**Triangulation a region between arbitrary polygons**

Stetsyk Oleksii

**Abstract.**

In this paper we propose the algorithm for triangulation of an area between arbitrary polygons on the plane with time complexity O(n log n). This algorithm consists of two steps: firstly, we reduce our problem to triangulation polygon with holes, and after that we use modification of monotone polygons methods. For first step we use Graham’s method to build convex hull. For second step we generalize method of monotone polygons for case when we have inner polygons. So, an efficient triangulation algorithm is received. The proposed algorithm is very simple for understanding and have a very simple implementation.

1. **Introduction**

The problem of optimal triangulation of a region between arbitrary polygons on a plane is considered. This topic is very relevant because triangulation is used in microbiology, geodesy, and others. There are few effective algorithms for solving this problem [1, 12]. However, searching for an optimal way of solution is still an actual task today.

If we analyze existing approaches to solving the problem under consideration, we may notice the following. Goodman showed an opportunity to triangulate a region between arbitrary k polygons in time (n + k ^ 2) [5]. Then Chazel showed that the complexity of the algorithm depends on the shape between the polyhedra [1]. In the work [3] authors proposed the method of the modified Delaunay triangulation with limitations for d-dimensional space in O(n^2) time. Tarian and Van Vick in [7] proposed a triangulation algorithm for a simple polygon for O (n log log n). Clarkson [9], Devil [10] and Seidel [11] proposed a randomized algorithm that has O (n log \* n) execution time. All algorithms mentioned above give good results in the case of convex polygons (polyhedrons), however, it is desirable to have an optimal solution to the general case. Naturally, the next question arises: whether it is possible to develop an algorithm that would give high efficiency and have a simple enough implementation. It is especially important in terms of practical applications of the algorithm. For instance, it can be a triangulation of a simple polygon with holes of arbitrary shape using a model of a unified algorithmic environment. In 2017, Tereshchenko proposed an effective algorithm for solving the problem for the case of arbitrary polygons in time (n log n) [2]. Our algorithm differs from the one proposed in the article so that we avoid using the method of monotone chains, and we use the modification of the method of partitioning on monotone polygons. Purpose of this article is next: to develop an effective and simple algorithm for triangulation a region between arbitrary polygons.

**2. Problem and method of solution.**

Formulation of the problem. Triangulate a region between arbitrary k polygons with the total number of n vertices.

2.1 Reducing our problem to triangulation polygon with holes.

Let's write all the vertices of all polygons. Now we will describe Graham’s algorithm. The first step in this algorithm is to find the point with the lowest y-coordinate. If the lowest y-coordinate exists in more than one point in the set, the point with the lowest x-coordinate out of the candidates should be chosen. Call this point *P*. This step takes O(*n*), where *n* is the number of points in question. Next, the set of points must be sorted in increasing order of the angle they and the point *P* make with the x-axis. Sorting in order of angle does not require computing the angle. It is possible to use any function of the angle which is monotonic in the interval [0, pi]. The cosine is easily computed using the dot product, or the slope of the line may be used. If numeric precision is at a stake, the comparison function used by the sorting algorithm can use the sign of the cross product to determine relative angles. The algorithm proceeds by considering each of the points in the sorted array in sequence. For each point, it is first determined whether traveling from the two points immediately preceding this point constitutes making a left turn or a right turn. If a right turn, the second-to-last point is not part of the convex hull, and lies 'inside' it. The same determination is then made for the set of the latest point and the two points that immediately precede the point found to have been inside the hull, and is repeated until a "left turn" set is encountered, at which point the algorithm moves on to the next point in the set of points in the sorted array minus any points that were found to be inside the hull; there is no need to consider these points again. (If at any stage the three points are collinear, one may opt either to discard or to report it, since in some applications it is required to final all points on the boundary of the convex hull.)

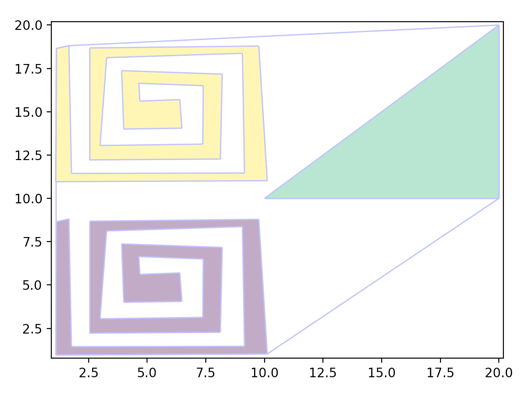


Fig. 1

On figure 1. we can see example of convex hull around three figures.

**2.2 Partition polygon with holes.**

A polygonal chain C is called monotone with respect to line L, if every line orthogonal to L intersects C at most once. We call these chains monotone chain. A polygon P is monotone with respect to a line L if its boundary can be split into two chains, each being monotone with respect to L. We call these polygons monotone polygons. We say that a polygon P is horizontally monotone (or *x-*monotone) if P is monotone *w.r.t. x-axis.* We can triangulate monotone polygon in {\displaystyle {\mathcal {O}}(n)} time O(n), where n {\displaystyle n} is the number of vertices of the monotone polygon. The algorithm is described in section 3.3 of the book [13]. If a simple polygon is not monotone, it can be made monotone, in {\displaystyle {\mathcal {O}}(n\log n)} time O(n log n), using a sweep-line approach. To see it, read section 3.2 of the book [13]. So, this is approach how we can triangulate polygon without holes. But what about inner polygons? To solve this problem, we need to generalize our partitioning approach. So, in book [13] we have 5 types of vertices.

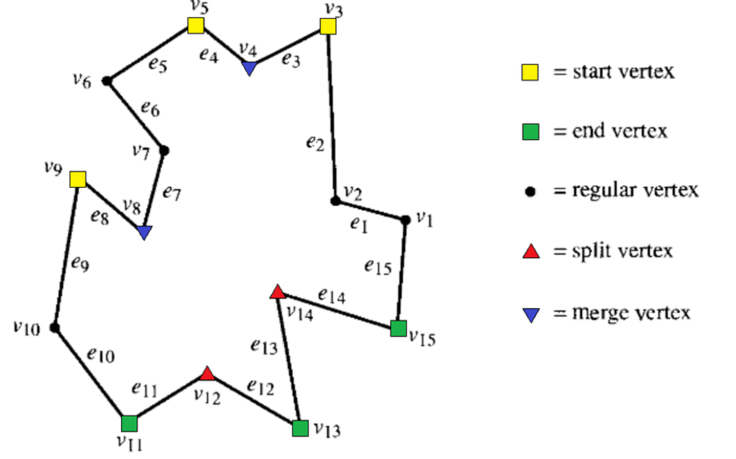


Fig. 2

For inner polygons we need just to replace start vertex and end vertex and split and merge vertex. And after that our algorithm will work for arbitrary polygon with holes.

**3. Complexity estimation.**

**Theorem 1.** The time complexity of solving the triangulation problem will take O (n log n) operations and using O (n) memory.

Substantiation. Let's look at each step of the algorithm described. The first step is the construction of a convex hull. According to [14], the construction of a convex hull takes, in the worst case, O (n log n) operations and O (n) memory. To put all of the points in a queue with priorities, and during the execution of the program to get them we need O (n log n) time and O (n) memory. Also, to add and remove an edge from the splay tree, we need O (log n) operations. Since, according to [13], the number of edges will be O (n), the total number of operations will be O (log n) \* O (n) = O (n log n) and memory usage will be O (n). Next is the triangulation of monotone polygons. According to [13], the triangulation of a monotone polygon with k vertices takes O (k) time and O (k) memory. So general time complexity will be O (n log n) + O (n log n) + O (n) = O (n log n), and memory usage O (n) + O (n) + O (n) = O (n). The theorem is proved.

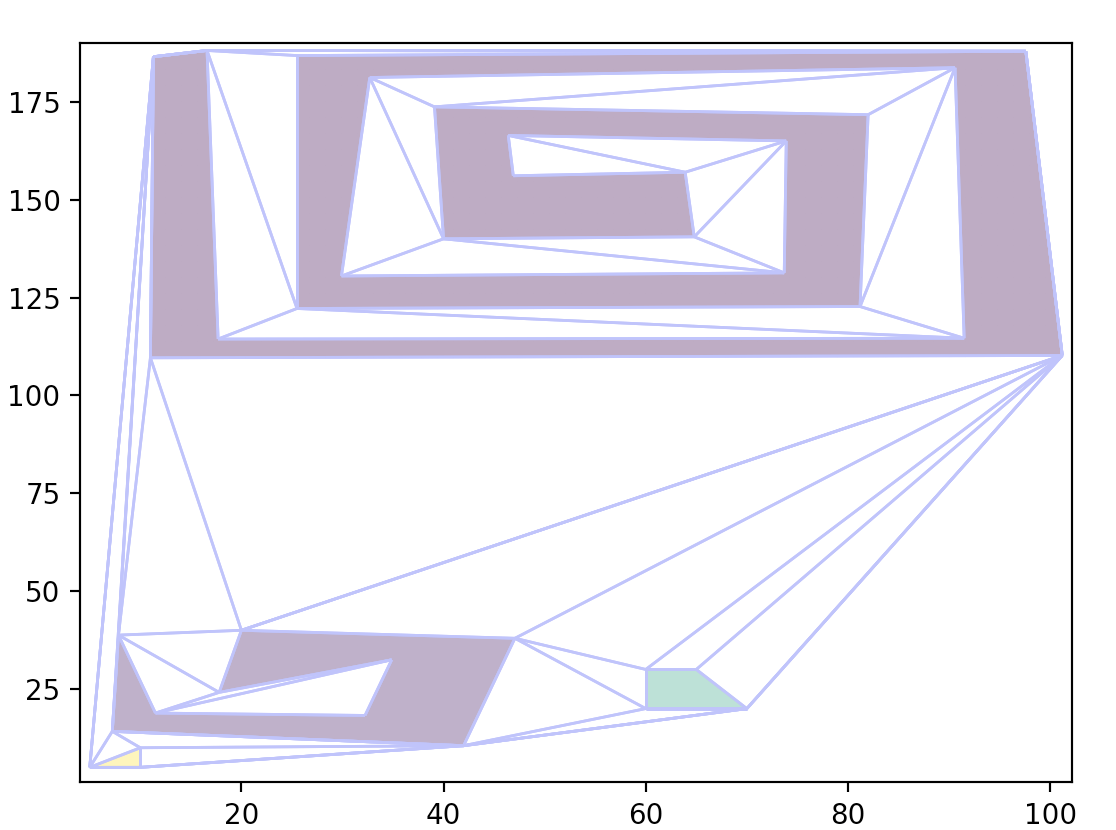


Fig. 3

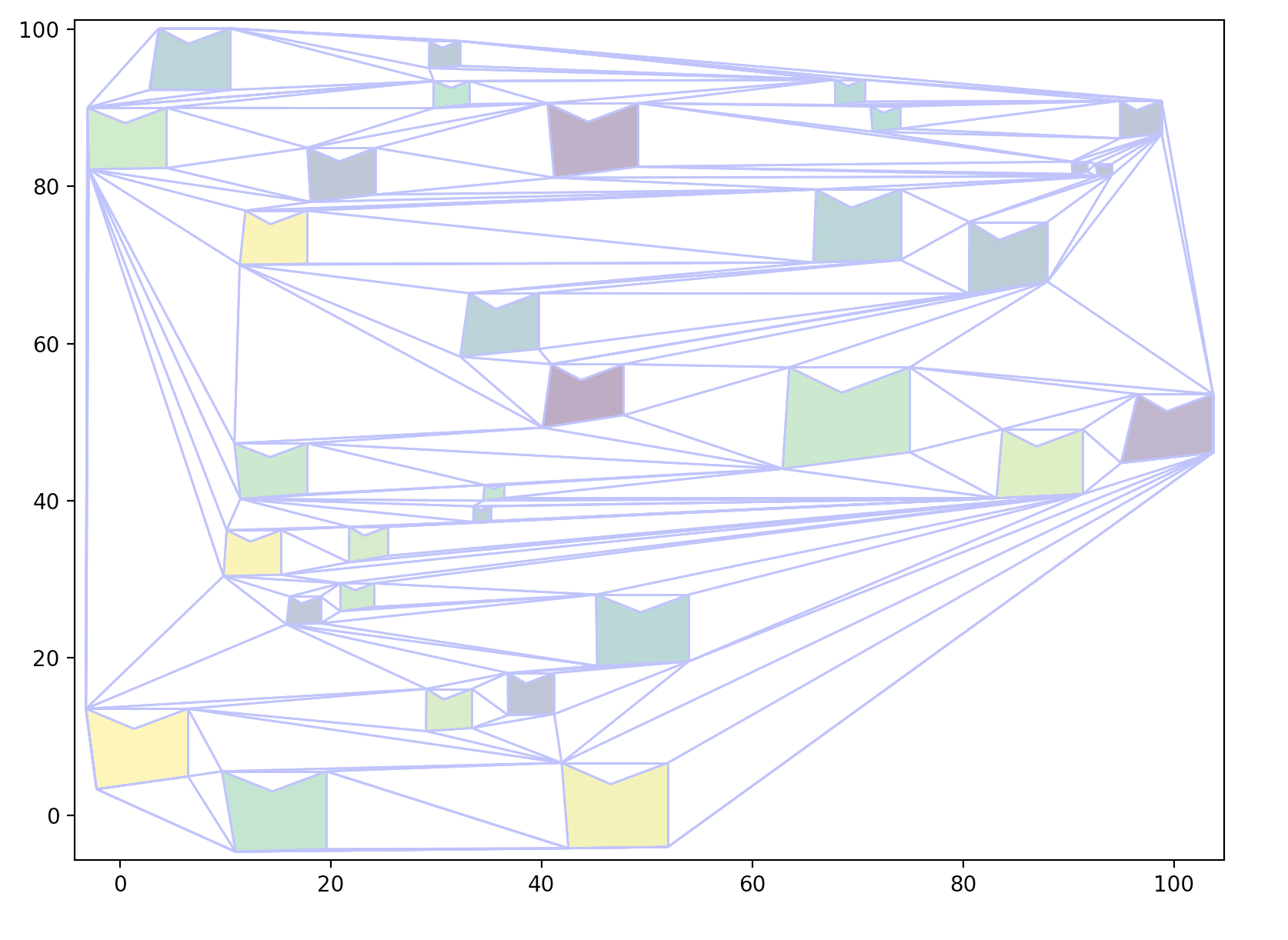


Fig. 4

4. Implementation

One feature of the proposed algorithm is a very simple implementation. The program takes on the input a certain number of polygons and triangulates the area between them. The algorithmic part of the program implementation is written in C ++, python is used for visualization. We use the matplotlib and pyQt libraries. C ++ implementation consists of two files main.cpp and splay.h. In splay.h, class splay tree is implemented, and in main.cpp we use this implementation for splitting into monotone polygons. For splay tree we use Wu Liang’s implementation [15]. Figure 3 and Figure 4 show examples of work of our implementation.

**Conclusion.** In this paper, we propose the method of triangulating a region between arbitrary polygons having n vertices in total in O(n log n) time. The main idea of the described method is in the fact that it consists of two stages: reducing the problem to triangulating a simple polygon with holes and triangulating of the polygon by splitting it onto monotone polygons using the modificated method of partitioning. The suggested method gives us quick work in practice. Another advantage of the proposed algorithm is that it can solve an extended problem: the case of “nested holes”.

**References**

[1] B. Chazelle, N. Shouraboura, “Bounds on the size of tetrahedralizations,” J. Discrete & Computational Geometry, vol. 14, no. 1, pp. 429-444, 1995.

[2] V. Tereshchenko, Y. Tereshchenko , “Triangulation a region between arbitrary polygons” International Journal of Computing, 16(3) 2017, 160-165.

[3] F. Preparata, M. I. Shamos, Computational Geometry: An Introduction, Springer-Verlag, Berlin, 1985, 398 p.

[4] V. S. Bileckiy, Small Mine Encyclopedia, vol. 1-3, Shidniy Vidavnichiy Dim, Donetsk, 2013, 1936 p. (in Ukrainian)

[5] J. E. Goodman, J. O’Rourke, Handbook of Discrete and Computational Geometry, 2nd ed., CRC, A CRC Press Company, Boca Raton, London, 2004, 1453 p.

[6] B. Joe, “Construction of three-dimensional Delaunay triangulations using local transformations,” J. Computer Aided Geometric Design, no. 8, pp. 123-142, 1991.

[7] R. E. Tarjan, C. J. Van Wyk, “An O(nlog logn)-time algorithm for triangulating a simple polygon,” SIAM J. Comput, vol. 17, pp. 143– 178, 1988.

[8] D. G. Kirkpatrick, M. M. Klawe, R. E. Tarjan, “Polygon triangulation in O(nlog logn) time with simple data structures,” J. Discrete Comput. Geom., vol. 7, pp. 329–346, 1992.

[9] K. L. Clarkson, R. E. Tarjan, C. J. Van Wyk, “A fast Las Vegas algorithm for triangulating a simple polygon,” J. Discrete Comput. Geom., vol. 4, pp. 423–432, 1989.

[10] O. Devillers, “Randomization yields simple O(nlogn) algorithms for difficult Ω(n) problems,” Internat. J. Comput. Geom. Appl., vol. 2, pp. 97–111, 1992.

[11] R. Seidel, “A simple and fast incremental randomized algorithm for computing Vasyl Tereshchenko, Yaroslav Tereshchenko / International Journal of Computing, 16(3) 2017, 160-165 165 trapezoidal decompositions and for triangulating polygons,” J. Comput. Geom. Theory Appl., vol. 1, pp. 51–64, 1991.

[12] V. Tereshchenko, S. Pilipenko, A. Fisunenko, “Domain triangulation between convex polytopes,” J. Procedia Computer Science, vol. 18, pp. 2500–2503, 2013.

[13] M. de Berg, M. van Kreveld, M. Overmars, O. Cheong, Computational Geometry, 3rd ed., Springer-Verlag, Berlin, 2008, 398 p. [14] B. Chazelle, “Triangulating a simple polygon in linear time,” J. Discrete Comput. Geom., vol. 6, pp. 485-524, 1991. [15] E. Edelsbrunner, L. J. Guibas,

[14] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009). Вступ в алгоритми (вид. 3rd). MIT Press.ISBN 0-262-03384-4.

[15]. Чаднов Р. В. Алгоритмы построения выпуклых оболочек и их применение в ГИС и САПР — Томск: Томск. гос. ун-т. Факультет информатики, 2004.- 61 с.

[16]. Jarvis, R. A. (1973). On the identification of the convex hull of a finite set of points in the plane. Information Processing Letters 2. с. 18–21.

[17]. Atsuyuki Okabe, Barry Boots, Kokichi Sugihara & Sung Nok Chiu (2000). Spatial Tessellations – Concepts and Applications of Voronoi Diagrams. 2nd edition. John Wiley, 2000, 671 pages, ISBN 0-471-98635-6

[18]. Ottman, T.; Soisalon-Soisinen, E.; Wood, Derick (1984), «On the definition and computation of rectilinear convex hulls», Information Sciences 33 (3): 157–171.