o_6	$\{(r_l, lp_l), (r_4, lp_4), (r_1, lp_1), (r_6, lp_6),$ $(r_5, lp_5 = \{e_{12}\}, \{o_6\}, \{o_4, e_3\}, \{e_0\})\}$	$\{r_7, r_r\}$	$\{o_1, o_3, o_5,$ $o_4, o_6\}$	$[lp_l, lp_4, lp_1,$ $lp_6, lp_3, lp_5]$
r_7	$\emptyset$	$\{(r_l, lp_l), (r_4, lp_4), (r_1, lp_1), (r_6, lp_6),$ $(r_5, lp_5), (r_7, lp_7 = \{e_{12}\}, \{o_6, e_{10}\}, \{e_0\})\}$	$\{r_r\}$	$\{o_1, o_3, o_5,$ $o_4, o_6\}$	$[lp_l, lp_3, lp_1, lp_6,$ $lp_2, lp_4, lp_7]$
r_r	o_7	$\{(r_l, lp_l), (r_4, lp_4), (r_1, lp_1), (r_6, lp_6), (r_5, lp_5),$ $(r_7, lp_7), (r_r, lp_r = \{o_7, e_{13}\}, \{e_{10}\})\}$	$\{\}$	$\{o_1, o_3, o_5,$ $o_4, o_6, o_7\}$	$[lp_l, lp_3, lp_1, lp_6,$ $lp_2, lp_4, lp_7, lp_r]$

location time is $O(\log n)$ and is performed only once for each polar diagram. In the worst case, to find k visible objects it is required $O(\log n + k \cdot n + n)$ to find the visibility map.

In $2D$ scenes the best performance of Algorithm 3 is obtained in dense scenes in which all rays reach to any object, making the process stops. In $2.5D$ scenes the best case is obtained when the ray crosses only few regions. To this end the observer must be in densely occluded scenes and close to the occluders. Then the ray $r(t)$ early takes a high gradient which exceeds *hmax* and makes the traversal process to finish. In general, the number of visible objects k is greater in $2.5D$ scenes which implies that the number of rays shot is also greater. However, the probability that the ray falls at intervals of adjacent rays is also high.

5 Application to walkthough in urban scenes

Urban scenes are very susceptible for requiring an efficient visibility determination method, specially in walkthrough applications. The observer is identified as a pedestrian that navigates through the scene, but only visualizes a small portion of it, at least with detail. Buildings become great occluders and it is assumed that a pedestrian does not see all the scene at a given time. Furthermore, most buildings have prism shape, so the method proposed in this paper is particularly appropriate for urban scenes.

Up to now, visibility in large scenes has been solved by means of occlusion culling techniques that find a potencially visible set that also includes non-visible objects. In web-based systems, not only geometry but texture files must be transmitted through the network, and these non visible objects slow down the interaction process. Using the method proposed in this paper, the exact visibility set of buildings can be obtained by defining the $2.5D$ scene with the footprints of the building blocks and their associated height. These data are retrieved from a spatial database of a real city in the South of Spain, Jaén.

Jaén city is on the side of a mountain, and the Digital Elevation Model of the terrain (DEM) is also considered for determining visibility. The polar diagrams are computed using the polygonal footprints of the blocks and the heights of the ground and the roof level. Algorithm 7 considers that object A is visible if the height of the ray $r(t)$ intersects with A, that is, $r(t) < heightRoof(A)$ and $r(t) > heightGround(A)$ when $r(t)$ reaches with the geometry of object A.

The scene is visualized using X3D [5], an open standard language for visualizing 3D scenes in web-based systems. As stated in the Introduction section, the visualization of complex scenes is partly solved in desktop computers with the occlusion query extension of the graphics cards [23]. However when the scene must be displayed in a small computer connected to the Internet, the bottleneck is the data transmission over the network. The smaller files that are transmitted, the faster the interaction and visualization are. In a client-server architecture, the server side receives an interaction request in a given coordinate. Then, a file in X3D format with the visible scene from this point is generated and transmitted through the network to the client device, which finally visualizes the resulting 3D scene. In these cases a file containing non-visible buildings implies more transmission time for primitives that finally will not be visualized. An exact occlusion method allows an accurate visualization and a real-time interaction.

© 2013 NSP
Natural Sciences Publishing Cor.

Algorithm 7 $VivisilityMap(E, p, r_L, r_R)$

Input: The four polar diagrams of E: $\mathscr{PD}(E) = \{\mathscr{P}_{0-}(E)$, $\mathscr{P}_{0+}(E)$, $\mathscr{P}_{\pi+}(E)$, $\mathscr{P}_{\pi-}(E)\}$
-The rays $r_L = p + t\vec{d_L}$ and $r_R = p + t\vec{d_R}$ defining the visibility angle.
Output: The visible set V
Var: T: a tree-based data structure of angularly sorted rays
- R: set of tuplas containing rays and lists of crossed polar regions (r_i, lp_i), $R = \{(r_L, lp), (r_{L+1}, lp_{L+1}), ..., (r_{R-1}, lp_{R-1}), (r_R, lp_R)\}$

BEGIN
1. Initialize V, $V \leftarrow \emptyset$
2. **FOR EACH** $[r_l, r_r] \subset [r_L, r_R]$ **DO**
(a) Select the polar diagram $\mathscr{P}(E) \in \mathscr{PD}(E)$ according to Figure 4
(b) Initialize T and R, $T \leftarrow \emptyset$, $R \leftarrow \emptyset$
(c) Locate p in the polar region of object o, $p \in \mathscr{P}_{(E)}(o)$
(d) Insert r_l and r_r clockwise sorted in T, $T \leftarrow T + \{r_l, r_r\}$
(e) bool $adjacency \leftarrow$ false
(f) **REPEAT**
i. **WHILE** (T is not empty) **DO**
A. Get the first ray r of T, $T \leftarrow T - \{r\}$
B. RayShootingQuery2.5D($\mathscr{P}(E)$, $r(t) = p + t\vec{d}, hmax, o): lp, V_{r(t)}$ (*Shoot ray r using Algorithm 5*)
C. **IF** $\exists o_k$, $o_k \in V_{r(t)}$ **and** $o_k \notin V$ (*There are new visible objects*)
D. **THEN** FOREACH $o_k \in V_{r(t)}$ **and** $o_k \notin V$
-$V \leftarrow V + \{o_k\}$
-let be rt_l the left tangent ray to o_k
-IF rt_l is not out of range and is not within a pair of adjacent rays
-THEN insert rt_l clockwise sorted in T, $T \leftarrow T + \{rt_l\}$
-let be rt_r the right tangent ray to o_k
-IF rt_r is not out of range and is not within a pair of adjacent rays
-THEN insert rt_r clockwise sorted in T, $T \leftarrow T + \{rt_r\}$
E. Insert (r, lp) into R, $R \leftarrow R + \{(r, lp)\}$
ii. **END_WHILE**
iii. **IF** all consecutive pair of tuples $(r_i, l_i, o_i) - (r_{i+1}, l_{i+1}, o_{i+1})$ are adjacent (use Algorithm 6)
iv. **THEN** $adjacency \leftarrow$true
v. **ELSE** $T \leftarrow T + find_potential_VO((r_i, lp), (r_{i+1}, lp))$ (Algorithm 4)
(g) **UNTIL** $adjacency =$true
3. **END_FOREACH**
4. **RETURN** V
END

Table 5: Polar diagram computation times.

	Scene 1183			Scene 7000		
num. blocks	1183			12168		
num. triangles	7000			70000		
Polar diagram						
construction time (ms)	437			1940		
Viewpoints (A,B,C)	A	B	C	A	B	C
num. visible blocks	1	2	2	4	2	3
Visibility time (ms)	30	89	10	42	29	30

The experimental results have been implemented using a Windows computer system: Intel(R) 3.42 GHz CPU, 1GB RAM. Table ?? reflects the computation times associated with the construction of the polar diagram and the visibility map of different scenes with 1183 and 12168 2.5D blocks of buildings respectively. Both scenes, in particular the second one, have a heavy geometry to be managed in a web-based system. The plane tessellation is computed in less than 2 seconds in the larger scene. Anyway, this time corresponds to the pre-processing phase.

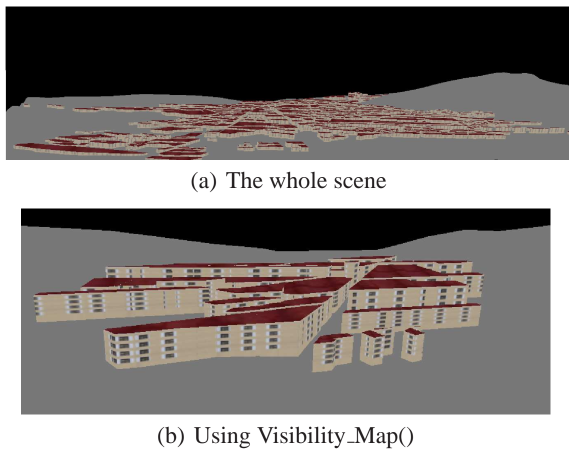
Once the four polar diagrams are computed we place three observers in each scene in the locations A, B and C. The best performance of the algorithm is found when the observer is close to the buildings and only several of them are visible. In the performance tests only few blocks of buildings and the time for detecting visibility ranges from 10 to 89 ms.

However, comparative tests based on computation times are then less interesting in web-based systems. The time for transmitting the information uses to be the bottleneck and it is highly dependent of the quality of the Internet connection. Table ?? shows the size of the files of the visible set of the first scene. The comparison of our method with any other that transmits the whole scene is unfeasible. Thus, the urban model has been inserted into a quadtree for determining the set of objects into the viewing angle.

The quadtree is a hierarchical data structure with an efficient location time, $O(\log_4 n)$. However, the number of objects placed into this angular sector may be much greater, as Table ?? shows. The first column shows several positioning coordinates of the observer. The method proposed in this paper points out that the set of visible objects in these positions is lower than twenty while the view-frustum contains more that one hundred of objects inside. Consequently, the size of the transmitted X3D files save up to 94% of space.

In Figure 13.a) the whole urban scene is depicted together with the relief of the area in which the city is located. In Figure 13.b) the scene only shows the set of visible buildings at a given time of the navigation. In Figure 14 it can be appreciated that from the viewpoint of a pedestrian, the resulting scene is the same when

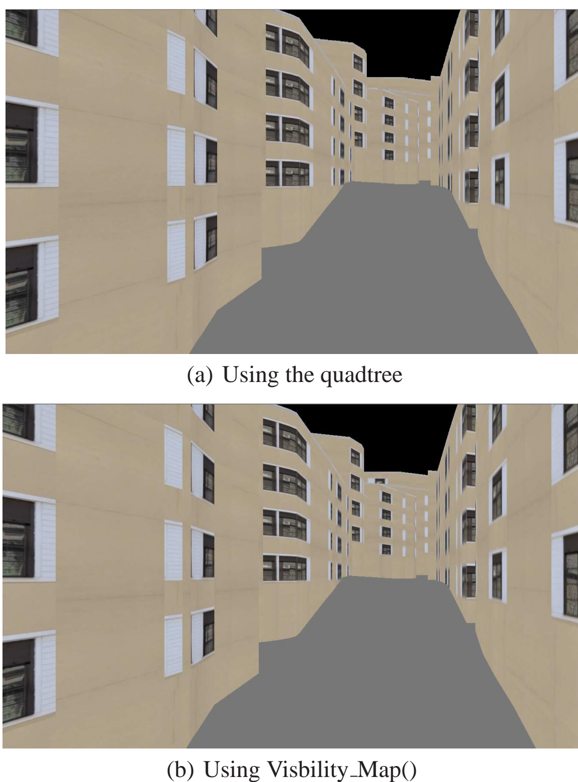
© 2013 NSP
Natural Sciences Publishing Cor.



(a) The whole scene

(b) Using Visibility_Map()

Fig. 13: An aerial perspective of the whole scene and using Visbility_Map



(a) Using the quadtree

(b) Using Visbility_Map()

Fig. 14: A pedestrian perspective of the city using Visbility_Map and using the quadtree

Table 6: Comparative test using polar diagrams and quadtrees.

Position	Blocks Quadt.	Blocks Polar D.	Quadtree (Kb)	Pol. D (Kb)	%space saved
0.57,0.09	173	18	1887	300	84,1
0.54,0.11	181	7	1826	57	96,9
0.62,0.10	124	15	1306	130	90,0
0.6,0.13	120	7	1248	138	88,9
0.53,0.13	149	3	1536	80	94,7

visualizing only the visible set but with a reduced number of blocks of buildings.

6 Conclusions and future work

In this work some algorithms for determining visibility have been developed both in $2D$ and in $3D$ scenes. In all cases the scene is decomposed using the polar diagram as pre-processing, which can be computed in optimal $O(n\log n)$ time. The location of the observer position into the scene is computed only once in logarithmic time. The data structure of this plane tessellation allows to manage topological relationships between adjacent regions for an efficient traversal process. The visibility solution for $2D$ scenes is straightforward extended to $2.5D$ with interesting applications to urban scenes. The terrain features can be considered in order to properly place the buildings in the urban environment.

The visibility map computation time is $O(k.n)$ for k visible objects in a scene of n polygonal or prism-shaped objects. This result is very interesting in densely occluded scenes such as urban environments. In these cases only a small number of rays must be shot to find k objects. In addition, these rays immediately take a high gradient, which allows to stop the traversal process requiring only a few steps of the algorithm. In these cases, the time computing is almost constant.

The advantage of obtaining the exact visibility set is specially appreciated in web-based systems in which the client device interacts with the portion of visible scene retrieved from the server side. The real-time navigation is only possible if the transmitted portion of scene by the network is greatly reduced.

It is also possible that the observer is located in such a position that almost the whole scene is visible. As future work our algorithm can be adapted to detect these cases and to apply different techniques for saving the transmission of geometry, such as the use of impostors or low-resolution models.

Acknowledgments

This work has been partially granted by the Conserjería de Innovación, Ciencia y Empresa of the Junta de Andalucía, under the research project `P07-TIC-02773`.

© 2013 NSP
Natural Sciences Publishing Cor.

References

[1] Tomas Akenine-Moller, Eric Haines, and Naty Hoffman. *Real-Time Rendering*. AK Peters, 2008.

[2] Sigal Ar, Bernard Chazelle, and Ayellet Tal. Self-customized bsp trees for collision detection. *Computational Geometry- Theory and Applications, Special Issue on Computational Geometry in Virtual Reality*, pages 23–29, 2000.

[3] Bahram Sadeghi Bigham, Ali Mohades, and Lidia Ortega. Dynamic polar diagram. *Inf. Process. Lett.*, 109(2):142–146, 2008.

[4] J. Bittner, Peter Wonka, and Michael Wimmer. Fast exact from-region visibility in urban scenes. In Kavita Bala and Philip Dutre, editors, *Rendering Techniques 2005 (Proceedings Eurographics Symposium on Rendering)*, pages 223–230. Eurographics, Eurographics Association, June 2005.

[5] Don Brutzman. *X3D: Extensible 3D Graphics for Web Authors*. Elsevier, San Diego, CA, 2007.

[6] D. Cohen-Or, Y. Chrysanthou, C.T. Silva, and F. Durand. A survey of visibility for walkthrough applications. *IEEE Transation and Computer Graphics*, 19(3):412–431, Jul-September 2003.

[7] Dieter W. Fellner and Steven Collins, editors. *Proceeding of the 13th International Conference on 3D Web Technology, Web3D 2008, Los Angeles, California, USA, August 9-10, 2008*. ACM, 2008.

[8] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[9] A. Fujimoto, Takayuki Tanaka, and K. Iwata. Tutorial: computer graphics; image synthesis. chapter ARTS: accelerated ray-tracing system, pages 148–159. Computer Science Press, Inc., New York, NY, USA, 1988.

[10] Marina L. Gavrilova, editor. *Generalized Voronoi Diagram: A Geometry-Based Approach to Computational Intelligence*, volume 158 of *Studies in Computational Intelligence*. Springer, 2008.

[11] A. S. Glassner. Tutorial: computer graphics; image synthesis. chapter Space subdivision for fast ray tracing, pages 160–167. Computer Science Press, Inc., New York, NY, USA, 1988.

[12] Dan Gordon and Shuhong Chen. Front-to-back display of bsp trees. *IEEE Comput. Graph. Appl.*, 11(5):79–85, September 1991.

[13] Clara I. Grima, Alberto Marquez, and Lidia Ortega. A new 2d tessellation for angle problems: The polar diagram. *Computational Geometry. Theory and Applications*, 34(2):58 – 74, 2006.

[14] H. Hey and W. Purgathofer. Occlusion culling methods. state of the art report. In *EUROGRAPHICS'01*, Manchester, 2001.

[15] M.R. Kaplan. The use of spatial coherence in ray tracing. *Techniques for Computer Graphics*, pages 173–193, 1987.

[16] Krzysztof S. Klimaszewski and Thomas W. Sederberg. Faster ray tracing using adaptive grids. *IEEE Comput. Graph. Appl.*, 17(1):42–51, January 1997.

[17] Marc J. van Kreveld, Jürg Nievergelt, Thomas Roos, and Peter Widmayer, editors. *Algorithmic Foundations of Geographic Information Systems, this book originated from the CISM Advanced School on the Algorithmic Foundations of Geographic Information Systems*. Springer-Verlag, London, UK, UK, 1997.

[18] Brian Mirtich. V-clip: fast and robust polyhedral collision detection. *ACM Trans. Graph.*, 17(3):177–208, July 1998.

[19] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley and Sons, 1992.

[20] Lidia Ortega and Francisco F. Feito. Collision detection using polar diagrams. *Computer & Graphics*, 29(5):726–737, 2005.

[21] Lidia M. Ortega, Antonio J. Rueda, and Francisco R. Feito. A solution to the path planning problem using angle preprocessing. *Robot. Auton. Syst.*, 58(1):27–36, January 2010.

[22] M. Pellegrini. Ray shooting and lines in space. *Handbook of Discrete and Computational Geometry*, pages 599–614, 1997.

[23] Fernando Randima. *GPU Gems: Programming Techniques, Tips, and Tricks for Real-time Graphics*. Addison-Wesley, 2004.

[24] John Spackman and Philip J. Willis. The smart navigation of a ray through an oct-tree. *Computers & Graphics*, 15(2):185–194, 1991.

L.M. Ortega received the BSc degree in Computer Science from the University of Granada, Spain, and the PhD degree from the University of Seville, Spain. She is professor at the Department of Computer Science at University of Jaén teaching at the High Polytechnics Institute. Her research is diversified into different areas such as Computational Geometry (tessellations, visibility), Computer Graphics (occlusion culling, collision detection path planning, based-layer decomposition), 3D GIS or Urban Systems..

M.D. Robles-Ortega received her Bachelor's degree in Computer Science and the PhD degree from the University of Jaén, Spain. Her research interests include Computer Graphics, Computational Geometry, GIS systems, Museum Design and Urban Environments. She has published research articles in reputed international journals of computer and engineering sciences.

© 2013 NSP
Natural Sciences Publishing Cor.