Appl Geomat (2017) 9:53–59
DOI 10.1007/s12518-016-0180-9

ORIGINAL PAPER

Approximation algorithms for visibility computation and testing over a terrain

Sharareh Alipour[1] · Mohammad Ghodsi[1,2] · Uğur Güdükbay[3] · Morteza Golkari[1]

Received: 19 April 2016 / Accepted: 6 December 2016 / Published online: 6 January 2017

Abstract Given a 2.5D terrain and a query point p on or above it, we want to find the triangles of terrain that are visible from p. We present an approximation algorithm to solve this problem. We implement the algorithm and test it on real data sets. The experimental results show that our approximate solution is very close to the exact solution and compared to the other similar works, the computational cost of our algorithm is lower. We analyze the computational complexity of the algorithm. We consider the visibility testing problem where the goal is to test whether a given triangle of the terrain is visible or not with respect to p. We present an algorithm for this problem and show that the average running time of this algorithm is the same as the running time of the case where we want to test the visibility between two query points p and q. We also propose a randomized algorithm for providing an estimate of the portion of the visible region of a terrain for a query point.

Keywords Visibility computation · Visibility counting problem · Visibility testing problem · Approximation algorithm · Randomized algorithm

✉ Sharareh Alipour
shalipour@ce.sharif.edu

Mohammad Ghodsi
ghodsi@sharif.edu

Uğur Güdükbay
gudukbay@cs.bilkent.edu.tr

Morteza Golkari
golkari@ce.sharif.edu

[1] Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

[2] Institute for Research in Fundamental Sciences (IPM), Tehran, Iran

[3] Department of Computer Engineering, Bilkent University, Ankara, Turkey

Introduction

Problem statement Suppose that T is a set of n disjoint triangles representing a 2.5D terrain. Two points $p, q \in R^3$ above or on T are visible to each other with respect to T if the line segment $\overline{pq}$ does not intersect any triangle of T. A triangle $\Delta \in T$ is also considered to be visible from a point p above or on T, if there exists a point $q \in \Delta$ such that p and q are visible to each other. Here, the visibility counting problem (VCP) is to find the number of triangles of T that are visible from any query point p. The visibility testing problem (VTP) is also defined as follows: given a query point p and a triangle $\Delta \in T$, we want to test whether Δ is visible from p.

Definition 1 The set of all points on T that are visible from a query point p is the visibility region of p and is denoted by *VR(p)*.

Related work Computing the visible region of a point is a well-known problem appearing in numerous applications (see, e.g., Cohen-Or and Shaked 1995; Cole and Sharir 1989; Floriani and Magillo 1994; Floriani and Magillo 1996; Franklin et al. 1994; Goodchild and Lee 1989; Stewart 1998). For example, the coverage area of an

antenna for which the line of sight may be approximated by clipping the region that is visible from the tip of the antenna with an appropriate disk centered at the antenna. As a similar problem, natural resource extractors may wish to site visual nuisances, such as clearcut forests and open-pit mines, where they cannot be seen from public roads. Zoning laws in some regions, such as the Adirondack Park of New York State, may obstruct new buildings that can be seen from a public lake (Franklin et al. 1994). Alsadik et al. (2014) proposed different methods for analyzing the visibility of point clouds from a photogrammetric viewpoint and developed surface-based and voxel-based visibility and hidden point removal (HPR) algorithms.

The complexity of computing the visibility region of a point, *VR(p)*, might be $\Omega(n^2)$ (Cole and Sharir 1989; Devai 1986), where n is the number of triangles in T. Therefore, approximation algorithms that compute an approximation of *VR(p)* can highly reduce the running time. 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 (Ben-Moshe et al. 2008). Note that the terrain representation is fixed and cannot be modified during the running time of the algorithm.

Ben-Moshe et al. (2008) propose a generic radar-like algorithm for computing an approximation of *VR(p)*. The algorithm extrapolates 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. Ben-Moshe et al. (2008) suggest a specific way to measure the resemblance (difference) and to extrapolate the visible region between two consecutive rays. They also present an alternative algorithm, called *Expanding Circular-Horizon (ECH)*, that 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 computing the visibility from a point. Overall, they proposed four algorithms, *fixed ECH*, *ECH*, *fixed radar-like*, and *radar-like* to approximate *VR(p)*. It is shown that the radar-like algorithm is the fastest among these four algorithms.

Our result We propose an algorithm to approximate the number of visible triangles of a terrain from a query point p, which is denoted by m_p. Moreover, the algorithm can be used to approximate *VR(p)*. We also consider the visibility testing problem and present our experimental results on real data sets. We represent the surface of the terrain as a triangulation mesh. The main idea of the algorithm is to compute the visible edges and vertices of the triangles. A preliminary version of the proposed algorithms and the experimental results are presented in Alipour et al. (2014).

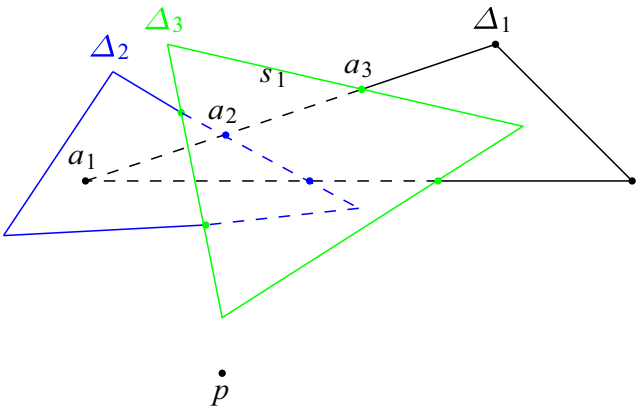


Fig. 1 The proposed algorithm for VTP. The *dashed lines* are the portion of the segments not visible from p. Also, note that the *triangles* shown are part of a terrain and for simplicity other triangles are not shown in this figure. a_1 is not visible from p and Δ_2 covers $\overline{a_1 a_2}$. In the next step, Δ_3 covers $a_2 a_3$. If we continue, either we reach the end of s_1 which means that s_1 is not visible from p or we find a visible point on s_1 which means that s_1 is visible from p

The structure of the paper is as follows: in "Visibility testing problem" section, we present the algorithm for visibility testing between a point and a triangle and the results of testing the algorithm on real data sets. We also provide the average running times and error rates of the proposed algorithm. In "The visibility counting problem" section, we present two approximation algorithms for the visibility counting problem and also an exact algorithm, which is used to compare the approximated solution to the exact solution. In "Experimental results for visibility counting" section, we provide the experimental results of the

Table 1 The average number of triangles that cover a triangle according to a random query point for each data set

Number of vertices	Average number of covering triangles
2400	1.70
2400	1.18
2400	1.06
5400	1.58
5400	1.09
5,400	1.03
9,600	1.53
9,600	1.10
9,600	1.04
29,400	1.48
29,400	1.02
29,400	1.08
60,000	1.51
60,000	1.37
60,000	1.08

proposed algorithm and compare our results to the results of Ben-Moshe et al. (2008). We conclude in "Conclusion" section and give some future research directions.

Visibility testing problem

Suppose that we are given a set S of n triangles in R^3 and a query point p. We classify the visible triangles of S from p into two groups:

- A triangle is an *edge-visible* triangle if there is a point on its edges that is visible from p.
- If the triangle is visible from p but there is not any visible point on its edges, then it is called a *mid-visible* triangle.

Lemma 1 *If the triangles of S form a terrain T, then all the visible triangles from a query point p are edge-visible.*

Using Lemma 1, to compute the number of visible triangles of a terrain T from a given query point p, it is enough to consider the edges of triangles.

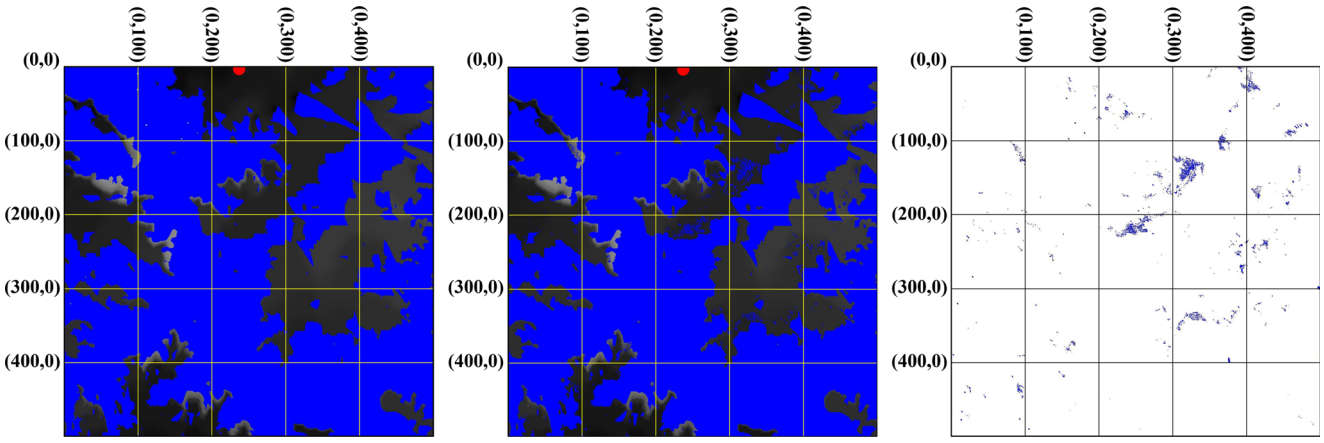
The proposed algorithm for visibility testing

According to Lemma 1, if a triangle $\Delta_1 \in T$ is visible from a query point p, then Δ_1 is edge-visible. Therefore, for each edge of Δ_1, starting from one of its vertices, a_1, we check the vertices. If a_1 is visible from p, then Δ_1 is visible to p and we terminate the algorithm, otherwise, we choose the first triangle Δ_2 hit by the ray emanating from p to a_1. Δ_2 covers some part of that edge. So, it is enough to check whether the remaining part of the edge is visible from p. For the remaining part, we consider the first point a_2 on that edge that is not covered by Δ_2. We shoot a ray from a_2 to p. If a_2 is visible from p, then Δ_1 is visible from p, otherwise, we consider the first triangle hit by the ray emanating from a_2 to p. We run the algorithm on Δ_2 in the same way as Δ_1. If we reach the end of the edge, then that edge is not visible from p. If all three edges of Δ_1 are invisible from p, then Δ_1 is also invisible from p (cf. Algorithm 1 and Fig. 1).

Algorithm 1 Algorithm for Visibility Testing

Input:
$\Delta \in T$.
p: a query point
Output:
A boolean indicates whether triangle Δ is visible to p or not.
Method:

```
1: Create three line segments s1,s2 and s3 based on Δ's edges.
2: Initialize segments queue segQueue with s1,s2 and s3.
3: while segQueue is not empty do
4:    seg = segQueue.pop()
5:    if At least one point on the line segment seg is visible from p then
6:       return true
7:    else
8:       Find the first concealer triangle T_C for invisible point p on the line segment seg.
9:       for each segment seg_t in segments queue segQueue do
10:         COVER LINE SEGMENT(p,T_C, seg_t)   ▷ this function removes invisible part from segment seg_1 according to p and T_C and divides seg_1 to new segments
11:      end for
12:   end if
13: end while
14: return false
```



(a) Approximated visible region (b) Actual visible region (c) Subtraction of the approximated visible region (a) from the exact visible region (b)

Fig. 2 The approximate **a** the exact **b** visible region of the query point p(3.3, 242.5, 2089.5). The query point is shown in *red*. The *blue areas* are not visible to the query point and all the other areas are visible. As the height of a visible area increases the color of that area is shown darker. Therefore, the *dark parts* are associated to the mountains and the white parts are associated to the valleys

Experimental results of visibility testing

The computational complexity of Algorithm 1 for each query point p and triangle Δ_1 depends on the number of rays shot, which is equal to the number of triangles covering the edges of Δ_1. We denote this number by $t_p(\Delta_1)$. Therefore, the computational complexity of this algorithm is $O(t_p(\Delta) f(n))$ in which $f(n)$ is the time for each ray shooting. In our data sets, the terrain is represented as a height field where a height value is specified for each point (i, j) on the regular 2D grid. The regular grid is triangulated to represent the terrain. We use a regular triangulation such that for every four points $x_1 = (i, j, k_1), x_2 = (i + 1, j, k_2), x_3 = (i, j + 1, k_3), x_4 = (i + 1, j + 1, k_4)$, two triangles are constructed: $\Delta_1 = (x_1, x_2, x_4)$ and $\Delta_2 = (x_1, x_3, x_4)$. In our experiments, we do not preprocess the triangles of terrain, so $f(n) = O(\sqrt(n))$. For each tested data set, we choose random points and run Algorithm 1 for each point p and each of the triangles of the terrain. For each triangle $\Delta_1 \in T$, we calculate $t_p(\Delta_1)$. The experimental results indicate:

$$E(t_p(\Delta_1)) = \sum_{\Delta_i \in T}^{n} \frac{t_p(\Delta_i)}{n} = O(1),$$

which means that the average number of triangles covering a triangle for a random query point is $O(1)$. So, the average running time of visibility testing between a triangle and a random query point is $O(f(n))$. Table 1 shows the average t_p for each data set, which is always smaller than two. It is shown that the average number of $t_p(\Delta)$ is independent of the size of data set.

The visibility counting problem

Our visibility counting algorithm is based on counting the number of triangles whose vertices are visible from the query point. The visibility region of a query point p consists of triangles whose vertices are visible from p and these triangles are stored in a queue structure.

Approximation algorithm for visibility counting

For each query point p, if at least one of the vertices of a triangle is visible from p, then we consider it as a visible triangle. To compute the number of triangles which at least one of their vertices are visible from p, we propose the following algorithm:

First, we emanate a random ray from p to the surface of T and select the first triangle Δ_1 that is intersects this ray. Obviously, Δ_1 is visible from p. In the next step, we consider the three neighbors of Δ_1 ; we check whether the vertices of these triangles are visible from p. If at least one of the vertices of these triangles is visible from p, then the triangle is visible from p. We run the algorithm recursively on the neighbors of the triangles. Otherwise, we decide that the triangle is an invisible triangle. For each non-visible triangle, we shoot a ray emanating from p to the vertices of visible triangles; the first triangle hit by that ray is considered as a visible triangle and we recursively run the algorithm on its neighbors. Algorithm 2 shows pseudo code of this algorithm.

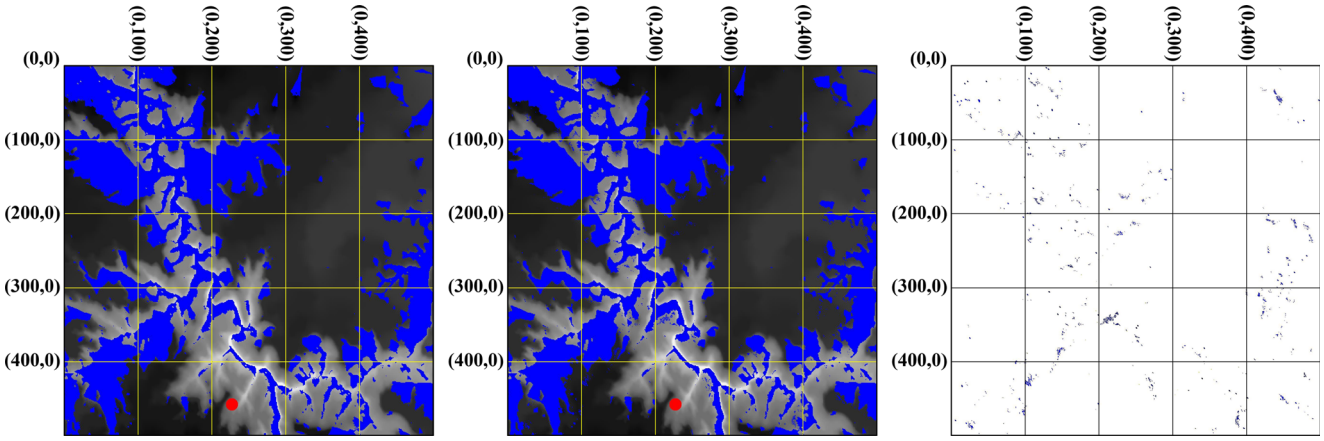
Algorithm 2 Approximation algorithm for computing the visible triangles

Input:
T: A 2D array of triangles.
e: A query point p.
Output:
The number of visible triangles from p.
Method:

```
1: Create empty Queue, bfsQueue.
2: Find the first triangle Δ1 intersected by a random ray
   emanating from p to the surface of terrain.
3: Add Δ1 to bfsQueue.
4: while bfsQueue is not empty do
5:    Δt = bfsQueue.pop()
6:    if At least one vertex of triangle Δt is visible from p
      then
7:       Add three neighbors of Δ1 to bfsQueue.
8:    else
9:       Find concealer triangles that cover vertices of trian-
         gle Δt.
10:      Shoot rays from p to the vertices of concealer trian-
         gles
11:      Find hit points of shooting ray and the surface of
         the terrain.
12:      Add triangles corresponding to the hit points to bf-
         sQueue.
13:   end if
14: end while
```

Computational complexity

In Algorithm 2, each triangle is stored in the queue just once. For each triangle, we shoot three rays to the vertices of that triangle. In each step of ray shooting, we check whether there is a triangle between p and a vertex. For each ray, we want to find the first triangle hit by a ray emanating from p to a specific direction. In both of these ray shootings, we have to check at most $O(\sqrt{n})$ triangles. So, the computational complexity of the algorithm is $O(\sqrt{n} m_p)$ where m_p is the number of visible triangles. It should be noted that if we preprocess the triangles, each ray shooting would take O(log n) time and this could reduce the computational complexity.



(a) Approximated visible region (b) Actual visible region (c) Subtraction of the approximated visible region (a) from the exact visible region (b)

Fig. 3 The approximate **a** and the exact **b** visible region of the query point $p(468.8, 232.5, 3228.6)$. The query point is shown in *red*. The *blue areas* are not visible to the query point and all the other areas are visible. As the height of a visible area increases the color of that area is shown darker, so the *dark parts* are associated to the mountains and the white parts are associated to the valleys

Randomized algorithm for visibility counting problem

We present a randomized algorithm to approximate the answer of visibility counting problem. Our randomized algorithm is based on random sampling and is similar to one introduced in (Alipour and Zarei 2011). To find the number of visible triangles, we first run the algorithm proposed in the previous section at most for $O(\sqrt{n})$ time. Obviously, if $m_p < \sqrt{n}$, the algorithm will terminate and we give an approximation value of m_p. If $m_p > \sqrt{n}$, we use the following algorithm. Suppose that the number of triangles is n. We choose each triangle with the probability of $\frac{1}{\sqrt{n}}$ and use the visibility testing algorithm for each triangle. We compute the number of triangles that are visible from p, multiply this number by $\sqrt{n}$, and report it as the approximated value of m_p. We can run this algorithm for t times and report the mean value of these t values. In (Alipour and Zarei 2011), it is shown that by Chebishev Lemma, our approximated value will be a $1+\delta$ approximation of the real solution.

The computational complexity of this algorithm is $O(\frac{1}{\delta^2}\sqrt{n}\ f(n)\ \log n)$ because we run the visibility testing algorithm for the selected triangles. Here, δ is the approximation factor. For more details on the approximation factor, please refer to (Alipour and Zarei 2011).

Using Algorithm 1, we test whether each triangle of T is visible from p. So, we can compute the exact number of visible triangles. We use this data in our experimental results to show that the approximate number of visible triangles is close to the exact number of visible triangles.

Table 2 The average running time for each data set. For each data set, the height of some random points are chosen as specific values

Number of vertices	Running time (ms)	Error (%)
2400	17	1.39
2400	14	0.45
2400	13	0.41
5400	44	2.03
5400	36	0.89
5400	28	0.45
9600	90	2.28
9600	80	1.17
9600	66	0.90
29,400	355	3.92
29,400	396	1.78
29,400	365	2.47
60,000	840	5.28
60,000	870	3.17
60,000	954	3.70

Experimental results for visibility counting

We ran the approximation algorithm on three real terrain data sets. The terrain is represented as a height field that for each point (i, j) on the regular 2D grid, a height value is specified. The regular grid is triangulated to represent the terrain as described previously. We run the proposed algorithm on height-field representation of the terrain.

Figures 2 and 3 show the actual and approximate visible regions of two query points. We calculate the approximate visible region of a query point using Algorithm 2 and use the exact algorithm to calculate the exact number of visible region of a query point. It is seen that the approximated visibility region of a query point is close to the actual visibility region of that point.

Algorithm 3 Randomized Algorithm for Visibility Counting

Input:
Triangles of terrain T.
p: a query point.
t: number of execution iterations of randomized algorithm.
Output:
counter: the number of triangles of terrain T visible from p
Method:

```
set p to 1/√n where n is number of triangles in terrain T.
set counter to 0.
loop
    t iterations
    for each triangle Δ of terrain T do
        select triangle Δ with probability of p.
        use VTP algorithm to check whether triangle Δ is
        visible from p or not.
        if Δ is visible from p then
            counter++
        end if
    end for
end loop
return counter/(t×p)
```

We define the error measure as follows. Let m'_p be the number of approximated visible triangles. Then the error associated with m'_p is $(m_p - m'_p)$ divided by m_p. For each data set, we choose 1000 random query points and calculate the error using m_p and m'_p for each query point p. Each random point is chosen in different heights above the terrain. It is obvious that the increase in the height of a point results in the increase of the number of visible triangles and the running time as well. Table 2 shows the average computation time and average error for each data set.

We compare our results to the results of the approach proposed by Ben-Moshe et al. (2008). They proposed four algorithms and measured the performances of these algorithms. We use data sets similar to the ones used in the experiments presented in Ben-Moshe et al. (2008). In their experiments, they tested ten input terrains representing different geographic regions. Each input terrain covers a rectangular area of approximately 5000–10,000 triangle vertices. For each terrain, they picked several view points (x, y coordinates) randomly. For each query point p, they 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 m above the surface of the terrain) and range of sight (either 500, 1000, 1500, 2500, or 3500 m). For each (approximated) region that was obtained, they computed the associated error according to the error measure they used.

Their error measure is defined as follows: Let R_p be an approximation of R_p obtained by some approximation algorithm where R_p is the region visible from 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 sight that is in use. In the case that R_p is small compared to the the sight and the difference between R'_p and R_p is high compared to R_p, the error will be very small. However, the difference between R'_p and R_p is generally high. Thus, our error measure is more accurate than theirs. The data sets they tested contain 5000–10,000 triangle vertices. We also tested some data sets with the number of triangles around 10,000.

As it is shown in Table 3, the best average running time of their algorithm is 274 ms with the error of 0.5 (using their error measure), whereas the average running time of our proposed algorithm is 70 ms with the error of 0.55 (using our error measure). If we use their error measure, then our error would be 0.35.

The test environment for our experiment and the one used in Ben-Moshe et al. (2008) are described in Table 4.

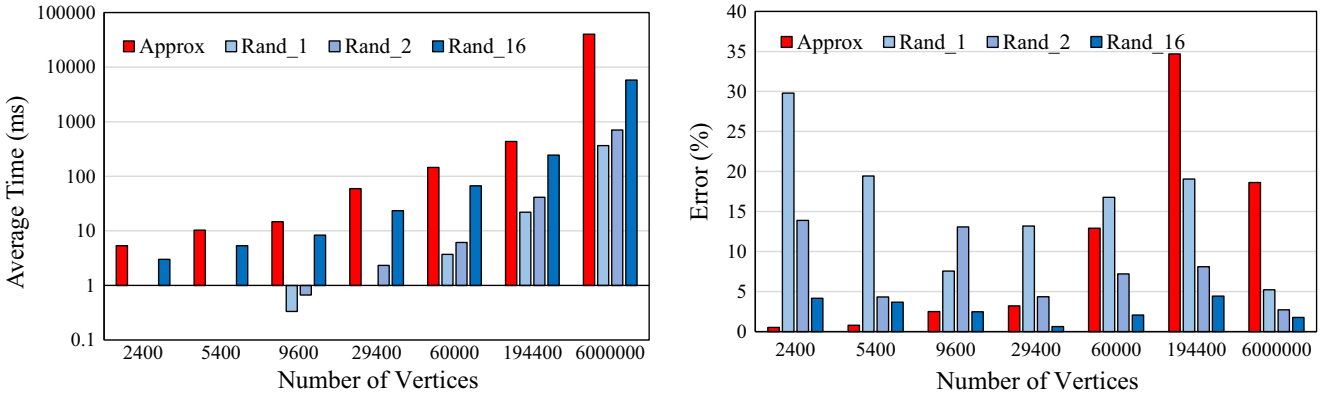
Figure 4 presents the average running times and the error rates of the proposed approximate and randomized algorithms. It should be noted that the proposed approximate algorithm (Algorithm 2) calculates the visibility region whereas the randomized algorithm (Algorithm 3) gives only an approximate measure of the visible region in terms of the number of triangles. The running times and the error rates of the proposed approximate algorithm are better than those of the Radar-like algorithm proposed by (Ben-Moshe et al. 2008). As it is expected, the running time of the randomized algorithm is significantly lower than that of the approximate algorithm. However, the randomized algorithm is only

Table 3 The results of Ben-Moshe et al. (2008). They define an error function for approximating the visible region of a query point. The running times of the algorithms (ms) are given for each error value. The best running time is obtained by the Radar-like algorithm

Error	1.00	0.75	0.5
Fixed ECH	597	1.0045	1.648
ECH	579	1.012	1.591
Fixed radar-like	112	192	301
Radar-like	101	168	274

Table 4 The test environments of Ben-Moshe et al. (2008) and ours

Ben-Moshe	Pentium 4	2.4 GHz	Linux 8.1	Java 1.4
Our platform	Core i5	2.4 GHz	Win10	Java 1.8



(a) Average running time (b) Average error

Fig. 4 The average running time and average error of the approximate and random algorithms

applicable for the cases where we do not need the visible region but we need a measure of the portion of the terrain that is visible from a query point.

Conclusion

We propose algorithms for the visibility counting and testing problems on a terrain. We implement our algorithm for VTP and the experimental results shows that the average running time of visibility testing between a query point and a triangle is almost equal to the running time of visibility testing between two points.

We also propose an approximation algorithm for VCP and tested it on real data sets. We compare our algorithm with the algorithms proposed by Ben-Moshe et al. (2008), which are the state-of-the-art algorithms for the same problem. We show that in almost similar conditions (considering the platforms, the number of triangles of terrains), the running time of our algorithm is better than that of their radar-like algorithm. Also, we would like to propose exact algorithms for VCP. Another possible extension is to adapt the proposed algorithms to the case where triangles are in 3D.

Acknowledgments Grand Canyon Data obtained from the United States Geological Survey (USGS), with processing by Chad McCabe of the Microsoft Geography Product Unit.

References

Alipour S, Zarei A (2011) Visibility testing and counting. In: Proceedings of the 5th Joint International Frontiers in Algorithmics, and 7th International Conference on Algorithmic Aspects in Information and Management, Springer-Verlag, Berlin, Heidelberg, FAW-AAIM'11, pp 343–351. http://dl.acm.org/citation.cfm?id=2021911.2021949

Alipour S, Ghodsi M, Güdükbay U, Golkari M (2014) An approximation algorithm for computing the visibility region of a point on a terrain and visibility testing. In: Proceedings of the International Conference on Computer Vision Theory and Applications. IEEE, Piscataway, New Jersey, pp 699–704. VISAPP 2014

Alsadik B, Gerke M, Vosselman G (2014) Visibility analysis of point cloud in close range photogrammetry. ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences pp 9–16. doi:10.5194/isprsannals-II-5-9-2014

Ben-Moshe B, Carmi P, Katz MJ (2008) Approximating the visible region of a point on a terrain. Geoinformatica 12(1):21–36. doi:10.1007/s10707-006-0017-5

Cohen-Or D, Shaked A (1995) Visibility and dead-zones in digital terrain maps. Comput Graph Forum 14(1):171–180. http://dblp.uni-trier.de/db/journals/cgf/cgf14.html#Cohen-OrS95

Cole R, Sharir M (1989) Visibility problems for polyhedral terrains. J Symb Comput 7(1):11 – 30. doi:10.1016/S0747-7171(89)80003-3. http://www.sciencedirect.com/science/article/pii/S0747717189800033

Devai F (1986) Quadratic bounds for hidden line elimination. In: Proceedings of the second annual symposium on Computational geometry, ACM, New York, NY, USA, SCG '86, pp 269–275. doi:10.1145/10515.10544

Floriani LD, Magillo P (1994) Visibility algorithms on triangulated digital terrain models. Int J Geogr Inf Syst 8(1):13–41. doi:10.1080/02693799408901985

Floriani LD, Magillo P (1996) Representing the visibility structure of a polyhedral terrain through a horizon map. Int J Geogr Inf Syst 10(5):541–561. doi:10.1080/02693799608902096

Franklin WR, Ray CK, Shashank M (1994) Geometric algorithms for siting of air defense missile batteries. Battelle Inc., Columbus OH

Goodchild M, Lee J (1989) Coverage problems and visibility regions on topographic surfaces. Ann Oper Res 18(1):175–186. doi:10.1007/BF02097802

Stewart A (1998) Fast horizon computation at all points of a terrain with visibility and shading applications. IEEE Trans Vis Comput Graph 4(1):82–93. doi:10.1109/2945.675656