Khulna University Studies Special Issue ICSTEM4IR: 244-258, 2022

ORIGINAL ARTICLE **SCIENCE & ENGINEERING**

PSEUDO-TRIANGULATION OF MONOTONE POLYGON USING SWEEP LINE ALGORITHM

S.M. Azoad Ahnaf*, Md. Saiful Islam, Md. Anisur Rahman

Computer Science and Engineering Discipline, Khulna University, Khulna, Bangladesh-9208

KUS: ICSTEM4IR-22/0039

Manuscript submitted: July 17, 2022 Accepted: September 26, 2022

Abstract

In the field of computational geometry, pseudo-triangulation of a polygon is an interesting topic. Breaking a polygon into pseudo-triangles reduces the calculation cost and increases computational power. The sweep line algorithm also known as the plane sweep algorithm uses an imaginary line or a plain surface on the Euclidean plain to perform various computations. Finding line intersection with sweep line was a revolutionary outcome. As polygons are the combination of edges and vertices, we tried to implement the concept of a sweep line to create pseudo-triangles inside the polygon. Among the vast triangulation algorithms, a few pseudo-triangulation algorithms are available. Using the sweep line algorithm, we have been able to pseudo-triangulate a monotone polygon which uses the time complexity of $O(n^3)$ to execute. As complex polygons can often be divided into a monotone polygon in just $O(log\ n)$ so it is possible to chain with our sweep line-based technique, any complex polygon can be pseudo-triangulated in $O(n^3)$.

Keywords: Polygon, triangulation, pseudo-triangle, sweep line, edge, vertex

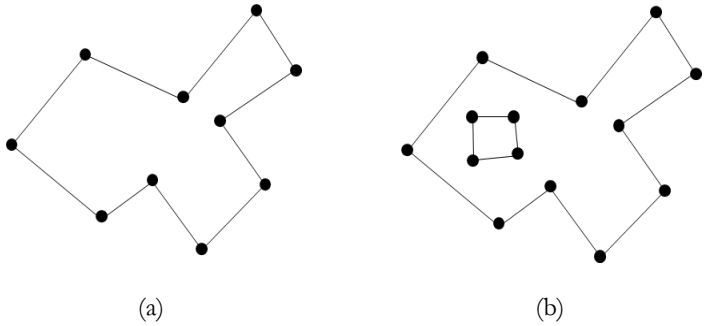
Introduction

Any two-dimensional shape, bounded or framed by straight lines, that does not have any curved or bent line is known as a polygon. The line segments of a polygon are called edges and the focus areas where two edges meet are called vertices. A polygon can be simple or complex depending on the shape. Usually, complex polygons have self-intersections or holes inside them. Both types of polygons can be found in (Figure 1).

As it is easier to work efficiently on a simple polygon, complex polygons are broken down into simpler shapes to reduce operational complexity. One of the simpler polygonal shapes is a monotone polygon. In computational geometry, a polygon is called a monotone concerning a straight line if and only if all possible lines perpendicular to that straight line intersect the polygon boundary at most two times (Figure 2). Depending on the axis of the line, a monotone polygon is further divided into x-monotone and y-monotone polygons.

The common method for decomposing a polygon is triangulation which reduces the polygonal shape into simple triangles as the simplest form of a polygon is a triangle. Thus, it is the first step of many advanced

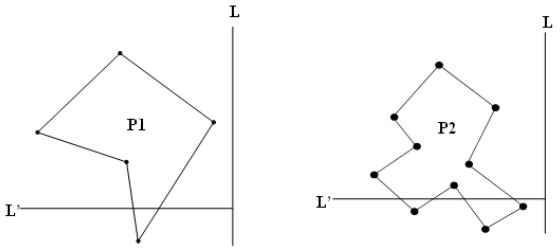
*Corresponding author: <azoadahnaf@gmail.com>
DOI: https://doi.org/10.53808/KUS.2022.ICSTEM4IR.0039-se



(a) (b)

Figure 1. (a) Simple Polygon; (b) Complex Polygon with a Hole.

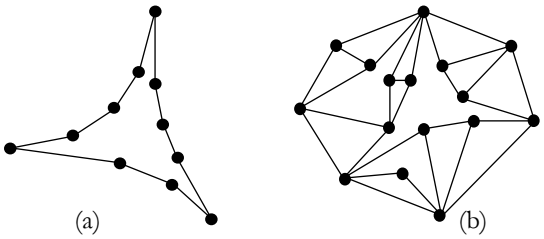
algorithms. The aspect of triangulation of polygon can be classified into three groups. They are based on - ear cutting, Delaunay triangulation, and using Steiner points. There are algorithms from each group available that have achieved the complexity of $O(n \log n)$ (Lamot & Zalik, 1999, 2000).



(a) (b)

Figure 2. (a) Monotone Polygon; (b) Non-monotone Polygon.

Polygons can also be decomposed using the pseudo-triangulation method. A polygon having exactly three convex vertices (whose internal angle is less than π degree) is called a pseudo-triangle (Figure 3). Any general triangle can also be called a pseudo-triangle as triangles have exactly three convex vertices. A pseudo-triangulation is a face-to-face tiling of a planner region into pseudo triangles (Rote et al., 2008). Pseudo-triangulation (Figure 3) is a generalization of triangulation, which means, that any triangulation is pseudo-triangulation, but not all pseudo-triangulation is triangulation.



(a) (b)

Figure 3. (a) Pseudo-triangle; (b) Pseudo-triangulation of a point set.

In computational geometry, a sweep-line or plane sweep algorithm is considered an arithmetic paradigm or algorithm implementation technique that uses an imaginary line to sweep across an object or

surface to solve several problems in Euclidean space. Using modified data structures, it stores and calculates the state of the points it travels over. Those information are later used to solve specific problems. This technique was adapted from the scanline of computer graphics and was introduced first in computational geometry to find line segment intersections (Shamos & Hoey, 1975).

Background Study

There are many existing algorithms available for polygon triangulation among those some have linear complexity (Chwa, 1987; Garey et al., 1978; Hertel & Mehlhorn, 1983; Tarjan & Van Wyk, 1986). Monotone polygons can also be triangulated in linear time (G. T. Toussaint, 1984). Most popular polygon triangulation algorithms are ear cutting (Hai et al., 2022; Lamot & Zalik, 2000; Livesu, 2020; Livesu et al., 2021; Mandot, 2013; Mašović et al., 2018; G. Toussaint, 1991), Delaunay triangulation (Bern et al., 1995; Lewis & Robinson, 1978; Shen et al., 2021), and Steiner points (Aronov et al., 2012; Eppstein, 1994; Erten & Üngör, 2009). There are even some works where a combination of different methods is utilized (Hadi et al., 2021; Üngör, 2004; Žalik, 2005).

As we can see, the number of works done on polygon triangulation is quite a bit, on the contrary, a limited number of works have been done on pseudo-triangulation. Though, pseudo-triangulation has many uses – art gallery problem, ray shooting, visibility complex problem, convex obstacles, the Carpenter's rule problem, and single vertex origami are a few of those (Rote et al., 2008). Several algorithms have been developed for solving art gallery problems by having polygons with holes (Alipour, 2016; Hoffmann et al., 1991; Iwerks & Mitchell, 2012; Speckmann & Tóth, 2005) which includes pseudo-triangulation.

Using the incremental insertion method, a pseudo-triangulation algorithm was proposed (Kolingerová et al., 2011). Incremental insertion is mostly used in Delaunay triangulation. To operate, not all the points need to be known and sorted beforehand. From the given point set, a triangle is made randomly by choosing any three non-linear points. For all the other points left, locating the pseudo-triangle which contains the point to be inserted is done next. Then inserting the given point two pseudo-triangles are constructed.

Majid proposed a method that can perform the construction of a polygon and also parallelly perform pseudo-triangulation from a given point set (Majid, 2012). From the known point set, this method first calculates the lowest coordinate values, then find out angle among themselves and sort them. Then by computing the convex hull of the polygon, several sub-polygons are formed and from there pseudo-triangulation is done.

A visibility graph is a graph of inter-visible locations for a set of points. For a polygon, the visibility graph represents the vertices as usual but the edges are expressed as a pair of vertices that is visible with respect to a vertex. Using this technique of visibility graph a polygon pseudo-triangulation algorithm was developed (Emamy, 2015). At first, the visibility information of the given polygon is extracted. Then from the visibility graph, blocking vertices are determined. For each vertex, adjacent vertices are kept in a hash table and then reflex chain and corner vertices are explored. From the corner vertices and reflex chain information, pseudo-triangles are generated.

The previously mentioned methods are pseudo-triangulation of a simple polygon that does not have self-intersection or holes. For polygons with holes an augmented ear-cutting method for pseudo-triangulation was proposed (Roy & Akter, 2018). This method is divided into two parts. Calculation of the visibility information is the first part of this work, and using the visibility information and augmented ear cutting method to obtain pseudo-triangulation is the second part of this work. This method works as follows- first, the boundary and hole vertices are taken as input. After that, all the visibility information of the vertices is extracted which leads to the calculation of the mutually visible vertices which can be connected with a straight line that lies inside of a polygon. Then, checking the angle of the vertices if $angle \geq 180°$ then that vertex is added to a reflex chain for all the other cases that vertex is treated as the last point of the reflex chain. For a

vertex whose neighbor vertices have $angle < 180°$ finding a common visible vertex and connecting that gives the pseudo-triangulation.

Materials and Methods

Simple polygon triangulation using sweep-line is a relatively common approach yet this has not been yet introduced in pseudo-triangulation as far as our concern. So, we have proposed a pseudo-triangulation algorithm for monotone polygons.

Materials

As there is no dataset available consisting of monotone polygons, so, we had to create our own set of monotone polygons to experiment on them. In the process we created around fifty monotone polygons.
The algorithm was implemented using python and the IDE used for the work was PyCharm.

Brief discussion of the methods

Though our proposed algorithm works for any kind monotone polygon, but in this study, we have used the y-monotone polygons only. The devised system is able to find pseudo-triangulation of the given y-monotone polygon using the sweep-line paradigm.

As an input a set of points is given which denotes the vertices of the polygon that we want to pseudo-triangulate. The points are given in clockwise order and as we want to work with monotone polygons, there is no hole or self-intersection in it. From the given points we calculate the necessary information and sort the points in descending order with respect to the x-coordinate. The necessary information includes the next and previous points of a given point, and the internal angle of the points. After that we find out the left and the right chains of the polygon formed with the given point set. Then using an imaginary sweep line, our algorithm traverses in descending order over the points and checks the conditions to perform pseudo-triangulation or normal triangulation. In Figure 4 we show the pictorial view of the major steps of the proposed algorithm. Later we present a working example and after that the pseudo code of the algorithm is given in Algorithm 1.

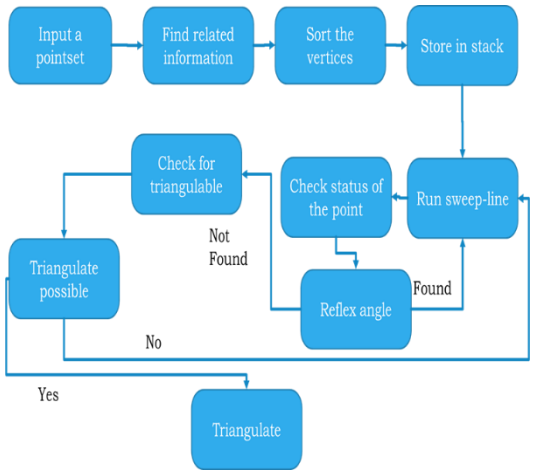


Figure 4. Major steps of the proposed algorithm.

Table 1. A monotone polygon input for the proposed algorithm

X	Y
4	9
5.5	6
5	3
6	0
3.5	1
2	3
3.5	6
3	7.5

Input for a monotone polygon is given as a set of pairs (first element for x-coordinate and second element for y-coordinate) in Table 1. Then the input points are enumerated and some related information are calculated and stored. The information is, which point is next to the current point, and which point is previous to the current point. For the use of sweep-line, the points are sorted with respect to the value of Y in decreasing order, and for two points having the same value, one is chosen randomly (Table 2).

With all the information, we can get a clear picture of a monotone polygon constructed (Figure 5). After that from the values, we find out 0 is the top and 3 is the bottom point. Now we calculate the left and right chains of the polygon.

The left chain is $0 \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 3$
The right chain is $0 \rightarrow 1 \rightarrow 2 \rightarrow 3$

Table 2. Sorted points of the monotone polygons

Self	Point: (X, Y)	Next	Previous
0	4,9	1	7
7	3,7.5	0	6
1	5.5,6	2	0
6	3.5,6	7	5
2	5,3	3	1
5	2,3	6	4
4	3.5,1	5	3
3	6,0	4	2

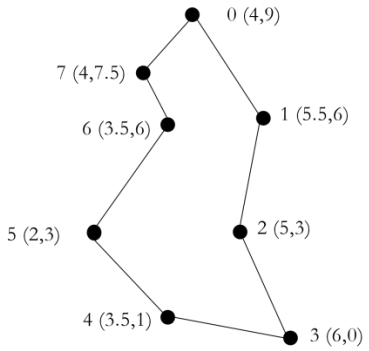


Figure 5. Input polygon with information.

After doing all the necessary preprocessing now the actual line sweep begins with an imaginary line sweeping from the top towards the bottom. When the line reaches 0, it knows its previous and next points, with those it calculates the internal angle then this angle is also stored with the related points. Whenever a reflex angle is found, that is stored in a stack. Also, the encountered point is checked to see in which chain it lies. We then stored the angle of each point in the related point information.

The internal angle between $p1p2$ line and $p2p3$ line can be calculated with the Equation-

$$radian = \cos^{-1}\left(\frac{(a+b-c)}{2\sqrt{a}\sqrt{b}}\right) \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots(1)$$

here the points are $p1(x1, y1)$, $p2(x2, y2)$ and $p3(x3, y3)$ and,

$$a = (x1 - x2)^2 + (y1 - y2)^2$$
$$b = (x2 - x3)^2 + (y2 - y3)^2$$
$$c = (x3 - x1)^2 + (y3 - y1)^2$$

For pseudo-triangulation, we store the top two points, with their angles first, while the sweep-line does not encounter the bottom-most point, it continues to sweep downwards. If any point is found at the left of its previous and next point, which means it is not a reflex angle. We encounter the first reflex angle at 6 but before that, the sweep line confronts three points that do not have a reflex angle so triangulation is done.

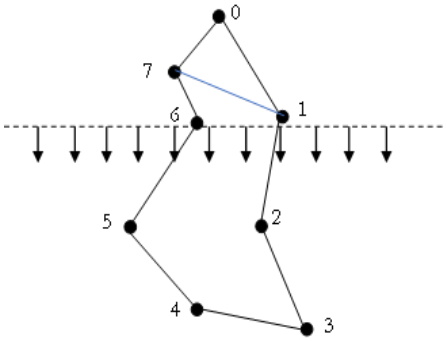


Figure 6. After the first triangulation sweep line encounters the first reflex angle.

The point for which reflex angle is obtained is kept in a stack with unprocessed points. When the stack reaches four points, it tries to perform triangulation to reduce the number of points in the stack. So, when the sweep line reached 2 which is another reflex angle but on a different chain, it checks possible triangulations and performs computation it achieves a triangulation, but in this case, it provides us with a pseudo-triangulation. In the meantime, the stack is now updated with current points. Moving down it finds another possible triangulation and performs it (Figure 7).

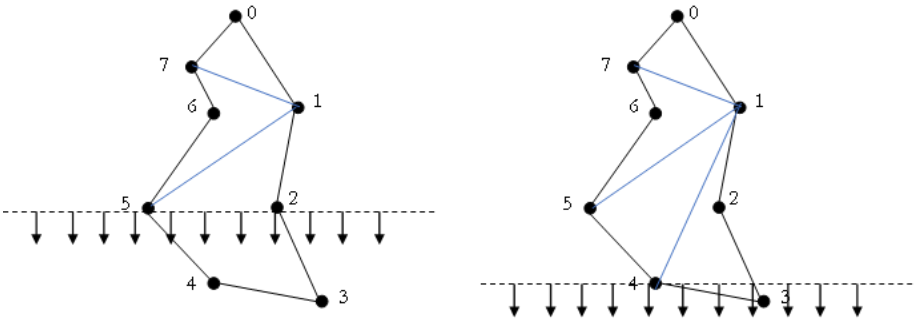
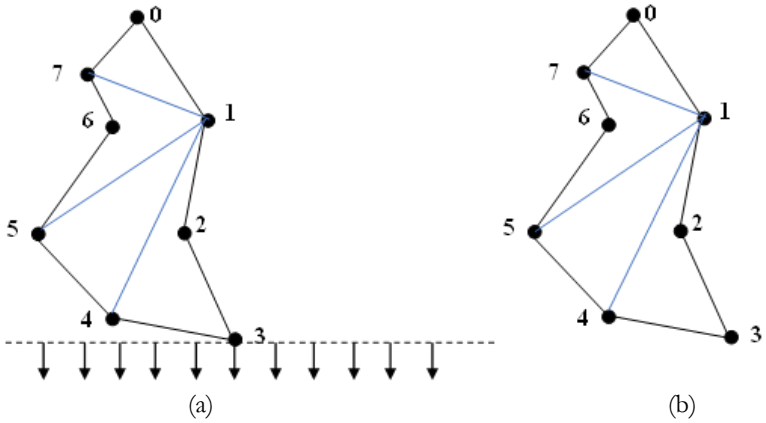


Figure 7. Steps of sweep-line traversing and completing triangulation.

When the sweep line reaches the endpoint and all the calculation and computation is done, we get the final outcome, a pseudo-triangulation of the given monotone polygon (Figure 8).



(a) (b)

Figure 8. (a) Sweep-line reaching the bottom-most point; (b) Final output polygon with pseudo-triangulations.

Proposed Algorithm

For performing the abovementioned operations, we needed several functionalities to check and compute. Several functions were used, devised, and modified to serve our needs. For example- we needed to find if a point is on the left side of a line, so we used $function\ is\_left\_of(current_{point}, start_{point}, end_{point})$; for returning the chain of a point $function\ is\_in\_chain(point, flag = 1)$; $function\ angle(p1, p2, p3)$ return internal angle of a vertex; to find the bottom point of a chain $function\ bottom\_of\_chain(chain)$ and finally we have the function that check if a line is inside a polygon is $function\ inside\_polygon(p1, p2)$.

The algorithm for the proposed system is –

Algorithm 1: Pseudo code of Pseudo-triangulation

1. *Input: A set of points as the convex hull of the y-monotone polygon in clockwise order.*
2. *Store information of the points – previous, next point of a point*
3. *Sort points according to the y-direction*
4. *Initialize stack with top two points*
5. *Initialize diagonals as an empty list*
6. *for* $i \leftarrow 3$ *to n, do*
7. *if the point is the bottom point*
8. *break*
9. *if the interior angle of the point is* $\geq 180°$
10. *add the current point to the stack*
11. *continue*
12. *for* $j \leftarrow length\_of\_stack\ to\ 2$*, do*
13. *if triangulation is possible*
14. *Add (stack.top, stack[j]) to diagonals*
15. *update stack*
16. *else*
17. *continue*
18. *Convert the diagonals to pseudo-triangles*
19. *Output: Pseudo-triangulated polygon*

Result Analysis and Discussion

The proposed system was able to perform pseudo-triangulation on the y-monotone polygon. Here to reduce redundancy, we have used polygon to express the y-monotone polygon. Some of the attained results are given in this section. Our system was able to obtain pseudo-triangulation for a given polygon. When there exist no pseudo-triangles in the polygon, our system simply finds out the triangles instead of pseudo-triangles. Also, for the cases, when any one of the two possible triangulation was available, our algorithm picks the one it

DOI: https://doi.org/10.53808/KUS.2022.ICSTEM4IR.0039-se

encounters first computationally. For a four-vertices concave monotone polygon, as no pseudo-triangle is possible, the outcome is a normal triangulation (Figure 9).

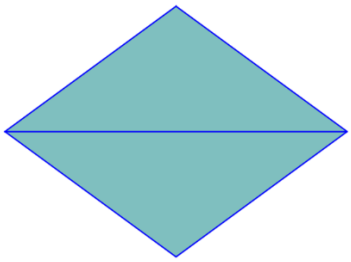


Figure 9. Pseudo-triangulated form of a 4-vertices concave monotone polygon

For a five-vertices polygon that has one reflect angle, one pseudo-triangulation is possible but that can be at two different places, our algorithm obtained the one it encountered first (Figure 10).

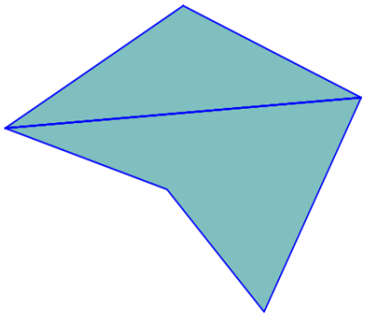


Figure 10. Pseudo-triangulated form of a five-vertices with one reflex angle polygon.

Now if we move onto polygon with having more reflex angles, we can see depending on the position of reflex angles, the number of pseudo-triangles varies. For a polygon having thirteen vertices and seven reflex angles, a total of four triangles among whose three are pseudo-triangles is possible (Figure 11). The proposed system finds out the minimum number of diagonal possible to obtain pseudo-triangulation.

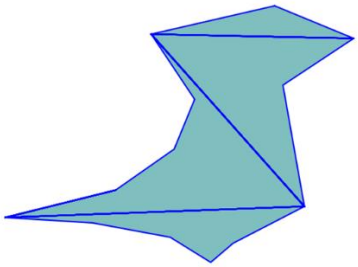


Figure 11. Pseudo-triangulation of a 13-vertices polygon among which 8 have reflex angles.

Moreover, if our system encounters pseudo-triangles, it can escape the triangulation, though it has to go through the full process (Figure 12).

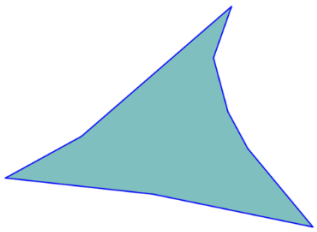


Figure 12. Triangulation is escaped as the polygon is a pseudo-triangle.

Complexity analysis

The time complexity of our proposed system is $O(n^3)$. It can be calculated as –

Table 3. The complexity of the proposed algorithm.

Statement	Complexity
Algorithm: Pseudo-triangulation	
1. *Input: A set of points as the convex hull of the y-monotone polygon in clockwise order.*	
2. *Store information of the points – previous, next point of a point*	n
3. *Sort points according to the y-direction*	$nlogn$
4. *Initialize stack with top two points*	n
5. *Initialize diagonals as an empty list*	n
6. *for* $i \leftarrow 3$ *to n, do*	n
7. *if the point is the bottom point*	$n \times 1 = n$
8. *break*	0
9. *if the interior angle of the point is* $\geq 180°$	$n \times 1 = n$
10. *add the current point to the stack*	$n \times 1 = n$
11. *continue*	$n \times 1 = n$
12. *for* $j \leftarrow length\_of\_stack$ *to* 2*, do*	$n \times n = n^2$
13. *if triangulation is possible*	$n^2 \times n = n^3$
14. *Add (stack.top, stack[j]) to diagonals*	n^3
15. *update stack*	n^3
16. *else*	n^3
17. *continue*	n^3
18. *Convert the diagonals to pseudo-triangles*	1
19. *Output: Pseudo-triangulated polygon*	
Total	$5n^3 + n^2 + 8n + nlogn + 1$
Complexity	$O(n^3)$

There is a relationship between runtime and the number of vertices, as an increase of vertices means more computation and thus needs more runtime (Figure 13). But no linear relationship is found between the number of reflex angles and vertices as we have seen from the previous figures, the positioning of the reflex angles matters in finding pseudo-triangles.

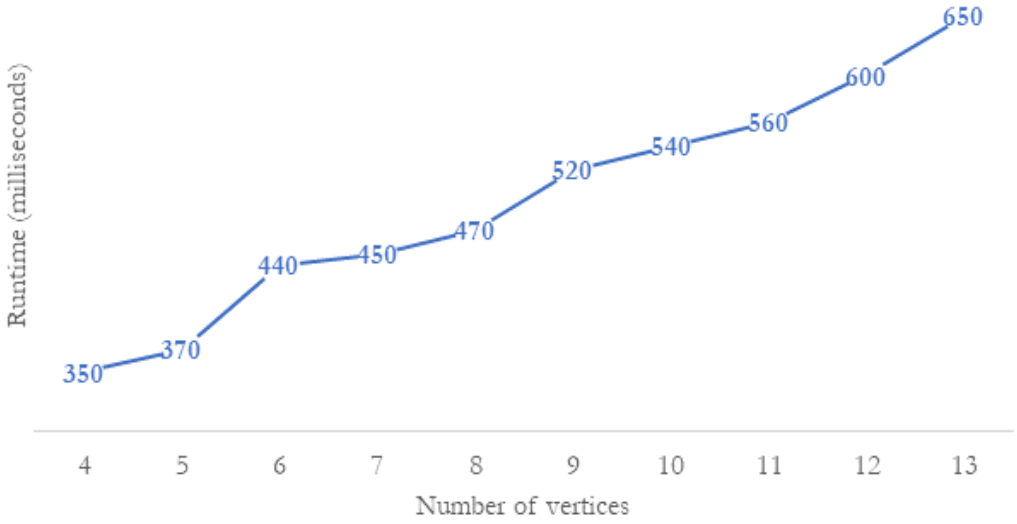


Figure 13. Relation between the number of vertices and runtime.

Comparison with other methods

For a four-vertices with one reflex angle polygon previous method gives two triangulation (Majid, 2012). But our method ignores triangulation as it is already a pseudo-triangle. Though the work of (Majid, 2012) has the time complexity of $O(n^3)$ which is exactly same as our proposed system, but our system in lieu of the higher complexity provides with better pseudo-triangulation (Figure 14**)**.

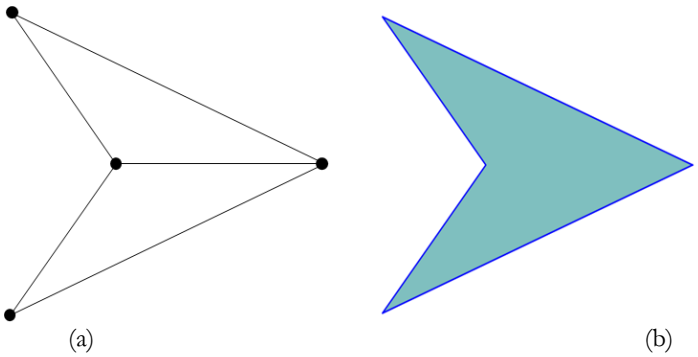
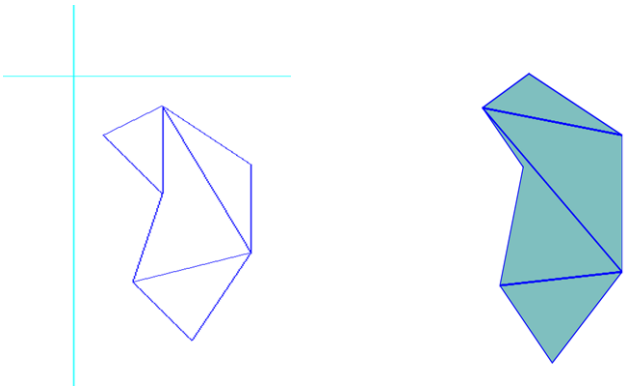


Figure 14. (a) Triangulation of (Majid, 2012); (b) Our proposed method pseudo-triangulation.

For seven vertices output of the method by Roy and Akter (Roy & Akter, 2018), though the triangulation outcome is different but the number of triangulation is the same as ours (Figure 15).



(a) (b)

Figure 15. Output comparison between our (b) method and (a) (Roy & Akter, 2018).

Figure 16 shows the comparison between the proposed algorithm and the approximation algorithm (Majid, 2012). We can see here, that the number of diagonals obtained for a polygon with a specific number of vertices is much lower in our proposed. As lesser number of triangles or pseudo-triangles indicates to perform less computation, so it can be said that working with polygons pseudo-triangulated using our method will be less complex to analyze.

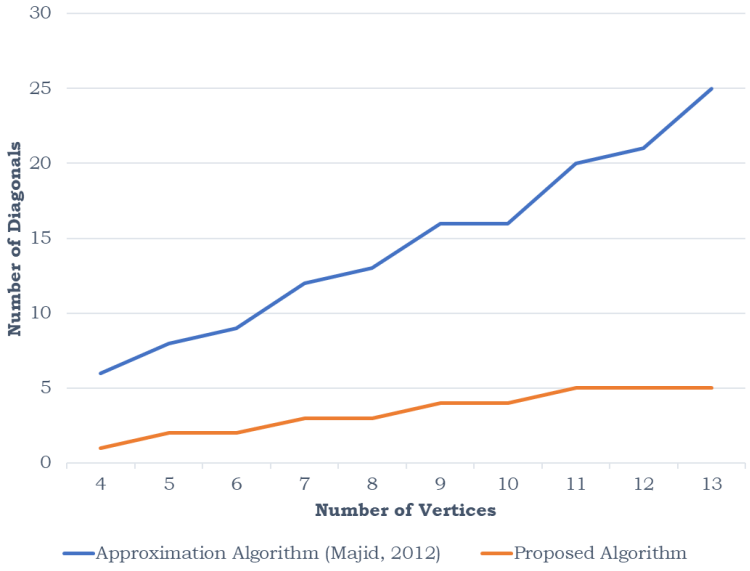


Figure 16. Comparison of the number of diagonals between proposed algorithm and approximation algorithm (Majid, 2012)

If we observe the relation between diagonals and vertices in our work and (Roy & Shapla, 2018) we find out that the numbers remain almost same (Figure 17**)**. As their system is able to pseudo-triangulate even for complex polygons with holes, thus our program is able to work faster.

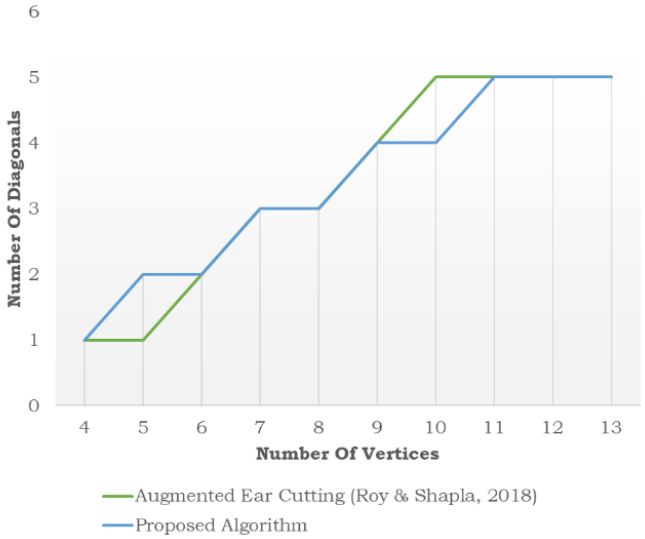


Figure 17. Comparison of the number of diagonals between proposed algorithm and approximation algorithm (Roy & Shapla, 2018).

Moreover, for the same polygon from Figure 14 the work of Emamy (Emamy, 2015) generated the same outcome as ours but there were no more examples to compare with.

Discussion

For pseudo-triangulation of a complex polygon, pseudo-triangulation of the monotone polygon is the first step. Though it can theoretically be done without breaking a polygon into monotone-polygons, but the computations become much more complex. On the other hand, complex polygons can easily be decomposed into monotone polygons. This decreases the complexity of computation and analysis. Our proposed method works on par if not better in comparison to the previously devised methods. Even monotone polygons can become complex to a degree and our system can handle those with ease. Triangulation outcome is better than other methods also.

Limitation of our proposed algorithm is that for some rare cases, it fails to triangulate properly. Mostly, when two data points of a chain lie on the same y-coordinate it cannot distinguish and draw diagonals properly and gives overlapped diagonals. Though having some limitations, our proposed method can pseudo-triangulate with more confidence in comparison to the methods available.

Conclusion and Future works

Triangulation is a very popular topic in various fields of computer science as well as mathematics. Pseudo-triangle and pseudo-triangulation are relatively new emerging topics. In this paper, an algorithm has been proposed for the pseudo-triangulation of the y-monotone polygon using the sweep-line approach. We have done some preprocessing of the input polygon to do pseudo-triangulation on it. We have tested the system with several inputs and observed the result.

In the future, addressing the limitations of this work can provide more feasible solutions. Either it can be paired with breaking a polygon into a monotone polygon to pseudo-triangulate a complex polygon or it can be further developed to perform pseudo-triangulation on a complex polygon without the need of breaking it into a monotone polygon.

References

Alipour, S. (2016). *On guarding polygons with holes*. *23*, 1–5.

Aronov, B., Asano, T., & Funke, S. (2012). Optimal Triangulations of Points and Segments with Steiner Points. *Http://Dx.Doi.Org/10.1142/S0218195910003219*, *20*(1), 89–104. https://doi.org/10.1142/S0218195910003219

Bern, M., Dobkin, D., & Eppstein, D. (1995). *Triangulating Polygons Without Large Angles*. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.5231

Chwa, K. Y. (1987). A New Triangulation-Linear Class of Simple Polygons. *International Journal of Computer Mathematics*, *22*(2), 135–147. https://doi.org/10.1080/00207168708803587

Emamy, Z. S. (2015). Pseudo-triangulating a simple polygon from its visibility graph. *2015 5th International Conference on Computer and Knowledge Engineering, ICCKE 2015*, 106–111. https://doi.org/10.1109/ICCKE.2015.7365868

Eppstein, D. (1994). Approximating the minimum weight steiner triangulation. *Discrete & Computational Geometry 1994 11:2*, *11*(2), 163–191. https://doi.org/10.1007/BF02574002

Erten, H., & Üngör, A. (2009). Quality Triangulations with Locally Optimal Steiner Points. *Http://Dx.Doi.Org/10.1137/080716748*, *31*(3), 2103–2130. https://doi.org/10.1137/080716748

Garey, M. R., Johnson, D. S., Preparata, F. P., & Tarjan, R. E. (1978). Triangulating a simple polygon. *Information Processing Letters*, *7*(4), 175–179. https://doi.org/10.1016/0020-0190(78)90062-5

Hadi, N. binti A., Farhani, A., & Dahalan, W. M. (2021). An Improved Simple Sweep Line Algorithm for Delaunay Refinement Triangulation. *Advanced Structured Materials*, *147*, 263–270. https://doi.org/10.1007/978-3-030-67307-9_23

Hai, Y., Guo, Y., & Dong, M. (2022). A CAE-oriented mesh hole-filling algorithm focusing on geometry and quality. *Engineering Computations*, *ahead-of-print*(ahead-of-print). https://doi.org/10.1108/EC-07-2021-0411

Hertel, S., & Mehlhorn, K. (1983). Fast triangulation of simple polygons. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *158 LNCS*, 207–218. https://doi.org/10.1007/3-540-12689-9_105

Hoffmann, F., Kaufmann, M., & Kriegel, K. (1991). The art gallery theorem for polygons with holes. *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, 39–48. https://doi.org/10.1109/SFCS.1991.185346

Iwerks, J., & Mitchell, J. S. B. (2012). The art gallery theorem for simple polygons in terms of the number of reflex and convex vertices. *Information Processing Letters*, *112*(20), 778–782. https://doi.org/10.1016/J.IPL.2012.07.005

Kolingerová, I., Trčka, J., & Hobza, L. (2011). Construction of pseudo-triangulation by incremental insertion. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6784 LNCS*(PART 3), 30–43. https://doi.org/10.1007/978-3-642-21931-3_3

Lamot, M., & Zalik, B. (1999). An overview of triangulation algorithms for simple polygons. *Proceedings of the International Conference on Information Visualisation*, *1999-Janua*, 153–158. https://doi.org/10.1109/

IV.1999.781552

Lamot, M., & Zalik, B. (2000). *A Contribution to Triangulation Algorithms for Simple Polygons*. 319–331.

Lewis, B. A., & Robinson, J. S. (1978). Triangulation of planar regions with applications. *The Computer Journal*, *21*(4), 324–332. https://doi.org/10.1093/COMJNL/21.4.324

Livesu, M. (2020). *Mapping Surfaces with Earcut*. http://arxiv.org/abs/2012.08233

Livesu, M., Cherchi, G., Scateni, R., & Attene, M. (2021). Deterministic Linear Time Constrained Triangulation using Simplified Earcut. *IEEE Transactions on Visualization and Computer Graphics*, *14*(8), 1–7. https://doi.org/10.1109/TVCG.2021.3070046

Majid, R. N. (2012). *Pseudo Triangulation of Point Set Using Polygon Construction*. 24–26.

Mandot, M. (2013). *Application of the Triangulation of Polygon*. *3*(11), 1188–1209.

Mašović, S. H., Elshaarawy, I. A., Stanimirović, P. S., & Krtolica, P. V. (2018). Orbiting triangle method for convex polygon triangulation. *Applicable Analysis and Discrete Mathematics*, *12*(2), 439–454. https://doi.org/10.2298/AADM170829013M

Rote, G., Santos, F., & Streinu, I. (2008). *Pseudo-triangulations—a survey*. *0000*, 343–410. https://doi.org/10.1090/conm/453/08807

Roy, S., & Akter, R. (2018). *Pseudo-Triangulation of a Complex Polygon Using Augmented Ear Cutting Approach and Visibility Information*.

Roy, S., & Shapla, R. (2018). *Pseudo-Triangulation of a Complex Polygon Using Augmented Ear Cutting Approach and Visibility Information Swarna Roy*.

Shamos, M. I., & Hoey, D. (1975). Geometric intersection problems. *17th IEEE Symp. Foundations of Computer Science (FOCS '76)*, 208–215. https://doi.org/10.1109/sfcs.1976.16

Shen, L., Yang, Z., & Hao, S. (2021). An adaptive triangulation optimization algorithm based on empty circumcircle. *IMCEC 2021 - IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference*, 1880–1884. https://doi.org/10.1109/IMCEC51613.2021.9482071

Speckmann, B., & Tóth, C. D. (2005). Allocating vertex π-guards in simple polygons via pseudo-triangulations. *Discrete and Computational Geometry*, *33*(2), 345–364. https://doi.org/10.1007/s00454-004-1091-9

Tarjan, R. E., & Van Wyk, C. J. (1986). *A linear-time algorithm for triangulating simple polygons*. 380–388. https://doi.org/10.1145/12130.12170

Toussaint, G. (1991). Efficient triangulation of simple polygons. *The Visual Computer*, *7*(5–6), 280–295. https://doi.org/10.1007/BF01905693

Toussaint, G. T. (1984). A new linear algorithm for triangulating monotone polygons. *Pattern Recognition Letters*, *2*(3), 155–158. https://doi.org/10.1016/0167-8655(84)90039-4

Üngör, A. (2004). Off-Centers: A New Type of Steiner Points for Computing Size-Optimal Quality-Guaranteed Delaunay Triangulations. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *2976*, 152–161. https://doi.org/10.1007/978-3-540-24698-5_19

Žalik, B. (2005). An efficient sweep-line Delaunay triangulation algorithm. *CAD Computer Aided Design*, *37*(10), 1027–1038. https://doi.org/10.1016/j.cad.2004.10.004