Approximating the Visible Region of a Point on a Terrain

Boaz Ben-Moshe* Paz Carmi† Matthew J. Katz‡

Abstract

Given a terrain and a point p on or above it, we wish to compute the region R_p that is visible from p. We present a generic radar-like algorithm for computing an approximation of R_p. The algorithm *interpolates* the visible region between two consecutive rays (emanating from p) whenever the rays are *close enough*; that is, whenever the difference between the sets of visible segments along the cross sections in the directions specified by the rays is below some threshold. Thus the density of the sampling by rays is sensitive to the shape of the visible region. We suggest a specific way to measure the resemblance (difference) and to interpolate the visible region between two consecutive rays. We also present an alternative algorithm, which uses circles of increasing radii centered at p instead of rays emanating from p. Both algorithms compute a representation of the (approximated) visible region that is especially suitable for is-visible-from-p queries, i.e., given a query point q on the terrain determine whether q is visible from p. Finally, we report on the experiments that we performed with these algorithms and with their corresponding fixed versions, using a natural error measure. Our main conclusion is that the radar-like algorithm is significantly better than the others.

Keywords: algorithms, triangulated irregular networks, visibility

*Department of Computer Science, College of Judea & Samaria, Ariel, 44837, `benmo@yosh.ac.il`.

†Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel, `carmip@cs.bgu.ac.il`.

‡Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel, `matya@cs.bgu.ac.il`.

1 Introduction

Let T be a triangulation representing a terrain (i.e., there is a height (z-coordinate) associated with each triangle vertex, and any vertical line intersects T at a single point (if at all)). T is sometimes called a triangulated irregular network (or TIN for short). We are interested in the following basic problem. Given a point p on (or above) T, compute the region R_p of T that is visible from p. A point q on T is visible from p if and only if the line segment $\overline{pq}$ lies above T (in the weak sense). Thus R_p consists of all points on T that are visible from p.

The problem of computing the visible region of a point on a terrain arises as a subproblem in numerous GIS-based applications. For example, the coverage area of an antenna for which line of sight is required may be approximated by clipping the region that is visible from the tip of the antenna with an appropriate disk centered at the antenna. In [16], Sharir considered the related shortest watchtower problem: Find the shortest vertical segment that has to be erected from the surface of the terrain T, so that its top endpoint can see the entire terrain. He presented an $O(n \log^2 n)$-time algorithm for this problem, where n is the number of triangular faces of T. An improved $O(n \log n)$-time algorithm was subsequently presented by Zhu [19]. See, e.g., [3, 5, 9, 10, 12, 17] for more examples, such as guarding and siting of air defense missile batteries.

These and other applications motivate the study of visibility problems on terrains; see, e.g., [4, 5, 6, 7, 9, 10, 14, 15]. De Floriani and Magillo [6] present two solutions to the *discrete* version of our problem, i.e., compute all vertices of T that are visible from p. Their more sophisticated solution is based on a data structure supporting ray shooting queries from p, due to Cole and Sharir [5], and its running time is $O(n \log^2 n)$. Katz et al. [14] present an output-sensitive algorithm for computing the visibility map of T from any fixed point; the running time of their algorithm is $O((n\alpha(n) + k) \log n)$, where $\alpha(n)$ is the inverse of Ackermann's function and k is the combinatorial complexity of the actual visibility map.

Since the combinatorial complexity of R_p might be $\Omega(n^2)$ [5, 8], it is desirable to also have fast approximation algorithms, i.e., algorithms that compute an approximation of R_p. Such algorithms may have sublinear running time in practice and may be simpler to implement. Moreover, a good approximation of the visible region is often sufficient, especially when the triangulation itself is only a rough approximation of the underlying terrain. Note that in this paper we are assuming that the terrain representation (i.e., the triangulation T) is fixed and cannot be modified. Simplifying the triangulation can of course result in a significant decrease in the actual running time of any algorithm for computing the visible region. This approach was studied in a previous paper [1]. See, e.g., [11] for more information on terrain simplification.

We present a generic radar-like algorithm for computing an approximation of R_p. The algorithm computes the visible segments along two rays ρ_1, ρ_2 emanating from p where the angle between the rays is not too big. (I.e., each of the rays specifies a direction, i.e., azimuth, and the algorithm computes the (projections of the) visible portions of the cross section of T in this direction.) It then has to decide whether the two sets of visible segments (one per ray) are *close enough* so that it can *interpolate* the visible region of p within the wedge defined by ρ_1 and ρ_2, or whether an intermediate ray is needed. In the latter case the algorithm will now consider the smaller wedge defined by ρ_1 and the intermediate ray. Thus a nice property of the algorithm is that the density of the sample rays varies and depends on the shape of R_p.

In order to use this generic algorithm one must provide (i) a measure of resemblance for two sets of visible segments, where each set consists of the visible segments along some ray from p, and (ii) an algorithm to interpolate the visible region between two rays whose corresponding sets were found similar enough. In Section 2 we describe in more detail the generic algorithm and provide the missing ingredients.

The running time of the radar-like algorithm is proportional to the sum of the sizes of the cross-sections that are computed. For a "typical" terrain, the algorithm only computes a constant number c of cross-sections and the expected size of a cross-section is $O(\sqrt{n})$. Thus the expected running time of the radar-like algorithm on a typical terrain is only $O(c\sqrt{n})$.

In Section 3 we present several other algorithms for computing the visible region R_p. The first algorithm computes R_p exactly. Since we need such an algorithm for the experimental evaluation of our approximate algorithms, we decided to devise one that is based on the general structure of the radar-like algorithm. Our exact algorithm is rather simple and is based on known results; nevertheless it seems useful. Specifically, the algorithm repeatedly computes the portion of R_p within a slice, defined by a pair of rays passing through consecutive vertices of the terrain (so that the slice does not contain a vertex of T in its interior). This computation can be done efficiently as is shown in [7].

The second algorithm (called the *expanding circular-horizon* algorithm or ECH for short) is in some sense orthogonal to the radar-like algorithm; it uses circles of increasing radii centered at the view point p instead of rays emanating from p. It is influenced by the exact algorithm described by De Floriani and Magillo [7]. The algorithm approximates the visible region R_p by maintaining the (approximate) viewing angles corresponding to a set of sample points on the expanding circular front (see Section 3). This allows us to partition the current front into maximal visible and invisible arcs. We now examine the sets of visible arcs on the current and previous fronts. If they are *close enough*, then the portion of R_p within the annulus defined by the two circles is approximated. Otherwise, we compute the visible arcs on a circle of intermediate radius and repeat.

Both the radar-like algorithm and the expanding circular-horizon algorithm have corresponding *fixed* versions, that play an important role in the experiments that were performed, see below. In the fixed version of the radar-like algorithm, the angle between two consecutive rays is fixed and we approximate the portion of R_p in the sector defined by the rays in any case, even if they are not *close enough*. In the fixed version of the expanding circular-horizon algorithm, the increase in radius between two consecutive circles is fixed and again we approximate the portion of R_p in the annulus defined by the circles in any case.

In Section 4 we suggest a natural way to measure the error in an approximation R'_p of R_p produced by one of our algorithms. Let l denote the *range of sight* that is in use. If the disk of radius l centered at p does not contain the underlying terrain, then we only consider the portion of the terrain that is contained in this disk when computing R_p and R'_p. The error associated with R'_p is the area of the **XOR** of R'_p and R_p, divided by the area of the disk of radius l centered at p. Using this error measure (and the exact algorithm), we performed a collection of experiments (described in Section 4) with the radar-like and expanding circular-horizon algorithms and their corresponding fixed versions. Our main conclusions from these experiments are that (i) the sensitive versions are significantly better than their corresponding fixed versions (when the total number of slices / annuli composing the final approximation is the same in both versions), and (ii) the radar-like algorithm is significantly better than the expanding circular-horizon algorithm. In Section 4 we offer some explanations to these findings.

2 The Radar-Like Algorithm

In this section we first present our radar-like generic algorithm. Next we describe the measure of resemblance and the interpolation algorithm that we devised, and that are needed in order to transform the generic algorithm into an actual algorithm.

The generic algorithm is presented in the frame below. The basic operation that is used is the cross-section operation, denoted *cross-section*(T, p, θ), which computes the visible segments along the ray emanating from p and forming an angle θ with the positive x-axis. More precisely, *cross-section*(T, p, θ) computes the (projections of the) visible portions of the cross section of T in the direction (i.e., azimuth) specified by this ray.

Roughly speaking, the generic algorithm sweeps the terrain T counterclockwise with a ray ρ emanating from p, performing the cross-section operation whenever the pattern of visible segments on ρ is about to change significantly with respect to the pattern that was found by the previous call to cross-section. The algorithm then interpolates, for each pair of consecutive patterns, the visible region of p within the wedge defined by the corresponding locations of ρ.

Given a triangulation T representing a terrain (i.e., with heights associated with the triangle vertices), and a view point p on or above T:

$\theta \leftarrow 0$.

$\alpha \leftarrow$ some constant angle, say, $\pi/45$.

$S_1 \leftarrow$ *cross-section*(T, p, θ) .

$S_2 \leftarrow$ *cross-section*$(T, p, \theta + \alpha)$.

while $(\theta \leq 360)$

    if (S_1 is **close-enough** to S_2)

        interpolate(S_1, S_2);

        $\theta \leftarrow S_2.angle$;

        $S_1 \leftarrow S_2$;

        $S_2 \leftarrow$ *cross-section*$(T, p, \min(\theta + \alpha, 360))$;

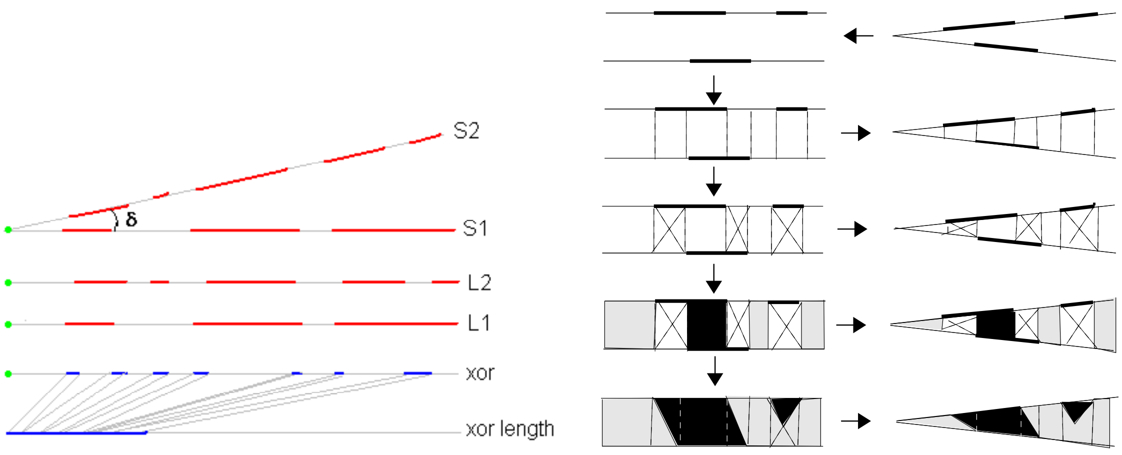
    else

        $\mu \leftarrow (S_1.angle + S_2.angle)/2$;

        $S_2 \leftarrow$ *cross-section*(T, p, μ);

In order to obtain an actual algorithm we must provide precise definitions of **close-enough** and **interpolate**.

Close-enough: A threshold function that checks whether two patterns S_1, S_2 are similar, where each of the patterns corresponds to the set of visible segments on some ray from p. There are of course many ways to define **close-enough**. We chose the following definition.



(a) (b)

Figure 1: Grey marks visible and black marks invisible. (a) The **close-enough** threshold function: δ times the relative length of the **XOR** of S_1 and S_2. (b) The **interpolate** function.

In practice, the rotating ray is actually a rotating segment of an appropriate length. Let l denote this length. We refer to l as the *range of sight*. Now rotate the ray containing S_2 clockwise until it coincides with the ray containing S_1. See Figure 1 (a). Next compute the length of the **XOR** of S_1 and S_2, that is, the total length covered by only one of the sets S_1, S_2. This length is then divided by l. Denote by v the value that was computed, and let δ be the angle between S_1 and S_2. If $\delta \cdot v \leq C$, where C is some constant, then return TRUE else return FALSE. The role of δ in the above formula is to force **close-enough** to return TRUE when the angle between the rays is small, even if the patterns that are being compared still differ significantly.

Interpolate: Given two patterns S_1, S_2 which are **close-enough**, we need to compute an approximation of the portion of the visible region of p that is contained in the corresponding wedge. We do this as follows. Consider Figure 1 (b). For each 'event point' (i.e., start or end point of a visible segment) on one of the two rays, draw a segment that connects it with the corresponding point on the other ray. For each rectangle that is obtained color it as follows, where grey means visible and black means invisible. If the horizontal edges of a rectangle are either both visible from p or both invisible from p, then, if both are visible, color it grey, else color it black. If, however, one of the horizontal edges is visible and the other is invisible, divide the rectangle into four pieces by drawing the two diagonals. The color of the upper and lower pieces is determined by the color of the upper and lower edges, respectively, and the color of the left and right pieces is determined by the color of the rectangles on the left and on the right, respectively. Assuming there are no two event points such that one of them is exactly above the other, the coloring procedure is well defined. That is, the odd numbered rectangles will be colored with a single color, and the even numbered rectangles will be divided.

Remark. One can think of alternative definitions for **close-enough** and **interpolate**. However, it seems reasonable to require the following two properties: (i) A small change in the set of visible segments along a ray should only cause small changes in the **close-enough** measure and in the visible region computed by **interpolate** (within the appropriate wedge), and (ii) If there are no "surprises" between two close enough rays, then the visible region computed by **interpolate** within the wedge should be very similar to the *real* visible region. In addition, the definitions should remain simple and easy to implement.

Both the running time of the radar-like algorithm and the combinatorial complexity of the computed region R'_p are proportional to the sum of the sizes of the cross-sections that are computed during the execution of the algorithm. For a "typical" terrain, the algorithm only issues a constant number c of calls to *cross-section*(T, p, θ), and the expected size of a cross-section is $O(\sqrt{n})$. Thus, for a typical terrain, the expected running time of the radar-like algorithm, as well as the expected combinatorial complexity of R'_p, is only $O(c\sqrt{n})$. Note that if the underlying terrain was obtained from a regular grid sampling, then the size of any cross-section is $O(\sqrt{n})$. In the worst case, though, the size of a cross-section is $O(n)$, implying a bound of $O(cn)$ on both the running time and combinatorial complexity. Moreover, the combinatorial complexity of the (approximated) visible region

R'_p can be significantly smaller than $O(c\sqrt{n})$, since the number of visible segments along a ray is usually much smaller than the size of the corresponding cross-section.

Is-visible-from-p queries. The representation of the (approximated) visible region R'_p that is computed by the radar-like algorithm is especially suitable for queries of the form: Given a query point q on T, determine whether q is visible from p, or, more precisely, determine whether (the projection of) q lies in R'_P. We first verify that q is within the range of sight l, i.e., that q lies within the disk of radius l centered at p. Next we determine whether q lies in R'_p, in logarithmic time, by two binary searches. The first search locates the sector of the disk in which q lies, and the second locates the 'rectangle' within the sector in which q lies. Finally, it remains to check in which of the at most four triangles corresponding to this rectangle q lies.

As noted above, the expected size of the data structure is $O(c\sqrt{n})$ or even less. Moreover, in comparison with point location in general triangulations (see, e.g., [2]), it is, in practice, much simpler to preprocess R'_p for visibility queries and less extra storage space is required.

3 Other Algorithms

In this section we present several other algorithms for computing the visible region. The first algorithm computes the visible region exactly; its general structure is similar to that of the radar-like algorithm. The second algorithm (called the *expanding circular-horizon* algorithm or ECH for short) is influenced by the exact algorithm of De Floriani and Magillo [7]. It computes an approximation of the visible region using circles of increasing radii (instead of rays) and similar definitions of *close-enough* and *interpolate*. Towards the end of this section we mention the *fixed* versions of the radar-like and expanding circular-horizon algorithms.

The algorithms presented in this section are part of our testing environment for the radar-like algorithm. However, we believe that the exact algorithm and the expanding circular-horizon algorithm are of independent interest.

3.1 The exact algorithm.

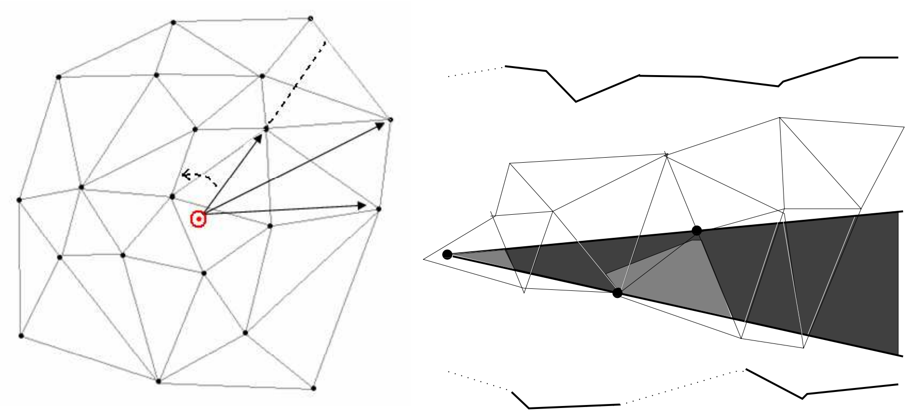


Figure 2: Left: the exact algorithm draws a ray through each vertex of T; Right: a slice that is defined by two consecutive rays, the corresponding cross sections, and the exact portion of R_p within the slice.

Since we need an exact algorithm (i.e., an algorithm that computes the visible region R_p exactly) for the experimental evaluation of our approximate algorithms, we decided to devise one that is based on the general structure of the radar-like algorithm, instead of using one of the known algorithms. Our exact algorithm is rather simple. It repeatedly computes the (exact) portion of R_p within a slice that is defined by a pair of rays passing through vertices of the terrain T, and that does not contain a vertex of T in its interior; see Figure 2. This can be done in time $O(m \log m)$, where m is the number of edges of the terrain that cross the slice, as is shown in [7].

Remark. The radar-like algorithm can be modified accordingly, so that whenever the slice under consideration does not contain a vertex of T in its interior, the portion of R_p within the slice is computed exactly, rather than calling **close-enough** (possibly more than once) and then **interpolate**.

3.2 The expanding circular-horizon algorithm.

This algorithm is in some sense orthogonal to the *radar-like* algorithm; it uses circles of increasing radii centered at the view point p instead of rays emanating from p.

For a point q on T, let ψ_q be the minimum viewing angle (with respect to the horizontal plane through p) at which a vertical pole of infinite height based at q can be seen from p; ψ_q is the *viewing angle* corresponding to q. Our algorithm approximates the visible region R_p, by maintaining the (approximate) viewing angles corresponding to the points on the expanding *circular horizon.* More precisely, the algorithm only considers the points on the circular horizon at direction (i.e., azimuth) $\alpha, 2\alpha, 3\alpha, \ldots$ with respect to p, where α is a parameter of the algorithm. Initially, the circular horizon is the point p itself, and the corresponding viewing angle is $-\pi/2$.

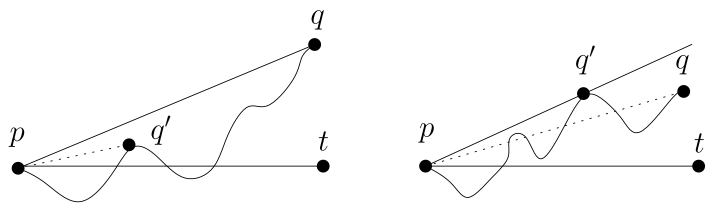


Figure 3: Left: $\psi_q = \phi_q = \angle_{qpt}$ and q is visible; Right: $\psi_q = \psi_{q'} > \phi_q$ and q is invisible.

The (approximate) viewing angles for the current circular horizon are computed from those of the previous circular horizon (by *resolve-viewing-angles*) as follows (see Figure 3). Let q be a point on the current horizon at direction $i\alpha$ with respect to p, and let q' be the point on the previous horizon at the same direction. Let ϕ_q be the angle between pq and the horizontal plane through p, then $\psi_q \leftarrow max\{\psi_{q'}, \phi_q\}$, and q is said to be *visible* from p if and only if $\psi_q = \phi_q$.

After applying *resolve-viewing-angles* to the current horizon C, we partition C into maximal visible and invisible arcs as follows. An arc of C between two consecutive sample points (i.e., points at direction $i\alpha$ and $(i+1)\alpha$ with respect to p) is called a *primitive* arc. We first consider each of the primitive arcs a. If both endpoints of a are visible (resp., invisible), then we assume all points in a are visible (resp., invisible). If however one of the endpoints q_i is visible and the other q_{i+1} is invisible, we assume all points in the half of a adjacent to q_i are visible and all points in the half adjacent to q_{i+1} are invisible. We now can partition C into maximal visible and invisible arcs.

We consider the current and previous horizons C_2 (of radius r_2) and C_1 of radius (r_1), respectively. As in the radar-like algorithm, we must decide whether the two sets of visible arcs are **close-enough**. If yes, we call **interpolate** to approximate the portion of R_p that is contained in the annulus defined by C_1 and C_2. If no, we generate the intermediate horizon C of radius $(r_1 + r_2)/2$, and repeat for the pair C_1, C. We use similar definitions of **close-enough** and **interpolate** as in the radar-like algorithm.

Given a triangulation T representing a terrain (i.e., with heights associated with the triangle vertices), and a view point p on or above T:

$\alpha \leftarrow$ some constant angle, say $\pi/180$.

$d \leftarrow$ some constant distance, say, 10 meters.

$r_1 \leftarrow r_{min}$.

$C_1 \leftarrow$ determine the viewing angles corresponding

to the $2\pi/\alpha$ sample points on the circle of radius r_1.

$C_2 \leftarrow$ *resolve-viewing-angles*$(T, p, C_1, r_1 + d)$.

while $(r_1 \leq r_{max})$

  if (C_1 is **close-enough** to C_2)

    interpolate(C_1, C_2);

    $r_1 \leftarrow C_2.radius$;

    $C_1 \leftarrow C_2$;

    $r \leftarrow \min(r_1 + d, r_{max})$;

    $C_2 \leftarrow$ *resolve-viewing-angles*(T, p, C_1, r);

  else

    $r \leftarrow (C_1.radius + C_2.radius)/2$;

    $C_2 \leftarrow$ *resolve-viewing-angles*(T, p, C_1, r);

3.3 The corresponding *fixed* versions.

Both the radar-like algorithm and the expanding circular-horizon algorithm have corresponding *fixed* versions. In the fixed version of the radar-like algorithm the angle between two consecutive rays is fixed and we approximate the portion of R_p in the sector defined by the rays in any case, even if they are not **close-enough**; see Figure 4 [1]. In the fixed version of the expanding circular-horizon algorithm, the increase in radius between two consecutive circles is fixed and again we approximate the portion of R_p in the annulus defined by the circles in any case, see Figure 5.

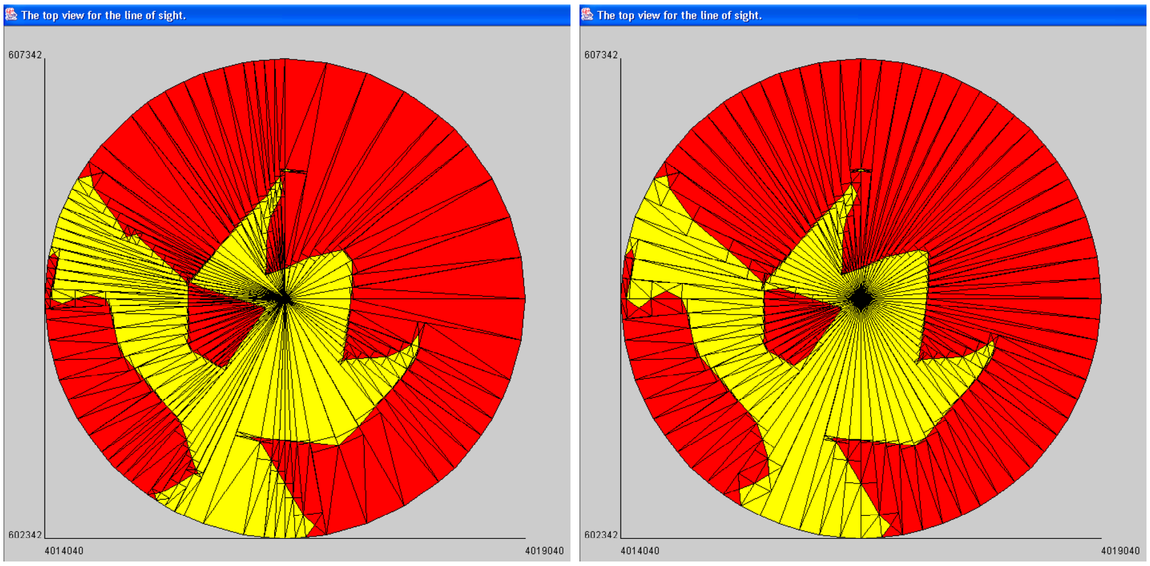


Figure 4: The visible region computed by the radar-like algorithm (left) and by its corresponding fixed version (right), each composed of 72 slices.

4 Experimental Results

In this section we report on the experiments that we performed with the approximation algorithms described in Sections 2 and 3. Namely, the radar-like algorithm, the expanding circular-horizon algorithm (ECH), and their corresponding fixed versions. We have also implemented the exact algorithm (Section 3.1), which is needed for the error computation.

[1]Each of the figures 4–6 involves a different data set.

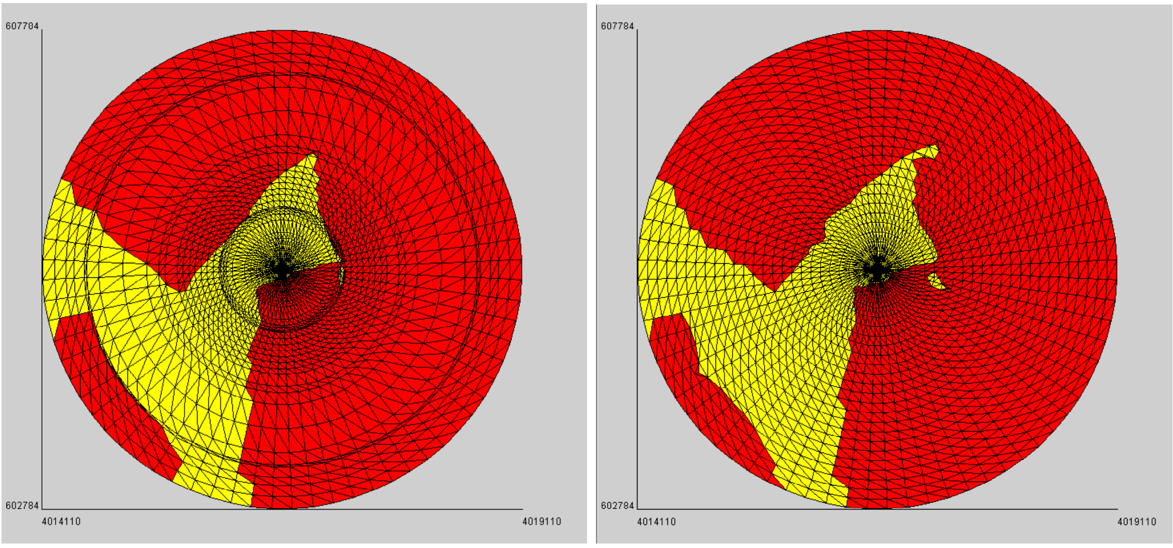


Figure 5: The visible region computed by the expanding circular-horizon algorithm (left) and by its corresponding fixed version (right).

4.1 The error measure.

In our experiments we use the following natural error measure. Let R'_p be an approximation of R_p obtained by some approximation algorithm, where R_p is the region visible from p. Then the error associated with R'_p is the area of the **XOR** of R'_p and R_p, divided by the area of the disk of radius l, where l is the range of sight that is in use. See Figure 6.

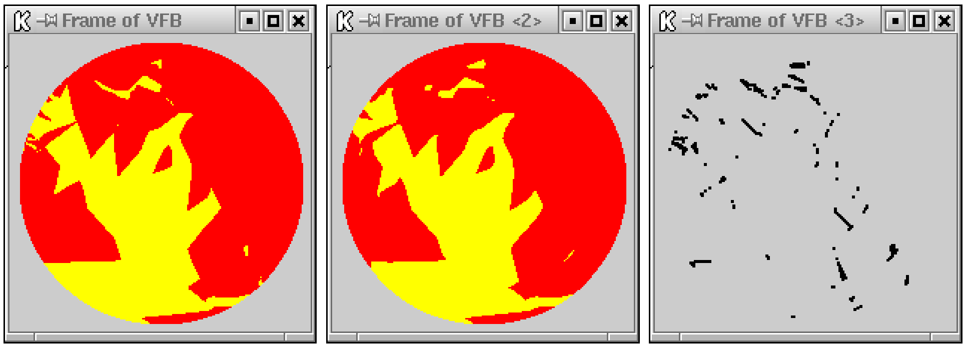


Figure 6: Left: the exact region R_p; Middle: the region R'_p computed by the radar-like algorithm; Right: $\mathbf{XOR}(R'_p, R_p)$.

4.2 The experiments.

Ten input terrains representing ten different and varied geographic regions were used. The terrains were downloaded from the "Geographic Data Download" page of the U.S. Geological Survey (USGS) web site [18]. Each input terrain (Delaunay triangulation) covers a rectangular area of approximately 15×10 km^2, and consists of approximately 5,000-10,000 vertices. For each terrain we picked several view points (x, y coordinates) randomly. For each view point p we applied each of the four approximation algorithms (as well as the exact algorithm) 20 times: once for each combination of height (either 1, 10, 20, or 50 meters above the surface of T) and range of sight (either 500, 1000, 1500, 2500, or 3500 meters). For each (approximated) region that was obtained, we computed the associated error, according to the error measure above. All this was repeated three times; once for each of the three levels of sampling (see below).

The *level of sampling* is determined by the number of calls to **interpolate** that are issued during the execution of an algorithm. We used three levels of sampling: 80, 140, and 220. Since the interpolation between two consecutive rays is comparable to the interpolation between two circular horizons, this seems a fair basis for comparison. (In order to achieve a specific level of sampling when running one of the non-fixed versions, we repeated the computation several times, with different values of the constant C, until the desired level of sampling was reached.)

4.3 The results.

Our results are presented in the following two sets of tables. The first three tables show the error for each of the four algorithms as a function of the sampling level and the range of sight. Consider, for example, the first table. This table contains our results for sampling level 80. The first line in this table corresponds to the fixed version of the expanding circular-horizon algorithm (ECH). The first entry in this line (4.53) is the average error

Sampling level: 80	500	1000	1500	2500	3500
Fixed ECH	4.53	3.97	3.87	3.88	3.75
ECH	3.71	3.49	3.40	3.29	3.35
Fixed radar-like	2.09	1.36	1.19	1.08	0.97
Radar-like	1.37	0.91	0.78	0.65	0.62

Table 1: Average error (in percents) for sampling level 80.

Sampling level: 140	500	1000	1500	2500	3500
Fixed ECH	2.71	2.35	2.12	2.16	2.12
ECH	2.30	2.09	1.88	1.88	1.94
Fixed radar-like	1.21	0.97	0.91	0.81	0.72
Radar-like	0.89	0.71	0.62	0.59	0.53

Table 2: Average error (in percents) for sampling level 140.

Sampling level: 220	500	1000	1500	2500	3500
Fixed ECH	1.36	1.30	1.18	1.19	1.20
ECH	1.22	1.15	1.13	1.09	1.13
Fixed radar-like	0.79	0.63	0.59	0.51	0.40
Radar-like	0.53	0.41	0.37	0.29	0.28

Table 3: Average error (in percents) for sampling level 220.

(in percents) obtained when running ECH with accuracy level 80 and range of sight 500 for each of the view points (over all terrains) and each of the four possible heights.

Tables 4 and 5 show the amount of work needed in order to reach a certain level of accuracy (i.e., error). We use two different cost models. In Table 4 the amount of work is measured by the number of calls to **cross-section** (alternatively, **resolve-viewing-angles**), and in Table 5 the amount of work is measured by the total running time. For example, using the fixed radar-like algorithm, the average number of calls to **cross-**

section needed to obtain an error of 1 percent was 80, and, using the ECH algorithm, the average running time needed to obtain an error of 0.5 percent was 1648 milliseconds. All experiments were performed on the following platform: Pentium 4, 2.4GHz, 512MB, Linux 8.1, Java 1.4.

Error:	1.00	0.75	0.50
Fixed ECH	263	616	1009
ECH	231	522	893
Fixed radar-like	80	140	220
Radar-like	61	103	174

Table 4: Average number of calls to **cross-section** (alternatively, **resolve-viewing-angles**) by accuracy level.

Error:	1.00	0.75	0.50
Fixed ECH	597	1045	1648
ECH	579	1012	1591
Fixed radar-like	112	192	301
Radar-like	101	168	274

Table 5: Average running time (in milliseconds) by accuracy level.

4.4 Conclusions.

Based on the results above the *radar-like* approach is significantly better than the *expanding circular-horizon* approach. For each of the sampling levels, the regions computed by the two radar-like algorithms were more accurate than those computed by the two ECH algorithms for any range of sight (see Tables 1-3). Moreover, for each level of accuracy, the ECH algorithms had to work much harder than the radar-like algorithms (according to both measures) in order to reach the desired level of accuracy (see Tables 4-5).

A possible explanation for the better performance of the radar-like algorithms is that in the ECH algorithms the computation of the visible arcs on a circular horizon is only an approximation — both the viewing angles ψ_q and the partition of the circular horizon into visible and invisible arcs that is based on these angles might be inaccuare. While in the radar-like algorithms the visible segments on a ray are computed exactly. Referring to Figure 7, if the ECH algorithms miss a ridge like the one in the left picture (drawn as a narrow rectangle), then all subsequent circles will miss it and therefore might conclude that the corresponding arcs are visible while they are not. On the other hand, a ridge like the one in the right picture that is missed by the radar-like algorithms does not affect subsequent computations leading to smaller errors.

The way in which the viewing angles ψ_q are computed also explains why the error of the ECH algorithms tends to be single-sided. That is, the area of the (approximated) visible regions computed by the ECH algorithms is usually larger that the area of the corresponding (exact) visible regions. The error of the radar-like algorithms, on the other hand, consists usually of both 'optimistic' and 'pessimistic' decisions.

Another clear conclusion is that the adaptive (i.e., non-fixed) versions are more accurate than their corresponding fixed versions. For each sampling level, the regions computed by the adaptive versions were significantly more accurate than those of the fixed versions (see Tables 1-3).

Of course, in the worst case, the error of all suggested algorithms is unbounded, since, even for relatively flat terrains with very moderate slopes, one can construct artificial examples, similar to the one in Figure 7, where the interpolated visibility between consecutive rays or circular horizons is completely wrong.

As expected, the adaptive versions are somewhat slower than the corresponding fixed versions for a given sampling level, since the adaptive versions perform more *cross-section* (alternatively, *resolve-viewing-angles*) operations. Actually, we found that on average the radar-like algorithm issues about 9 percent more calls to *cross-section* than the fixed radar-like algorithm. However, when taking into consideration the improved accuracy of the adaptive version, we see (Table 5) that the adaptive version is on average about 10 percent faster than the corresponding fixed version.

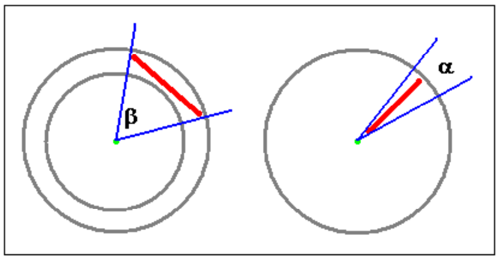


Figure 7: ECH vs. the radar-like algorithm.

Finally, we recently performed some experiments with the two radar-like algorithms using larger terrains, each consisting of $10^6 - 1.6 \cdot 10^7$ vertices and covering a rectangular area of 100×100 km^2. Using several levels of sampling and ranges of up to 50 km, the adaptive version remained more accurate than the fixed version. Also, as one might expect, the accuracy level of the radar-like algorithms improved when a higher resolution map of the same region was used (keeping the sampling level and range fixed). Moreover, since the running time of the radar-like algorithms is often sublinear, the gap between the running times of the radar-like algorithms and any exact algorithm becomes very significant as the size of the underlying terrain increases, and as long as the required error is not extremely small.

Acknowledgment. The authors wish to thank Ofir Ganani and Maor Mishkin who implemented the radar-like and the expanding circular-horizon algorithms.

Research by Ben-Moshe and Katz was partially supported by grant no. 2000160 from the U.S.-Israel Binational Science Foundation, and by the MAGNET program of the Israel Ministry of Industry and Trade (LSRT consortium). Research by Carmi is partially supported by a Kreitman Foundation doctoral fellowship.

References

[1] B. Ben-Moshe, M.J. Katz, J.S.B. Mitchell and Y. Nir. Visibility preserving terrain simplification. *Proc. 18th ACM Sympos. Comput. Geom.* 303–311, 2002.

[2] De Berg, van Kreveld, Overmars and Schwarzkopf. *Computational Geometry, Algorithms and Applications*, 2nd Edition. Springer, 2000.

[3] P. Bose, T. Shermer, G. Toussaint and B. Zhu. Guarding polyhedral terrains. *Comput. Geom. Theory Appl.* 7 (1997), 173–185.

[4] D. Cohen-Or and A. Shaked. Visibility and dead-zones in digital terrain maps. *Computer Graphics Forum* 14(3):171–179, 1995.

[5] R. Cole and M. Sharir. Visibility problems for polyhedral terrains. *J. of Symbolic Computation* 7:11–30, 1989.

[6] L. De Floriani and P. Magillo. Visibility algorithms on triangulated digital terrain models. *Internat. J. of GIS* 8(1):13–41, 1994.

[7] L. De Floriani and P. Magillo. Representing the visibility structure of a polyhedral terrain through a horizon map. *Internat. J. of GIS* 10(5):541–562, 1996.

[8] F. Devai. Quadratic bounds for hidden line elimination. *Proc. 2nd ACM Sympos. Comput. Geom.* 269–275, 1986.

[9] W.R. Franklin. Siting observers on terrain. *Symp. Spatial Data Handling*, 2002.

[10] W.R. Franklin, C.K. Ray and S. Mehta. Geometric algorithms for siting of air defense missile batteries. *Tech. Report*, 1994.

[11] P.S. Heckbert and M. Garland. Fast polygonal approximation of terrains and height fields. Report CMU-CS-95-181, Carnegie Mellon University, 1995.

[12] M.F. Goodchild and J. Lee. Coverage problems and visibility regions on topographic surfaces. *Annals of Operation Research* 18:175–186, 1989.

[13] N. Greene, M. Kass and G. Miller. Hierarchical z-buffer visibility. *Computer Graphics Proc., Annu. Conference Series* 273–278, 1993.

[14] M.J. Katz, M.H. Overmars and M. Sharir. Efficient hidden surface removal for objects with small union size. *Comput. Geom. Theory Appl.* 2 (1992), 223–234.

[15] J. Reif and S. Sen. An efficient output-sensitive hidden surface removal algorithm and its parallelization. *Proc. 4th ACM Sympos. Comput. Geom.* 193–200, 1988.

[16] M. Sharir. The shortest watchtower and related problems for polyhedral terrains. *Information Processing Letters* 29 (1988), 265–270.

[17] A.J. Stewart. Fast horizon computation at all points of a terrain with visibility and shading applications. *IEEE Trans. Visualization Computer Graphics* 4(1):82–93, 1998.

[18] U.S. Geological Survey (USGS) web site, Geographic Data Download page `http://edc.usgs.gov/geodata/`.

[19] Binhai Zhu. Computing the shortest watchtower of a polyhedral terrain in $O(n \log n)$ time. *Comput. Geom. Theory Appl.* 8 (1997), 181–193.