Applied Geomatics manuscript No.
(will be inserted by the editor)

Approximation Algorithms for Visibility Computation and Testing over a Terrain

Sharareh Alipour · Mohammad Ghodsi · Uğur Güdükbay · Morteza Golkari

the date of receipt and acceptance should be inserted later

Abstract Given a 2.5D terrain and a query point p on or above it, we want to count the number of triangles on the 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 running time of our algorithm is superior. We analyze the computational complexity of the algorithm. We also 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 propose 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.

Sh. Alipour
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.
E-mail: shalipour@cs.sharif.edu

M. Ghodsi
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.
E-mail: ghodsi@sharif.edu

U. Güdükbay
Department of Computer Engineering, Bilkent University, Ankara, Turkey.
E-mail: gudukbay@cs.bilkent.edu.tr

M. Golkari
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.
E-mail: golkari@ce.sharif.edu

1 Introduction

Problem Statement. Suppose that T is a set of n disjoint triangles representing a 2.5D terrain. Two points $p, q \in R^3$ on or above 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 on or above 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 a 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, an algorithm that computes an approximation of *VR(p)* can highly reduce the computation 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). It should be noted 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 propose an alternative algorithm that uses circles of increasing radii centered at p instead of rays emanating from p. Both algorithms compute a representation of the (approximate) visible region that is especially suitable for computing the visibility from a point. Overall, they used 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).

The rest of the paper is organized as follows. Section 2 presents the algorithm for visibility testing between a point and a triangle and the results of testing the algorithm. We also provide the average running times and error rates of the proposed algorithm. Section 3 describes two approximation algorithms for the visibility counting problem and an exact algorithm that is used to compare the approximated solution to the exact solution. Section 4 provides the experimental results of the proposed algorithm on real data sets and compare our results to the results of Ben-Moshe et al (2008). Section 5 gives conclusions and future research directions.

2 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.

2.1 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, begin from one of its vertices, such as a_1. 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 Figure 1).

2.2 Experimental Results of Visibility Testing

The computational complexity of Algorithm 1 for query point p and triangle Δ_1 depends on the number of rays

Algorithm 1 Algorithm for Visibility Testing

Input:
$\varDelta \in T$: A triangle over the terrain
p: A query point
Output:
A boolean value that indicates whether triangle $\varDelta$ is visible from p or not.
Method:

```
Create three line segments s1, s2 and s3 based on Δ's edges.
Initialize segments queue segQueue with s1, s2 and s3.
while segQueue is not empty do
    seg = segQueue.pop()
    if At least one point on the line segment seg is visible from p then
        return true
    else
        Find the first concealer triangle T_C for invisible point p on the line segment seg.
        for each segment seg_t in segments queue segQueue do
            COVER LINE SEGMENT(p, T_C, seg_t)    ▷ This function removes the invisible part from segment seg1 according to p and T_C and divides seg1 to new segments
        end for
    end if
end while
return false
```

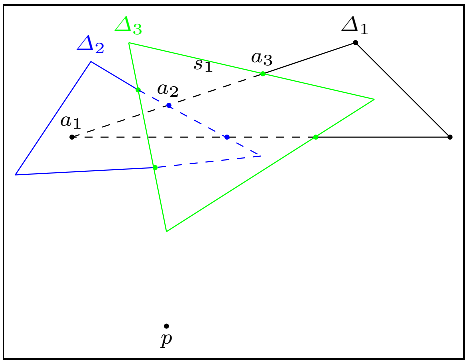


Fig. 1: The proposed algorithm for visibility testing. The dashed lines are the portions of the line segments that are not visible from p. a_1 is not visible from p and $\varDelta_2$ covers $\overline{a_1 a_2}$. In the next step, $\varDelta_3$ covers $a_2 a_3$. If we continue, either we reach the end 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.

shot, which is equal to the number of triangles covering the edges of $\varDelta_1$. We denote this number by $t_p(\varDelta_1)$. Therefore, the computational complexity of this algorithm is $O(t_p(\varDelta)f(n))$ in which $f(n)$ is the time for shooting each ray. In our data sets, the terrain is represented as a height field where a height value is specified for each point (i, j) on a 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: $\varDelta_1 = (x_1, x_2, x_4)$ and $\varDelta_2 = (x_1, x_3, x_4)$. In our experiments, we do not preprocess the triangles of the 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 triangle of the terrain. For each triangle $\varDelta_1 \in T$, we calculate $t_p(\varDelta_1)$. The experimental results indicate:

$$E(t_p(\varDelta_1)) = \sum_{\varDelta_i \in T}^{n} \frac{t_p(\varDelta_i)}{n} = O(1),$$

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 smaller than two for the tested data sets. It is seen that the average number of $t_p(\varDelta)$ is independent of the size of the data set.

Number of vertices	Average number of covering triangles
2,400	1.70
2,400	1.18
2,400	1.06
5,400	1.58
5,400	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

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

3 The Visibility Counting Problem

Our visibility counting algorithm is based on counting the number of triangles whose vertices are visible from a query point. To compute the visible region of query point p, the algorithm can be easily modified to output the triangles whose vertices are visible from p.

3.1 Approximation Algorithm for Visibility Counting

For each query point p, if at least one of the vertices of a triangle is visible from p, we consider it as a visible triangle. To compute the number of triangles that 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 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 the pseudocode of the approximation algorithm for counting the visible triangles.

Algorithm 2 Approximation algorithm for counting the visible triangles

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

Create empty Queue, *bfsQueue*.
Find the first triangle Δ_1 intersected by a random ray emanating from p to the surface of terrain.
Add Δ_1 to *bfsQueue*.
while bfsQueue is not empty **do**
    Δ_t = *bfsQueue.pop()*
    if At least one vertex of triangle Δ_t is visible from p **then**
        Add three neighbors of Δ_1 to *bfsQueue*.
    else
        Find concealer triangles that cover vertices of triangle Δ_t.
        Shoot rays from p to the vertices of concealer triangles.
        Find the intersection points of the rays shot and the surface of the terrain.
        Add triangles corresponding to the intersection points to *bfsQueue*.
    end if
end while

3.2 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.

3.3 Randomized Algorithm for Visibility Counting

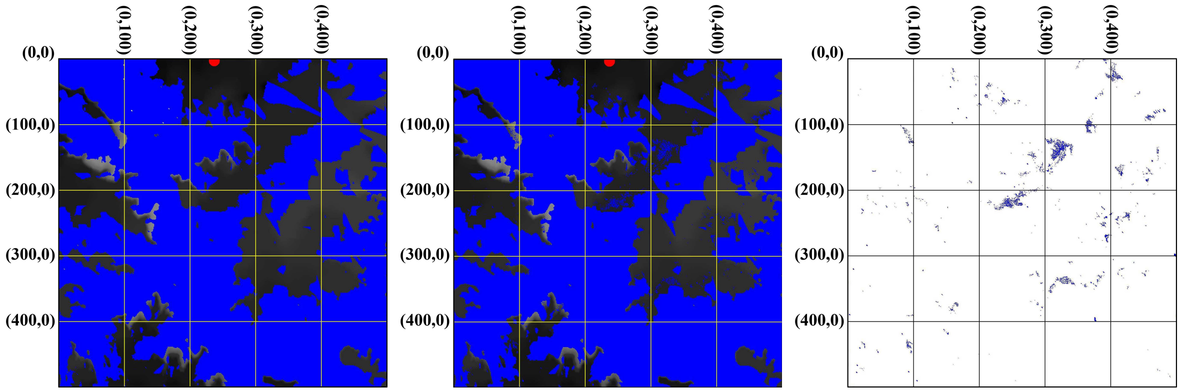
We present an approximate randomized algorithm for the visibility counting problem. Our algorithm is based on random sampling and is similar to the one introduced in Alipour and Zarei (2011). To find the number of visible triangles, we first run the algorithm proposed in the previous section at most for $O(\sqrt{n})$ times. Obviously, if $m_p < \sqrt{n}$, the algorithm will terminate and output an approximation 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 perform visibility testing for each triangle. We compute the number of triangles that are visible to p, multiply this number by $\sqrt{n}$, and report it as the approximate value of m_p. We run this algorithm for t times and report the mean value of these t values.

Alipour and Zarei (2011) show that, by Chebishev Lemma, the approximate 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 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.

4 Experimental Results for Visibility Counting

We ran the approximation algorithm on three real terrain data sets. The terrain is represented as a height

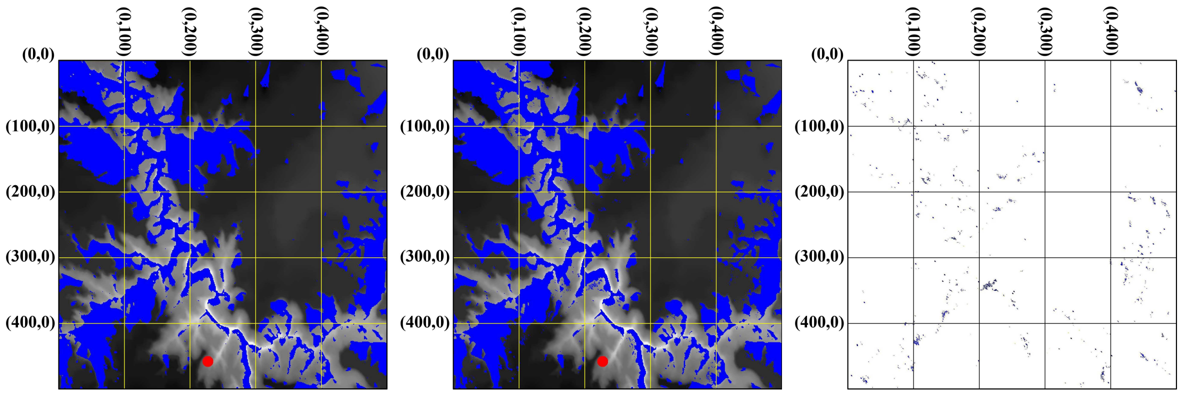


(a) Approximated visible region

(b) Actual visible region

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

Fig. 2: The approximate (a) and 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.



(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.

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 in Subsection 2.2. 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 approximate visibility region of a query point is close to the actual visibility region of that point.

We define the error measure as follows. Let m'_p be the number of visible triangles in the approximate solution. 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 at different heights above the terrain. It is obvious that the increase in the height of a point results in an increase in the number of visible triangles and the run-

Algorithm 3 Randomized Algorithm for Visibility Counting

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

set p to $\frac{1}{\sqrt{n}}$ 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 visibility testing 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 $\frac{counter}{t\ p}$

ning time as well. Table 2 shows the average time and average error for each data set. We compare our results

Number of vertices	Running time (ms)	Error (%)
2,400	17	1.39
2,400	14	0.45
2,400	13	0.41
5,400	44	2.03
5,400	36	0.89
5,400	28	0.45
9,600	90	2.28
9,600	80	1.17
9,600	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

Table 2: The average running times for each data set. For each data set, some random points are selected at different heights.

to the results of the approach proposed by Ben-Moshe et al (2008). They proposed four algorithms and measured the computational costs 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 $5,000-10,000$ 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 meters above the surface of the terrain) and range of sight (either 500, 1,000, 1,500, 2,500, or 3,500 meters). For each (approximate) 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 exclusive or 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 $5,000-10,000$ triangle vertices. We also tested some data sets with the number of triangles is 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 will be 0.35.

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 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.

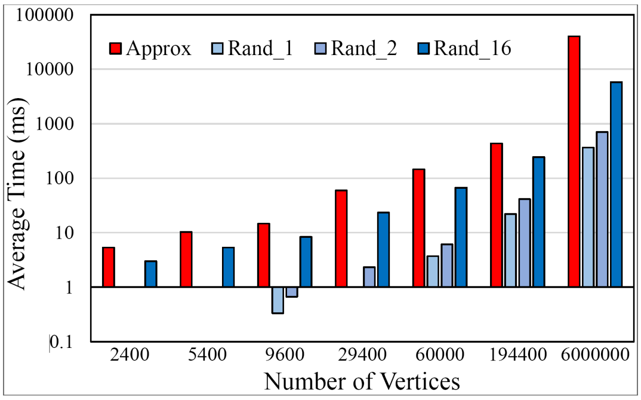
The test environment for our experiments and the one used in Ben-Moshe et al (2008) are described in Table 4.

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 calculates the visibility region whereas the randomized algorithm gives only an approximate measure of the visible region in terms of the

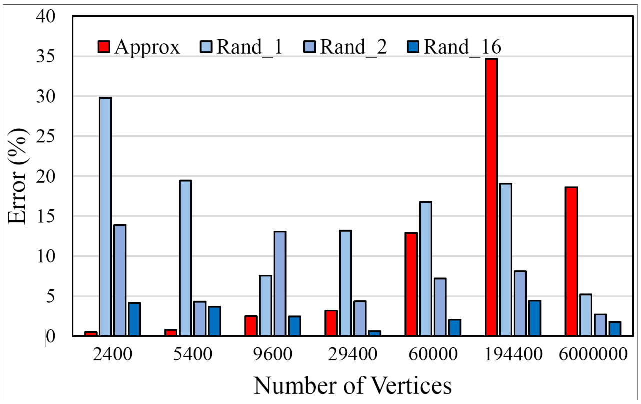
Ben-Moshe	Pentium 4	2.4GHz	Linux 8.1	Java 1.4
Our platform	Core i5	2.4GHz	Win10	Java 1.8

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

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 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.



(a) Average running time



(b) Average error

Fig. 4: The average running times and the error rates of the approximate and random algorithms.

5 Conclusion

We propose algorithms for visibility computation and testing over a terrain. We implement our algorithm for visibility testing and the experimental results show 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 visibility counting 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 used in the experiments), the running time of our algorithm is better than that of their radar-like algorithm. As a future extension, we would like to propose exact algorithms for visibility counting. Another possible extension is to adapt the proposed algorithms to the case where triangles are in 3D.

6 Acknowledgement

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

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, VISAPP 2014, pp 699–704

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 Volume II-5:9–16

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

Cohen-Or D, Shaked A (1995) Visibility and dead-zones in digital terrain maps. Computer Graphics Forum 14(3):171–180

Cole R, Sharir M (1989) Visibility problems for polyhedral terrains. Journal of Symbolic Computation 7(1):11 – 30

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

Floriani LD, Magillo P (1994) Visibility algorithms on triangulated digital terrain models. International Journal of Geographical Information Systems 8(1):13–41

Floriani LD, Magillo P (1996) Representing the visibility structure of a polyhedral terrain through a horizon map. International Journal of Geographical Information Systems 10(5):541–561

Franklin WR, Ray CK, Shashank M (1994) Geometric Algorithms for Siting of Air Defense Missile Batteries. Battelle, Inc., Columbus OH

Goodchild MF, Lee J (1989) Coverage problems and visibility regions on topographic surfaces. Annals of Operations Research 18(1):175–186

Stewart A (1998) Fast horizon computation at all points of a terrain with visibility and shading applications. IEEE Transactions on Visualization and Computer Graphics 4(1):82–93