An Easier Approach to Visible Edge Determination from Moving Viewpoint

Ishat E Rabban, Kaysar Abdullah, Shibbir Ahmed, and M. Sohel Rahman
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh
Email: {ieranikg, kzr.buet08, shibbirahmedtanvin, sohel.kcl}@gmail.com

***Abstract*— In this paper, we present an algorithm for visible edge determination from moving viewpoint in 2D-space by an easier approach. We show implementation details and performance analysis of the algorithm to justify the efficiency and correctness of the proposed algorithm. The time complexity of the algorithm is $O(n^2)$ in the worst case but we hope time complexity can be reduced practically because of the effective optimization in case of sorting angularly and finding intersection of points in a fixed range. Further modification of this algorithm is expected to determine the visible surface from moving perspective yielding the near optimal solution in 3D-space.**

Keywords—Algorithms; Computational Geometry; Visible Line Determination; Hidden Line Elimination; Computer Graphics.

I. INTRODUCTION

Any lines visible in a view that define edges or outlines of objects are referred to as *visible lines*. Given a set of objects and a viewing point, we wish to determine which lines or regions of the objects are visible, either from the fixed point or moving viewpoint, so that we can display only the visible lines. This process is known as *visible line determination* or *hidden line elimination* [1]. In visible line determination, lines are assumed to be edges of opaque surfaces that may obscure the edges of other surfaces further from the viewpoint.

Although the statement of this fundamental idea is simple, its implementation requires significant processing power and involves large amounts of computation time. These requirements have encouraged the development of numerous properly structured visible line determination algorithms[1] such as *Robert's Algorithm*, *Apple's Algorithm*, *Haloed Lines*, *The z-Buffer Algorithm*, *List-priority Algorithms*, *The Depth-Sort Algorithm*, *The Binary Space Partition(BSP) Tree Algorithm*, *Scan-line Algorithms* etc. Recently Devai [3] proposed algorithm that runs in the $O(n^2)$ time which is worst-case optimal but limited within fixed viewpoint.

We propose a less complicated and more efficient visible edge determination algorithm which is applicable for moving viewpoint also. We consider the viewpoint moving along a particular line in 2D space. In case of moving viewpoint, for every new position of the viewpoint the algorithm is just like as the fixed viewpoint with three different cases which is discussed in this paper. At First, all the points of objects are sorted angularly with respect to the viewpoint. Then, from the nearest point of the viewpoint the algorithm proceeds with computing and updating active edge list. If infinite active edge is found, unbounded region is returned and if the infinite edge intersects another visible edge, the algorithm proceeds with the newly formed active edge with intersection point for determining visible region as output. But in case of finding the intersection point along with sorting the points of the objects every time based on the angular measurement would be inefficient and time complexity would be worse. That is why we also propose two improvements in sorting angularly and finding intersection points in case of extension of active edge in this paper.

II. BACKGROUND AND MOTIVATION

As hidden line elimination problem is a fundamental problem of Computer Graphics, many algorithms in different times have been proposed [2]. There are algorithms for hidden line elimination in literature whose running time is sensitive to the number of intersections, k, (of the projection of the segments) in the image plane, typically of the order of $O\{(n+k)\log n\}$[4, 5]. Very recently this has been improved to $O((n\log n)+k+t)$ by Goodrich [6] where t is number of intersecting polygons on the image plane. The algorithm for eliminating hidden line proposed in [9] improves the running time to $O((n\alpha(n)+k)\log n)$.

Although there has been some previous work on output-sensitive algorithms, they have been restricted to objects that are horizontal axis-parallel rectangles [7]. A commonly used technique is to process the surfaces in increasing distance from the viewer (negative x direction) so that each point needs to be tested only once for visibility, i.e., a point once detected as visible is not going to be altered later in the course of the algorithm (the same holds true for the included points). The origins of this approach can be found in [8].

However, instead of performing the visibility test for each point on the display device so that the complexity of the algorithm is also dependent on the resolution of the display device. The output of that algorithm is a graph of the final image and not a pixel by pixel description of the image. Our

techniques apply moving viewpoint in any direction very efficiently and active edge list is updated continuously to detect the visible edge. We analyze the time complexity for our proposed algorithm for determining the visible edge from both fixed and moving viewpoint and it is computed as $O(n^2)$. An algorithm based on a similar approach was proposed by Guting and Ottmann [10] to handle special cases such as aligned rectangular faces and c-oriented polygons.

Recently, a technique has been proposed to detect the visible edges in digital image in a paper [12]. A model of the resolution of a CCD camera has been built by considering geometric model and the effect of light diffusion. Knowing the ability of this sensor to reproduce some given spatial frequencies, the spectrum blocks of an image are compared with the required minimum value for a human eye to be visible. After that, visibility maps have been obtained where all visible blocks are highlighted. Finally, from this information the visible edges have been deduced by using a binarization technique.

The new algorithm proposed here implies an easier approach which proceeds in completely different manner by calculating the linear and angular distances along with the intersection point of some selected edges and points for further use. We modify our algorithm in case of moving viewpoint such that for each time the update of the sorted list requires less running time than the normal case. Thus we eliminate the complication of implementing other existing algorithms and extend our proposed algorithm for incremental visible edge determination considering better time complexity with near optimal solution.

III. Our Approach

We present an algorithm for finding visible edge from moving viewpoint for a given set of 2D objects as input. For this we consider determining visible edges from a fixed viewpoint. We calculate both the angular measurement and linear distances of each point of the given objects from the viewpoint to determine an active edge. Active edge is a finite line segment which is updated with time and indicates visible edge or part of a visible edge. After processing for each individual point of the objects along with the active edges the proposed algorithm updates the visible edge list and eventually returns the final set of visible edges or visible unbounded region. The entire approach can be presented elaborately considering the methodologies for fixed viewpoint and moving viewpoint.

A. Fixed Viewpoint

We can say that every 2D object constitutes of many points. So we consider every points of an object and determine the angular measurement from a fixed viewpoint. We also compute the linear distance of every single point from that fixed viewpoint. Based on the computation of angular measurement, we sort all other points with respect to that particular viewpoint. After determining the closest point first active edge is determined.

Let us consider one 2D object and a fixed viewpoint. Now we can say definitely that there exists at least one visible edge as from all the points of the objects there exists the nearest point which is definitely visible. If that point is not visible, there is another obstacle ahead of that viewpoint which is visible. So, considering that visible point, there is another point which is also visible from that viewpoint. Among all these visible points, any two points are selected based on the computations of the lowest angularity and shortest linear distance, an edge is formed which is active edge. After the determination of the first active edge we consider three cases for further computation of determining the final visible edge list from that fixed viewpoint.

1) First Case:

In this case, we consider if another point of the objects outside the first active edge is situated behind the active edge which will be computed from angular measurement and linear distances from the fixed viewpoint, and then we eliminate that point for further calculation. Because this point will not be visible from that particular fixed viewpoint as the angularity measurement suggests that the angle is between the two end points of active edge.

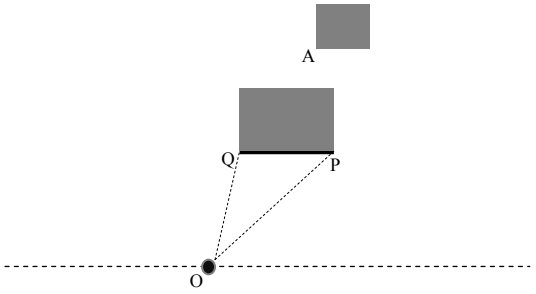

Fig. 1. In First Case, *A* is not considered for further calculation as it is behind the active edge *PQ* from the fixed viewpoint *O*.

2) Second Case:

In this case, we consider if another point of the objects outside the first active edge is situated in front of the active edge which will be computed from angular measurement and linear distances from the fixed viewpoint, then we consider it as next visible point and include that point as the end point of next active edge list.

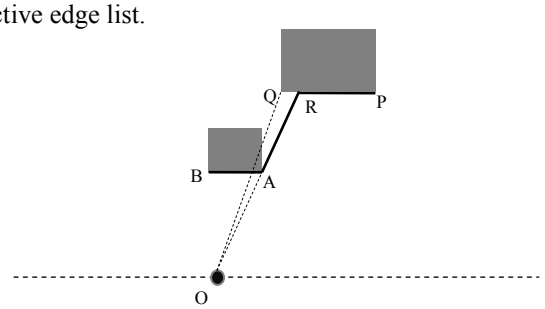

Fig. 2. In Second Case, *A* is considered for further calculation as it is in front of the active edge *PQ* from the fixed viewpoint *O*. As the extension line intersect *PQ* in the point *R*, so the newly active edge is *PR* instead of *PQ* and it is treated as visible edge and next computation is performed considering the active edge *AR*. From the first case, after inserting *AR* in the visible edge list, next active edge will be *AB*.

Now, from the newly formed active list, visible point is determined. In this case, the intersection point of the previous active edge and the new point will be considered the latest active edge and first end point and the intersection point will be the newly visible edge. According to this procedure, further computation is performed.

3) Third Case:

In this case, we consider processing the points clockwise according to the angular measurement. Now, if another point of new or the same object is situated inside the angular measurement of the previously computed visible edges, then through the fixed viewpoint and that end point of updated active edge, an infinite extension line is drawn. Thus the drawn edge is considered as updated active edge. Finally this computation returns an unbounded region. If that extension edge intersects any previously computed visible edge, the intersection point and the end point ultimately form the new active edge by the technique stated in the second case. Thus this entire computation returns an unbounded region or visible edges like previous two cases.

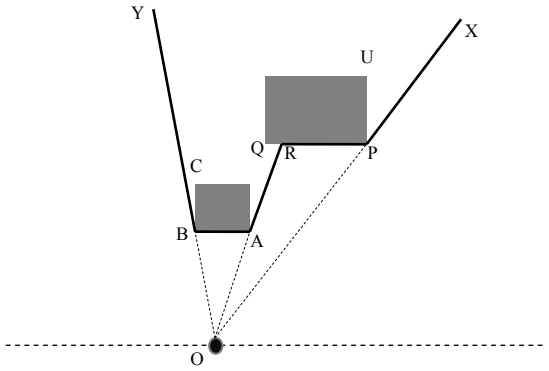

Fig. 3. In Third Case, *C* and *U* are omitted for further calculation as their angular measurements are between the range of the angular measurements of two end points of visible active edge *AB* and *PR* respectively. So, *OB* and OP are extended to infinite. But if it intersects any point of a visible edge like extended *OA* intersects *PQ* at the point *R* and formed the updated visible edge as *PR*, then the list of active edge and visible edge will be updated accordingly. Finally it returns the unbounded region *XPRABY*.

B. Moving Viewpoint

We consider the viewpoint moving along a particular line in 2D space. In case of moving viewpoint, for every new position of the viewpoint the algorithm is just like as the fixed viewpoint with three different cases. At First, all the points of objects are sorted angularly with respect to the viewpoint. Then, from the nearest point of the viewpoint the algorithm proceeds with computing and updating active edge list. If infinite active edge is found, unbounded region is returned and if the infinite edge intersects another visible edge, the algorithm proceeds with the newly formed active edge with intersection point for determining visible region as output. But in case of finding the intersection point along with sorting the points of the objects every time based on the angular measurement would be inefficient and time complexity would be worse. That is why we propose two improvements in sorting angularly and finding intersection point in case of extension of active edge.

1) Modifying Angularly Sorted Order:

We observe that angularly sorted order is changed not all the time in case of moving viewpoint. Rather, the sorted order changes when the moving viewpoint crosses the intersection point of the edge on which the viewpoint moves and the extension of joined line of any two end points of the object. We observe that angular measurement of only those two end points changes and the position of those two end points in the angularly sorted order swaps. So, we modify the algorithm in such a way that the angular span of all the edges with respect to the viewpoint is calculated as a preprocessing step. Instead of performing calculation every time, the points in that range are used only for further calculation.

2) Improving Intersection Point Determination:

We observe that in case of moving viewpoint if the intersection points are determined for all the points of the object every time, the time complexity would be worse. So we calculate the intersection of all possible points and endpoints of forming edges along with the line on which the viewpoint moves. Again, we modify the algorithm in such a way that the angular span of all the edges with respect to the viewpoint is calculated as a preprocessing step. Instead of performing calculation every time, the points in that range are used only for every time in case of intersection point determination.

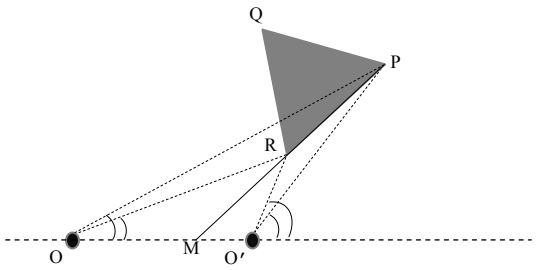

Fig. 4. The angularly sorted list in ascending order of the end points of object *PQR* is *{R, P, Q}*. This order remains unchanged until the viewpoint moves further the intersection point *M*. After crossing *M*, the angularly sorted order changes and the updated sorted order is *{P, R, Q}*. So we observe the swap of two end points *P* and *R* of the object.

IV. THE ALGORITHM

We propose Algorithm1, *Determine_Visible_Edge*, shows the pseudocode for determining the visible edge from moving viewpoint. The input to the algorithm is a set of vertices of polygonal objects. The nearest point from the viewpoint is denoted as P_0 and the point which is used for computing active edge is denoted as P_i. For the three cases stated in the previous section, *if-else* statement has been used and loop is used for computing all the points in the sorted list. The output of the algorithm is a set of visible edge or an unbounded visible region.

Algorithm1: DETERMINE_VISIBLE_EDGE(point P_0)

Input : A set of vertices of polygonal objects

Output : A set of visible edge or an unbounded visible region

1.1: *Sort angularly all points around* P_0

1.2: *Determine the initial sorted list*

1.3: **for** *all points* P_i *in the sorted list*

1.4: **if** P_i *is behind the active edge* **then**

1.5: *Update nothing*

1.6: **else if** P_i *is in front of the active edge* **then**

1.7: *Update visible edge list and active edge accordingly*

1.8: **else if** P_i *is in the other end of active edge* **then**

1.9: *Update visible edge list and active edge accordingly*

1.10: **end if**

1.11: **end for**

V. TIME COMPLEXITY ANALYSIS

Time complexity is an important performance metric for algorithm. We analyze the time complexity for our proposed algorithm for determining the visible edge from both fixed and moving viewpoint and it is computed as $O(n^2)$. At first, we compute the linear distance of all the points of n objects from the viewpoint as the distance of any point from another point can be computed with the time complexity of $O(1)$. So, for n number of points of n objects the complexity is $O(n)$. Then, we compute the complexity of angular sort which is $O(n\log n)$. As, total number of angle measured is $O(n)$ and the best time complexity for any sorting algorithm [11] is $O(n\log n)$. So, time complexity for angular sort is $O(n\log n)$ in best case. After that, we computed the complexity for determining initial active edge which is $O(n)$ for n number of points. Now, the loop is executed for n times and for the first and second case time complexity is $O(1)$. But for the third case it is $O(n)$. So, the total cost of the algorithm is $O(n^2)$ in the worst case. But we hope this time complexity can be reduced practically because we optimize the algorithm especially for sorting and the third case in the loop execution.

VI. IMPLEMENTATION DETAILS

We have implemented the proposed algorithm using *C++* Language and *OpenGL API* (Application Programming Interface) for rendering 2D graphics. We have used *vector* as data structure for the quick and easy allocation of the necessary memory needed. We have taken the input in the form of rectangles from separate text file. The output shown for the different position of moving viewpoint in a 2D space with rectangular obstacles is the unbounded visible region with drawn lines.

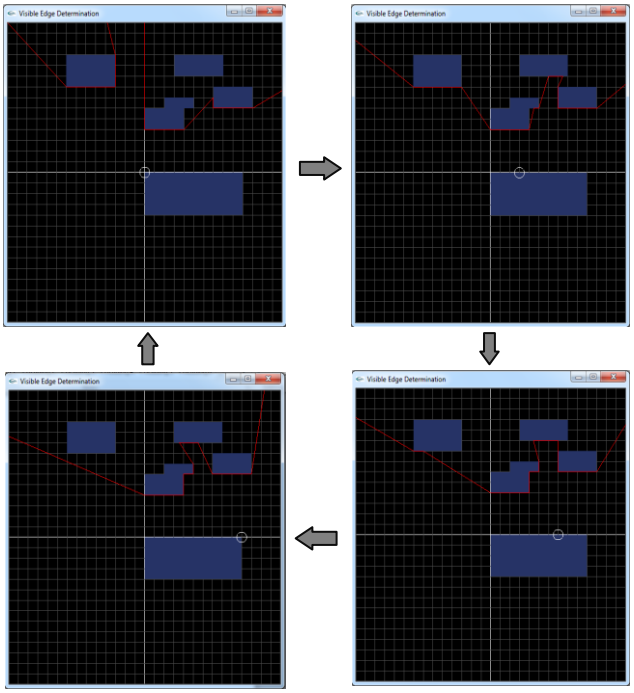

Fig. 5. The *Output* window of the implementation of our proposed algorithm for determining visible edge or visible unbounded region from moving viewpoint in different position in a 2D space with rectangular obstacles.

VII. PERFORMANCE ANALYSIS

We have analyzed our proposed algorithm of visible edge determination from moving viewpoint for 10 runs with different number of obstacles. The position and size of the obstacles are generated randomly such that they do not intersect with each other. We have measured running time of the proposed algorithm for different number of obstacles. The running times of the initial run of the algorithm around the first point in each setup are listed in Table I.

TABLE I. PERFORMANCE ANALYSIS OF PROPOSED ALGORITHM

No.	Number of Obstacles	Running Time(msecs)
1.	1000	22
2.	2000	42
3.	3000	70
4.	4000	96
5.	5000	130
6.	6000	161
7.	7000	200
8.	8000	241
9.	9000	298
10.	10000	365

In the following graph we have plotted running time against number of obstacles to illustrate the performance analysis of our proposed algorithm. This clearly proves our claim that the proposed algorithm performs far better than its worst case time complexity, which is $O(n^2)$.

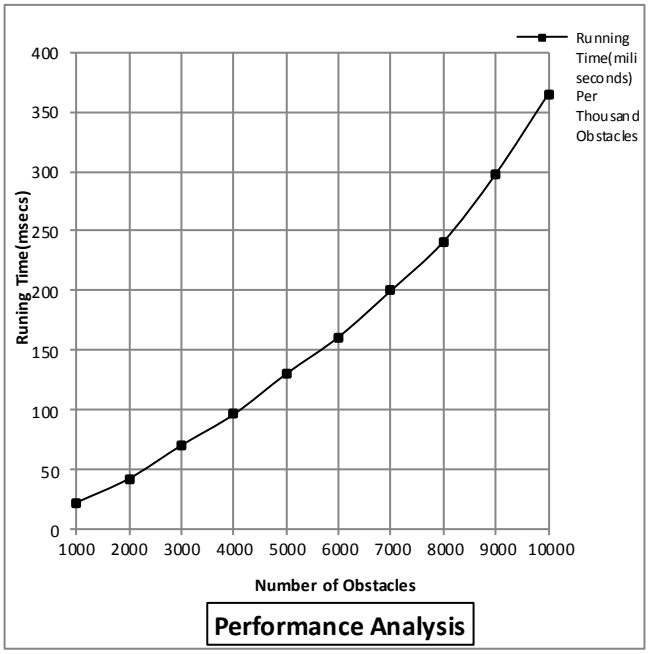

Fig. 6. Performance analysis of proposed algorithm for visible edge determination from moving viewpoint

VIII. CONCLUSION

In this paper, we have presented an easier and efficient algorithm for visible edge determination from moving viewpoint. We have computed and provided explanation of the time complexity for the proposed algorithm which is $O(n^2)$ in the worst case. We have also implemented the algorithm and showed the expected output which indicates better performance and better running time of the algorithm for determining the visible edge from moving viewpoint in 2D-space with several obstacles. Although, this incremental algorithm is applied in 2D-space, further modification of this algorithm is expected to determine the visible surface from moving perspective in 3D-space with better time complexity as shown in this easier approach.

ACKNOWLEDGMENT

The authors would like to gratefully acknowledge the concepts of James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes through their book entitled *Computer Graphics, Principles and Practice*. Besides the authors acknowledge the contribution and guidelines provided in the research works of all the authors of the corresponding papers and scientific articles regarding visible edge determination.

REFERENCES

[1] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, "Computer Graphics: Principles & Practice," 2nd ed., Addison-Wesley, Boston, 1994.

[2] R. F. Sproull,, I. E.Sutherland, , R. A. Schumacker, "A characterization of ten hidden-surface algorithms," Computing Surveys 6 (1), pp. 1-25, 1974.

[3] F. Devai, "Quadratic bounds for hidden line elimination," ACM Transactions on Graphics, pp. 19-28, 1986.

[4] O. Nurmi, "A fast line-sweep algorithm for hidden line elimination," BIT 25, pp. 466-472, 1985.

[5] A. Schmitt, "Time and space bounds for hidden line and hidden surface elimination algorithms," In Proceedings of the EUROGRAPHICS, pp. 43-56, 1981.

[6] M. T. Goodrich, "A polygonal approach to hidden-line elimination," GVGIP: Graphical Models and Image Processing 54 (l), pp. 1-12, 1992.

[7] M. Bern, "Hidden surface removal for rectangles," In Proceedings of 4th ACM Symposium on Computational Geometry, pp. 183-192, 1988.

[8] T. J. Wright, "A two-space solution to the hidden line problem for plotting functions of two variables," IEEE Trans. on Comput. c-32 (l), pp. 28-33, 1972.

[9] F. Prepata, J. Vitter, "A simplified technique for hidden-line elimination in terrains," In the Proceedings of STACS, February 1992.

[10] R. H. Guting, T. Ottmann, "New Algorithms for Special Cases of the Hidden Line Elimination Problem," In Proceedings of the 2nd Symposium of Theoretical Aspects of Computer Science, pp. 161-172, 1985.

[11] K. Mehlhorn, "Data Structures and Algorithms, Vol. 1: Sorting and Searching, Vol. 3," Multidimensional Searching and Computational Geometry, Springer Publishing Company, Springer, 1984.

[12] N. Hautiere, D. Aubert, "Visible Edges Thresholding: a HVS based Approach," In Proceedings of the 18th International Conference on Pattern Recognition, 2006. ICPR 2006, Vol. 2, pp. 155-158.