**Implementation of Graham’s Scan Method for Triangulation of Non Convex Polygons in Java Netbeans Environment**

Faruk Selimović1, Muzafer Saračević1, Vesad Doljak1

1Department of computer sciences, University of Novi Pazar, Serbia,

faruk@uninp.edu.rs

muzafers@uninp.edu.rs

vesko95@yahoo.com

*Abstract* ***-* This paper presents an algorithm for triangulation of non-convex polygons on the principle of Graham’s scan. This method is based on finding the so-called "ear" of triangulation of the observed polygons. Second aspect is storing in a temporary stack all the previous points of the polygon from which would later be pulled the diagonals that will actually create the triangulations. Triangulation of the polygon has increasing application in the design of the Geographic Information Systems (GIS) and it is the basis for calculating the surface of irregular terrains. In particular, we have implemented mentioned method in Java Net Beans environment with a graphical user interface. Java, as one of the leading object-oriented programming languages allows efficient implementation of this algorithm, and it is one of the main reasons of implementation of this algorithm in this programming language***.*

*Keywords* - ***Polygon triangulation, Graham’s scan method, non-convex polygon, Java Net-Beans environment.***

1. **Introduction and preliminaries**

Polygons are very important objects in geometry and in particular to their applications in computing [1, 2]. In order for the surfaces and the curves to be calculated more easily and in order to more easily graphically represent them, they are replaced by polygons or polygon surfaces. Triangulation of polygons in today's modern information age plays such an important role in the design of a GIS (Geographic Information Systems) as well as in projecting of GPS (Global Positioning System) where the location of objects on the ground, by using the x, y and z coordinates, actually enables the use of the triangulation system.

The concept of triangulation of the monotone polygon actually represents sharing the inside of the polygon into triangles, mutually non intersecting internal diagonals. Each simple or monotonous polygon has triangulation. Any triangulation of simple polygon with n vertices consists of exactly n - 2 triangles. This can be proved by mathematical induction [3]. There are many algorithms for triangulating polygons. They mainly differ in complexity, design and programming languages ​​in which they are implemented. Below, we present the basic concepts needed for further consideration:

* Def 1. Polygon line A1 ... An, An + 1 is the union of lengths A1, A2 ... An, An + 1, which we call the edges of polygonal lines. Points A1 ... An + 1 are called the vertices of polygonal lines. If An + 1 = A1 we say that the polygonal line is closed and we call it the polygon. If any two edges of polygonal lines have no common points except that each two adjacent lines have common vertices, polygon line or polygon, we call them simple [4].
* Def 2. For the line that connects two non-consecutive tops of the polygon P we say it to be a diagonal from P.
* Def 3. For the top pi of the polygon P we say that it is the ear if the page [pi-1, pi + 1] is diagonal of P, and pi + 1 is major (Eng. Top) top of the ear pi. On Figure 1 we present the tops of polygons which we call ear. For the polygon left in Figure 1 we say that Pi is not the ear, because the diagonal [pi-1, pi + 1] is not within the polygon. As for the polygon on the right in Figure 1 for Pi we say that it is the ear because the segment [pi-1, pi + 1] is inside the polygon and does not intersect any other lines.



Figure 1. a) Pi is not an ear b) Pi is an ear

* Def 4. For the two ears pi and pj of polygon P we say do not overlap if the triangles △ (pi-1, pi, pi + 1) and △ (pj-1, pj, pj + 1) are mutually disjoint.

Theorem 1. Every simple polygon with at least four tops has at least two non-overlapping ears (Two ears theorem). In Figure 2 we present ear triangulation of convex and non-convex polygon having four vertices and two non-overlapping ears [5].



 Figure 2. a) Convex polygon b) Non-convex (concave) polygon

1. **Related works**

Authors in paper [6] propose a triangulation method for a set of points in the plane. The method is based on the idea of constructing convex layers by Graham's scan. Algorithm is easily parallelized: each layer can be triangulated independently. The main feature of the proposed algorithm is that it has a very simple implementation and the elements (triangles) of the resulting triangulation are presented in the form of simple and at the same time fast data structures: concatenable triangle queue or triangle tree. This makes the algorithm convenient for solving a wide range of applied problems of computational geometry and computer graphics, including simulation in science and engineering, rendering and morphing.

In paper [7] authors presents the Graham’s scan as a fundamental backtracking technique in computational geometry which was originally designed to compute the convex hull of a set of point in the plane and has since found application in several different contexts. Authors show how to use the Graham’s scan to triangulate a simple polygon. The resulting algorithm triangulates an n-vertex polygon P in *O(kn)* time where k −1 is the number of concave vertices in P.

Authors in paper [8] prove that the triangulations produced by Graham's scan are Hamiltonean. Furthermore, they prove that any triangulation T of a point set which has a point adjacent to all the points in P (a center of T) is Hamiltonean. Given T authors define a graph *G(T)* whose vertices are the triangles of T, two of which are adjacent if they share an edge. They prove that T is Hamiltonean if G(T) has a Hamiltonean path.

In paper [9] authors define an almost-convex polygon as a non-convex polygon in which any two vertices see each other inside the polygon unless they are not adjacent and belong to a chain of consecutive concave vertices. Using inclusion-exclusion techniques, authors find formulas for the number of triangulations of almost-convex polygons in terms of the number and position of the concave vertices. Authors translate these formulas into the language of generating functions and provide several simple asymptotic estimates. Authors also prove that certain balanced configurations yield the maximum number of triangulations.

In our paper [10] we presents a new technique of generation of convex polygon triangulation. The properties of Catalan number were examined. The method presented in the paper was constructed on basis of ballot combinatorial problem. The movements in constructed method through polygon are derived upon vertices and leaves of the binary tree. In this paper are given two algorithms who are reverse to each other and transform the triangulation to ballot record and vice versa.

Implementations of the algorithm for generating and displaying triangulations of the convex polygon, in three programming languages (Java, Python, C++) are described and compared in our paper [11]. Our main aim is to show the advantages and disadvantages of these programming languages in resolving this useful algorithm in computational geometry and computer graphics.

1. **Algorithm based on graham’s scan method**

In this paper we are presenting the working principle of the algorithm based on Graham’s scan which has a complexity of *O(kn),* where k - 1 number of concave vertices in the polygon. This is an important technique of backtracking in computational geometry for finding a convex envelope set of points [13]. By combining Graham’s scan and ear-cutting algorithm is obtained the so-called triangulation algorithm based on Graham’s scan [7].

Let P be a polygon with a set of vertices {p0, p1. . . , pn-1}. Vertices are scanned, starting with the top p2. In every step it is tested whether the current top polygon is the main peak of the ear. If not the main top ear, then the current top moves to the next. If it is the main peak ear, then that ear is removed from the polygon, i.e. we add diagonal into the polygon and delete the top of the polygon. In this case the current top does not move, except in special case, when the ear is successor of p0. This guarantees that the p0 is going to be removed.

Let us explain now the most important function for the implementation of this algorithm (See Appendix A):Let the P = {p0, p1. . . , pn-1} polygon be presented with twice related cyclic list. Let the predecessor (p) and successor (p) respectively be predecessor and successor of peak p in the list. Let D set (abstract data structures set) in which we save diagonals used in triangulation, and R set of all concave tops of P. Function *findEar (P, R, pi)* returns true if pi is the ear, otherwise it returns false. Below we present the function that implements this algorithm [13].

1. **Implementation of algorithm in Java Net-Beans Enviroment**

In this section we explain the implementation of the algorithm in Java Net-Beans environment. The main advantage is that programs written in Java can run on almost any computer, whether it is the Windows operating system, Macintosh computers, UNIX or Linux, large mainframe computers or mobile phones, and even the much smaller devices whose correlation is building what is popularly called "Internet of things". By starting the class Triang\_GrahamScan.java, which contains an executable method, we are actually launching the Applet by which we communicate with the application presented in Figure 3.



Figure 3. Applet for communication with the application

By simple clicking on the surface of the Apple (window) we actually enter the vertices in a doubly linked list, i.e., we call on the method *add\_element()* from the class dual\_Linked\_List.java that adds coordinates of the vertices in a doubly linked list. After completing the entry of the vertices by simply clicking the right button of the mouse we will close the polygonal line and thus the polygon will be to formed. By clicking “Triangulated polygon”, we call for the method *concave\_vertex(),* which examines all the vertices of the polygons whether they are concave.

This is important because the concave vertices are typically found within the triangles that need to be potential triangulations. Then is called the method *cut\_ear\_triang().* This method requires the first triangulation in which there are no concave vertices because it is the most important in the process of triangulation. Actually it asks for the first vertice which is the "ear". The algorithm works by taking into account the first three vertices (p1, p2, p3) which are inserted in the polygon and examine whether p2 is ear.

* If p2 is an ear (this is in fact the first triangulation) then the observed triangle is cut (p2 threads is removed from the list) and remains the so called "Modified triangle". The algorithm continues to work so you can now observe the triangle (p\_n, p1, p3). Now we examine whether p1 is an ear, and so on until you come to the last thread on the list. Remember, the way in which it is determined whether a thread is an ear is calculation of the surface area of the observed triangle (method *area\_Triangle*) as vector product. If the vector product is positive it means that the secondary crown (in our case, p2) is an ear, i.e. that it is not concave, otherwise the crown is concave.
* If p2 is not an ear, the algorithm continues to work by the counter in the loop increasing by 1 and now we observe the triangle (p2, p3, p4). It is examined whether p3 is an ear, and so on until the ear is found. Figure 4. presents the work of the algorithm.



Figure 4: a) Observed polygon; b) First triangulation;

c) Second Triangulation; d) Third and fourth triangulation

Also, in addition to the visual display of the work of the algorithm we present the source code in Java of the main method *cut\_ear\_triang* (See Appendix B).

1. **Conclusion**

In this paper we presented working principle of the algorithm based on Graham’s scan method for triangulation of concave polygons. We have performed Java implementation of proposed method for triangulation. In particular, we have implemented method in Java Net Beans environment with a graphical user interface which offers efficient and quick generation of triangulations.

This is an important technique of backtracking in computational geometry for finding a convex envelope set of points. In the computer graphics is used for modeling 3D objects. It is used to detect a collision of object (a physical simulation, calculating the sense of touch, prevents the passage of an object through another one, etc.). In urban planning is used in determining the position of important facilities, in road construction (detection of obstacles), in archeology for determining the areas of influence of groups of animals. Due to this wide use, this paper is the basis of further research and finding a faster algorithm for more efficient solution to the problem of triangulation of monotone-free polygons. Despite the fact that triangulation of monotone-free polygons in modern information society has a very important role in the GIS industry, it also has wide application in other activity branches.

1. **References**
2. Minsky, M., & Papert, S. (1969), “Perceptrons: An Introduction to Computational Geometry”, M.I.T. Press, Cambridge, Massachusetts, USA.
3. Preparata, F. P., & Shamos, M. I. (1985), “Computational Geometry: an introduction”, Springer-Verlag, NewYork, USA.
4. Berg, M., Cheong, O., Kreveld, M., & Overmars, M. (2008), “Computational Geometry Algorithms and Applications: Third Edition”, Springer-Verlag Berlin Heidelberg.
5. Rvović, V. (2014). Application of Silverlight for displaying geometric bodies: Master thesis (pp.5-6), Novi Sad, Serbia.
6. ElGindy, H., Everett, H. & Toussaint, G. (1993), “Slicing an ear using prune-and-search”,  Pattern Recognition Letters, 14(9), 719-722.
7. Tereshchenko,V., Tereshchenko, Y., & Kotsur, D. (2015), “Point triangulation using Graham’s scan”, In: Fifth International Conference on the Innovative Computing Technology (INTECH 2015), Pontevedra, pp. 148-151.
8. Kong, X., Everett, H., & Toussaint G. (1990), “The Graham scan triangulates simple polygons”, Pattern Recognition Letters, 11(11), 713-716.
9. Fabila Monroy, R., & Urrutia, J. (2005), “Graham triangulations and triangulations with a center are Hamiltonian”, Information Processing Letters, 93(6), 295-299.
10. Hurtado, F., & Noy, M. (1997), “Counting triangulations of almost-convex polygons”, Ars Combinatoria, 45 , 169-179.
11. Saračević, M., Stanimirović, P., Krtolica, P., & Mašović, S. (2014), “Construction and Notation of Convex Polygon Triangulation based on ballot problem”, ROMJIST- Journal of Information Science and Technology, 17 (3), 237-251.
12. Saračević, M., Stanimirović, P., Mašović, S., & Biševac, E. (2012), “Implementation of the convex polygon triangulation algorithm”, Facta universitatis-series mathematics and informatics, 27(2), 213-228.
13. De Berg, M., Cheong, O., Kreveld, M. & Overmars, M. (2008), “Computational Geometry, Algorithms and Applications: Third Edition”, Springer-Verlag Berlin Heidelberg.
14. Penharda, D. (2006), “Algorithms for testing the membership of a point polygon: Bsc thesis”, Zagreb, Croatia.

**Apendix A**

find (P, R, pi) {

if (R empty) return true; // P is convex

else

if (pi is convex){

if(triangle (predecessor(pi), pi, successor(pi)) does not contain any of the top R)

return true;

else return false;

}

pi = p2;

while (pi!= p0){

if(Find\_Ear(P, R, predecessor(pi)) && P not triangle){

// predecessor (pi) is ear

D = D + [predecessor (predecessor (pi)), pi]; // save diagonal

P = P predecessor (pi); // remove ear

// pi becomes convex

if (pi u R && pi is convex) R = R - pi;

if(predecessor (pi) u R && predecessor (pi) konveksan)

R = R - predecessor (pi);

// successor is removed

if (predecessor (pi) = p0) pi = successor (pi);

}

else pi = successor (pi);

}

**Appendix B**

void cut\_ear\_triang(){

Point2D.Double VerticesList= (Point2D.Double)DP\_List.AddFirtsElement();

DP\_List.ShowNextPosition();

DP\_List.ShowNextPosition();

Point2D.Double p = (Point2D.Double)DP\_List.CurrentElement();

while (!p.equals(Temena\_u\_listi) && numberVertices>4) {

DP\_List.ShowBackPosition();

if (Ear\_trang((Point2D.Double)DP\_List.AddBackElement(), (Point2D.Double)DP\_List.CurrentElement(), p) && numberVertices> 4) {

Line2D.Double li = new Line2D.Double ((Point2D)DP\_List.AddBackElement(), (Point2D)DP\_List.AddNextElement());

diagonal.addElement(li);

DP\_List.Briši\_tekući();

numberVertices--;

if (concaveVer.contains(p) && ConvexVer( (Point2D.Double)DP\_List.AddBackElement(), p,

(Point2D.Double)DP\_List.AddNextElement())){

concaveVer.remove(p);

}

DP\_List.ShowBackPosition();

if (concaveVer.contains(DP\_List.CurrentElement()) && ConvexVer( (Point2D.Double)DP\_List.AddBackElement(),(Point2D.Double)

DP\_List.CurrentElement(), p)){

concaveVer.remove(DP\_List.CurrentElement());

}

DP\_List.ShowNextPosition();

if (Temena\_u\_listi.equals(DP\_List.CurrentElement())) {

p = (Point2D.Double)DP\_List.AddNextElement();

DP\_List.ShowNextPosition();

 }

 } else{

DP\_List.ShowNextPosition();

p = (Point2D.Double)DP\_List.AddNextElement();

DP\_List.ShowNextPosition();

 }

 }

DP\_List.AddFirtsElement();

DP\_List.ShowNextPosition();

}