Visibility Map determination using Angle preprocessing

L. Ortega, A.J. Rueda and F. Feito[*]

Abstract

Visibility determination becomes especially significant in Computer Graphics for walkthrough applications. The aim is to determine those visible objects of the scene from the viewer position. Even when this problem is nowadays efficiently solved by the graphics hardware, when a real-time response is required, some additional CPU processing in the scene geometry may be necessary to deal with large scenes or with observers moving into the scene.

The present work introduces a new technique based on angle preprocessing with *polar diagrams.* The result is displaying $2.5D$ scenes in a more efficient way than using acceleration graphics hardware.

1 Introduction

The increasing capabilities of the graphics hardware allows to speed up image generation, obtaining a great realism sensation. However, scene sizes can be increased to become too high to provide interactive frame-rates. A previous process to the rendering phase should be able to discard as much non-visible objects as possible, in order to release the hardware of displaying scene primitives that are invisible from a certain viewer position. These techniques are called *Visibility culling* and some of them have been already incorporated in the hardware. The aim is reducing as much as possible the resulting Potentially Visible Set (PVS) for a high performance in the rendering, obtaining the *visibility map* when the PVS becomes minimum. Occlusion culling methods, summarized in [1, 2, 7], try to reduce the PVS as possible, but not forgetting any visible primitive. The strategies including this characteristic are called *conservative methods.*

Normally, the type of scene if determinant in order to select a proper occlusion culling method. Urban scenes, as well as architectural environments, are considered a special case of densely occluded environments, studied by *cell-portal culling* techniques [8]. If cities consist of a set of 2.5D objects representing buildings instead of 3D models [3, 11], the scene geometry becomes more straightforward to process, what could make possible a exact visibility approach in real time, even if virtual observers are moving into the scene. The additional CPU time to compute the visibility set in the $2.5D$ scene is compensated in the rendering phase.

In this work we propose a new visibility culling approach based on *polar diagrams*, a plane tessellation, that used as preprocessing, solves efficiently some angle-based problems, as Visibility Determination or Convex Hull of points and objects [4, 5], Path Plannig [6] and Collision Detection [10].

Section 2 defines the polar diagram and its angular characteristics. Section 3 studies visibility with polar diagrams as a previous step to find the visible set of objects from any point in the scene (Section 4), and finally Section 5 shows the experimental results when this algorithm is used to display urban scenes.

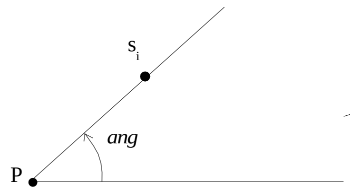
2 The polar diagram

The polar diagram of the scene E, consisting of n two-dimensional objects, $E = \{o_0, o_1, ..., o_{n-1}\}$, denoted as $\mathcal{P}(E)$, is a plane partition in *polar regions.* Each generator object o_i creates a polar region $\mathcal{P}_E(o_i)$ representing the locus of points with common angular characteristics in a starting direction. Any point in the plane can only belong to a polar region, what 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, $p \in \mathcal{P}_E(o_i)$, we know that o_i is the first object found after performing an angular scanning from the horizontal line crossing p in counterclockwise direction [9].

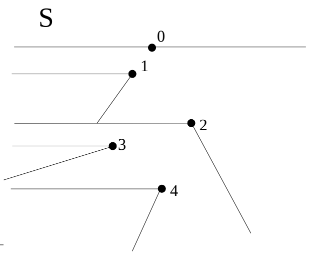
The *polar angle* of point p with respect to o_i, denoted as $ang_{o_i}(p)$, is the angle formed by the positive horizontal line of p and the straight line linking p and o, as shown in Figure 1.a). The polar angle must be lower than π, involving that p must be under the horizontal line defined by s_i. The locus of points with smaller positive polar angle with respect to $o_i \in E$ is the polar region of o_i. Thus, $\mathcal{P}_E(o_i) = \{(x, y) \in E^2 \mid ang_{o_i}(x, y) < ang_{o_j}(x, y), \forall j \neq i\}$. The same criterion is valid for points or any other geometric object. Figure 1.b) depicts the polar diagram of the cloud of points S

The polar diagram can be computed efficiently using the Divide and Conquer or the Incremental methods, both working in $\Theta(n \log n)$. The strength of using this tessellation as preprocessing is avoiding any angular sweep by locating a point into a polar region in logarithmic time [4, 5].

[*]Department of Computer Science, University of Jaén, Spain {lidia|ajrueda|ffeito}@ujaen.es



(a) polar angle of p with respect to s_1



(b) polar diagram of S

Figure 1: Example of polar angle and polar diagram.

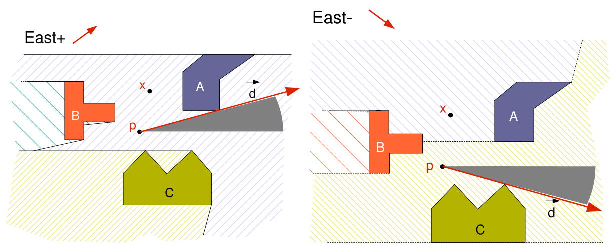


Figure 2: $p \in \mathcal{P}_E(A)$ in the *East*+ polar diagram and $p \in \mathcal{P}_E(C)$ in the *East*− polar diagram.

3 Visibility resolution

Polar diagrams can solve Visibility problems due to its capability of determining the nearest angular neighbours. Observe Figure 2 and suppose that the maximum visibility angle from point p in East direction is required. Instead of solving the problem in linear time by performing two angular sweeps, object A is found because point p is located in its polar region in $O(n \log n)$ time, as depicted in Figure 2.a). However, a new polar angle criterion should be necessary to find object C, and consequently a new plane tessellation. With the positive polar angle criterion (counterclockwise) the *East*+ polar diagram is constructed, and using the negative one (clockwise), the *East*−, as Figure 2.b) shows. The maximum visibility angle is obtained throwing tangent lines towards A in *East*+, and towards C in the *East*− polar diagram.

Summarizing, given the observer position p looking at the East direction, and the pair of polar diagrams *East*+ and *East*−, the visibility problem solution can be dealt with a simple result:

- if the point lies in regions associated to different objects in both polar diagrams, as point p in the figure, it always means that there is an open visibility angle in *East*+ direction.

- when a point lies in regions belonging to the same object, the visibility angle is null.

if $0 \leq \overrightarrow{d} \leq \pi/2$ use p.d. East+	
if $\pi/2 \leq \overrightarrow{d} \leq \pi$ use p.d. West-	
if $\pi \leq \overrightarrow{d} \leq 3\pi/2$ use p.d. West+	
if $3\pi/2 \leq \overrightarrow{d} \leq 0$ use p.d. East-	

Figure 3: The Polar diagram election depends on the direction.

Point x belongs only to the polar regions of object A, what implies that the visibility angle is null. Anyway, this information can be enormously important because we know exactly the first object obstructing a trajectory in the specified direction, what becomes useful in collision detection problems.

To manage visibility problems in any other non-orthogonal direction, a fixed and specific set of polar diagrams can be constructed, each of them covering an angular spectrum, a quadrant of the coordinate system, for instance. The *East*+ polar diagram can avoid angular sweeps in the interval $[0, \pi/2]$, what means that for all rectilinear movements in this angular interval, the use of the *East* polar diagram is enough to solve visibility problems. If the movement direction is in the range $[\pi/2, \pi]$, the visibility information can be given by the *West*- polar diagram, that is, starting from π and sweeping clockwise. Figure 3 shows a table with this correspondence. In the worst case, the first visibility object in a given direction is solved using four polar diagrams.

Actually, visibility problems need to be solved in angular intervals, the equivalent to the observer vision angle. The previous problem is more a collision detection resolution as described in [10]: a point object is thrown from the position p with direction $\overrightarrow{d}$, describing the ray $r(t) = p + t\,\overrightarrow{d}$. The result is the first object intersecting its trajectory. However, this method is the key to compute the visibility map as we describe next.

4 Visibility culling in $2D$ scenes

The visibility culling resolution becomes an exact visibility approach using polar diagrams in two-dimensional scenes because it allows to compute the exact set of visible objects, even the set of visible edges, what becomes specially interesting in scenes whose polygons contain a large number of vertices. This new approach is conservative: obtains an angular-sorted list of primitives and is independent of the scene size, that is, the processing time depends only on the visible set size.

Let be E a two-dimensional scene consisting on a set of polygonal objects, $E = \{o_1, o_2, \ldots, o_n\}$, and an observer located in position p looking at the scene in an angular interval $[\vec{r_L}, \vec{r_R}]$, being $\vec{r_L}$ a ray defin-

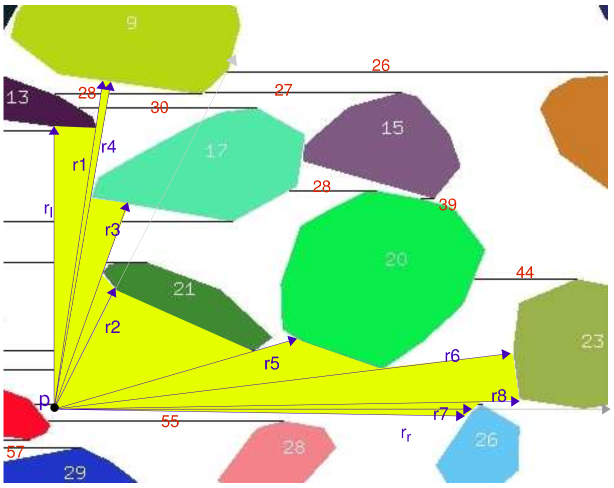


Figure 4: Example of visible set from p.

ing the left portion of the angular sector, and $\vec{r_R}$ the one defining the right portion, both rays starting from point p as observed in Figure 4.

- If $\vec{r_L}$ and $\vec{r_R}$ are located in different quadrants, the angular interval $[\vec{r_L}, \vec{r_R}]$ is divided into the resulting sub-intervals from the intersection with the coordinate axes.
- Each of these sub-intervals, $[\vec{r_l}, \vec{r_r}] \subseteq [\vec{r_L}, \vec{r_R}]$ is located in only a polar diagram. For all of them: (1) locate p in a polar region using the appropiate polar diagram and (2) obtain a sorted list of rays R_{lr}, between $\vec{r_l}$ and $\vec{r_r}$, $R_{lr} = \{\vec{r_l}, \vec{r_1}, \vec{r_2}, ..., \vec{r_r}\}$, being $\vec{r_j}$, $j \in [i, 1, 2, ..., d]$ a ray starting from point p.

For every r_j a collision detection is performed, determining not only the intersected object (or primitive), o_i, $i \in [0..N-1]$, but also a set of crossed polar regions, or what becomes similar, the set of crossed polar edges. Let be l_j the sorted set of crossed polar edges by r_j until a collision detection is detected, or until the ray left the scene, $l_j = \{e_k, e_{k+1}, ..., e_m\}$, $k \geq m$. The final set of rays R_{lr}, represents the visibility map because each pair $[r_j, r_{j+1}]$, $j \in [l, 1, 2, ..., r-1]$, symbolizes an angular sector (a triangle) from the viewer position p towards an only polygon o_i, as depicted in Figure 4.

The method starts from the rays $\vec{r_l}$ and $\vec{r_r}$, the ones defining the vision angle. The rest of rays of R_{lr} are obtained throwing tangent rays towards the involved objects of the collision method described in [10]. Whenever a ray $\vec{r_j}$ reaches a new object o_i, the following step consists of throwing the ray $\vec{r_{j+1}}$ as right tangent (or left tangent depending on the running sequence of the algorithm) from p to o_i. The final goal is joining $\vec{r_l}$ and $\vec{r_r}$ with a fan of rays. The ray $\vec{r_j}$ can:

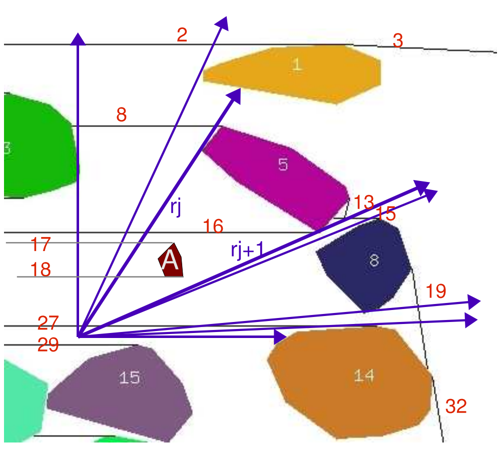


Figure 5: The method is able to find object A.

- reach the right (or left) tangent vertex of o_i and then collides with some other object o_m (in the example $\vec{r_1}$ is tangent to o_{13} and intersects with o_9).
- not reach the right (or left) tangent vertex of o_i because it collides previously with some other o_k (in the example the ray $\vec{r_2}$, the right tangent line to o_9, collides before with o_{21}). These cases are easily detected checking the length of the lists of rays; if l_i is longer than l_j, as in this example, the search is diverged in two angular intervals, the first between $\vec{r_i}$ and $\vec{r_j}$, and the second between $\vec{r_j}$ and $\vec{r_d}$, that is, $[\vec{r_l}, ..., \vec{r_j}, ..., \vec{r_r}]$. The process starts from $\vec{r_j}$ left to reach $\vec{r_l}$, and right to reach $\vec{r_r}$ recursively, closing the angular sequence.

Conclusions after applying polar diagrams in the described visibility culling technique are the following: $O(2k)$ rays thrown for k visible objects, being k independent of the scene size n. In the example ten rays are necessary to locate seven visible objects. The method is conservative because it is able to find all visible objects from p without forgetting any of them. In fact, it is an exact visibility culling method for $2D$ scenes because the exact set of visible edges from p is found.

The example of Figure 5 clearly shows that the method is conservative. Suppose that object A is located in such a position that the described algorithm throws tangent rays, $\vec{r_j}$ and $\vec{r_{j+1}}$ towards tangent positions of object o_5, but object A is not reached at all. However A can be detected by checking the lists $l_j = \{e_{27}, e_{18}, e_{17}, e_{16}, e_8, e_2\}$ and $l_{j+1} = \{e_{27}, e_{16}, e_8, e_2\}$. Both begin and end with the same sequence but edges e_{18} and e_{17} does not appear in list l_{j+1}, what means that there is another object just in the middle of the angular interval $[\vec{r_j}, \vec{r_{j+1}}]$.

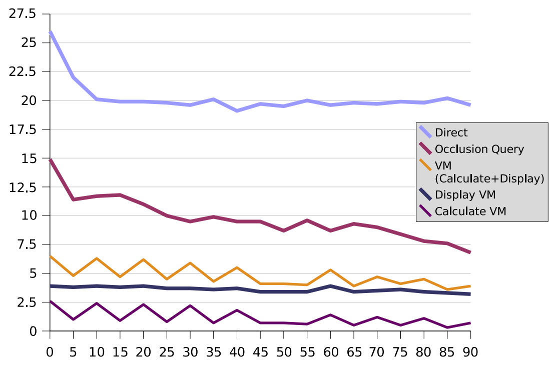


Figure 6: Experimental results.

5 Experimental results

Normally the efficiently of occlusion culling methods are highly dependent of the environment characteristics. However, the construction of this plane partition is not altered by the scene density, or by the convexity or non-convexity of occluders, only the number of vertices is relevant. In any case polar diagrams are calculated only once in the preprocessing phase. The location and collision processes are determined by the number of objects N. In the calculation of the visibility map, the more visible primitives, the more collision detections are necessary. If occluders are large objects, their capacity to hide some others is greater, and the number of rays to throw probably lower. Furthermore, a collision detection process is dependent on the number of crossed polar edges, and consequently on the number of scene objects.

The experimentation has been carried out in a Pentium IV running at 2.4GHz with a plane city of $N = 600$ buildings with non-convex floor, each of them representing a ground plant of a building or a block of buildings. The total number of vertices in the scene is $n = 45.000$, a number only relevant in the polar diagram construction process. Each of the four polar diagrams are calculated in $100ms$. The rest of algorithms, location or collision detection, run in proportional times to $2N = 12000$ polar edges.

Figure 6 shows the requiring time to display the same view of the city while an observer moves along of part of its outskirts. The required time to display the scene at every step is represented in edge Y, while the edge X represents each of this steps. For simplicity, we suppose a vision angle of 90, centered in 45.

The chart is eloquent at all: the top most line represents the time to display the scene directly (*Direct*), without using any CPU or hardware improvement. When hardware acceleration is activated the visualization process is clearly improved (*Occlusion Query* [1]). However using the visibility map and displaying only those visible objects, the time behaviour is still better. The time required for performing the visibility map is appreciably lower that any of the required for visualization, what proves the advantage of using a CPU preprocessing. During the walkthrow process, sometimes the walker sees only a reduced set of building and in other occasions can see the row of buildinds of an avenue. That is the reason of the zigzag observed in the line defining the visibility map (*Visibility Map*). The line labeled as *MV+Display* adds the time required for the visibility map calculation and the one for displaying the resulting visible set.

References

[1] T. Akenine-Möller and E. Haines. *Real-Time Rendering*. A. K. Peters, 2002.

[2] 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.

[3] Laura Downs, Tomas Möller, and Carlo H. Séquin. Occlusion horizons for driving through urban scenery. In *Symposium on Interactive 3D Graphics*, pages 121–124, 2001.

[4] C. I. Grima, A. Máquez, and L. Ortega. A locus approach to angle problems in computational geometry. In *Proc. of 14th European Workshop in Computational Geometry*, Barcelona, 1998.

[5] C. I. Grima, A. Máquez, and L. Ortega. Polar diagrams of geometric objects. In *Proc. of 14th European Workshop in Computational Geometry*, pages 149–151, Sophia-Antipolis, 1999.

[6] C.I. Grima, A. Márquez, and L. Ortega. Motion planning and visibility problems using the polar diagram. In *EUROGRAPHICS'03 Short Presentations*, pages 13–19, 2003.

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

[8] D.P. Luebke and C. Georges. Portals and mirrors: Simple, fast evaluation of potencially visible sets. In *Proceedings 1995 Symposium on Interactive 3D Graphics*, 1995.

[9] L. Ortega. *El Diagrama Polar, Ph Thesis*. University of Seville (Spain), 2002.

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

[11] P. Wonka, M. Wimmer, and D. Schmalstieg. Occluder shadows for fast walkthroughs of urban environments. In *Proceedings of EUROGRAPHICS*, pages 51–60, 1999.