# Triangulation of Large 2D Object Sets Using Several Algorithms

Alexander Emelyanov
Institute of Computing for Physics and Technology
Protvino, Moscow reg., Russia
aiem@rambler.ru

## Abstract

An approach for fast triangulation of large sets of 2D polygonal objects with possible holes using several triangulation algorithms is described. A benefit of using this approach in cases when such sets of objects have certain statistical properties (for example, GIS and similar systems) is highlighted.

***Keywords:*** *simple polygon triangulation, multi-algorithm processing, GIS.*

## 1. INTRODUCTION

In many fields of science and industry we deal with graphics data represented by 2D polygonal objects with possible holes. Each such object is represented by its boundaries (the external boundary and the boundaries of holes, if they exist). Each boundary is defined by its vertices. This representation very close corresponds to the GDI. As a consequence, GDI-based tools are widely used to display such graphics data.

At the same time, using technologies based on the DirectX and the OpenGL provides an essentially better performance and quality, moreover that the most part of modern PC video-cards (even built-in and office ones) have a hardware support of these technologies. But to use them we need to triangulate input objects first. There are many algorithms for this purpose, and in general case realization of this operation is not a problem.

But in several important cases we need to display a stream of such graphics data in real time. A typical example is GIS systems, when a user device continually gets data from a data source. So, we need to have an approach which is capable to make triangulation as fast as possible. The triangulation quality in such cases is usually restricted only by requirements of used rendering devices, which usually are quite weak.

## 2. GENERAL STRATEGY

### 2.1 Basic definitions

**Definition D2.1.1** A given object ($O_1$) has a complexity ($C_1$) greater than the complexity ($C_2$) of second one ($O_2$) concerning a given triangulation algorithm ($A$), if the algorithm can't process $O_1$ but can process $O_2$ or if the processing time per vertex for $O_1$ is greater than for $O_2$.

**Definition D2.1.2** A given triangulation algorithm ($A_1$) has a robustness ($R_1$) greater than the robustness ($R_2$) of second one ($A_2$) concerning a given object ($O$), if $A_1$ can process $O$ and $A_2$ can't.

It is obvious, that these introduced above characteristics aren't absolute. For two objects ($O_1$, $O_2$) and two algorithms ($A_1$, $A_2$) can be that $C_1 > C_2$ concerning $A_1$ but $C_1 < C_2$ concerning $A_2$; and $R_1 > R_2$ concerning $O_1$ but $R_1 < R_2$ concerning $O_2$. But in practice in a majority cases such characteristics can be considered as absolute. So, for simplicity and clarity of the further explanations let's assume that if $C_1 > C_2$ concerning $A_1$ than it takes place for any other algorithm; and if $R_1 > R_2$ concerning $O_1$ than it takes place for any other object.

Let's note that among triangulation algorithms used in practice in general the following dependence takes place:

$$R \sim 1/S \quad \text{(E2.1.1)}$$

where $S$ is the speed of work.

**Definition D2.1.3** Let there is a set of triangulation algorithms. An algorithm of this set is the *corresponding* to a given object (or a group of objects) concerning the algorithm set, if it can successfully process this object (group) and among other algorithms of the set with the same property has the maximal speed of work.

From E2.1.1 follows that the corresponding algorithm has the minimal robustness among algorithms which can process a given object (group).

### 2.2 Strategy of sorted objects processing (SOP)

Let's suppose that there is a method, which can distribute a given object set into $N$ groups, with condition that the complexity of any object of the group with index $i$ is greater than the complexity of any object of the group with index $j$ if $i > j$, $1 \le i, j \le N$;

Let there is also a set of $M$ triangulation algorithms, which have the different robustness, with condition that at least the most robust algorithm can process the whole object set. Let's assign to each object group its corresponding algorithm from the given set of algorithms.

Thus, if among these corresponding algorithms there are at least two different ones, the total cost of processing the whole object set using for each object group its corresponding algorithm is less than the cost of processing the object set using only the corresponding algorithm of the last object group.

The cost of this strategy can be expressed by the following formula:

$$T = T_d + \sum_{i=1}^{N} t_i \quad \text{(E2.2.1)}$$

where $T_d$ is the cost of the object distribution into the groups; $t_i$ is the cost of processing $i$-th group by its corresponding algorithm.

So, this strategy is effective if:

- there is a valuable difference in the speed between the corresponding algorithms;
- the strongest algorithm isn't corresponding to a majority of the objects;
- the cost of the object distribution is quite small..

To implement the strategy we need:

- to choose (or develop) a set of suitable triangulation algorithms;
- to make a classification of objects taking into account properties of the chosen algorithms;
- to develop an algorithm to distribute objects into groups according to the developed classification.

### 2.3 Strategy of processing by sorted algorithms (PSA)

There is also another possibility of multi-algorithm processing implementation, that doesn't require the preliminary distribution of objects. Let there is a set of $N$ objects (the index of an object in the set doesn't depend on its complexity) and there is a set of $M$ algorithms, which are sorted in the ascending order of their robustness, with condition that at least the last algorithm can process the whole object set. The strategy can be represented by the following pseudo code:

```
Objects: array [1..N] of Object;
Algorithms: array [1..M] of Algorithm;

for (i = 1 to N)
  begin
   for (j = 1 to M)
     if (Algorithms[j].processing(Objects[i]) = SUCCESS)
       break;
  end;
```

In others words, we sequentially try to process an object starting from the weakest (and the fastest) algorithm, and apply the algorithms one by one until the object is processed. In general, during processing an object by an algorithm, which finally is turned out unsuccessful, an useful information about the object can be obtained and used for more precise classification of the object and accelerate its processing by more robust algorithms.

The cost of the strategy can be represented by the following formula:

$$T = \sum_{i=1}^{N} \sum_{j=1}^{m_i} t_i^j \quad \text{(E2.3.1)}$$

where $m_i$ is the index of the corresponding algorithm of $i$-th object; $t_i^j$ is the cost of processing $i$-th object by $j$-th algorithm.

So, such strategy can be effective if:

- a majority of an input consists of quite simple objects;
- there is a valuable difference in the cost between the weakest and the strongest algorithms.

### 2.4 Strategy of processing probabilistically sorted objects (PSO)

The performance of the strategy PSA can be improved if there is a method that for a given object can predict that it can't be processed by algorithms with the indices smaller than some certain one. In this case it is reasonable to start the consecutive algorithm application from the algorithm with this index (let's call it the *beginning algorithm*). In fact this method establishes the corresponding classification of the objects, such that objects with the same beginning algorithm are in the same group. If the prediction method is absolutely reliable, the beginning algorithm of a given object is its corresponding one and the PSA becomes to the SOP.

Because in practice, most often we can't achieve 100% reliability of such prediction, the introduced strategy combines properties of the SOP and the PSA. The prediction reliability ($r$) can be considered as the parameter ($0 \le r \le 1$) of such combination, that can be expressed in the following way:

$$PSO = r(SOP) + (1-r)(PSA) \quad \text{(E2.4.1)}$$

An algorithmic scheme of the strategy is presented in scheme S2.4.1.

## 3. OBJECT ESTIMATION AND CLASSIFICATION

As it has been shown in the previous section, one of the key elements of the multi-algorithm processing is a proper method of estimation of the complexity of input objects concerning chosen triangulation algorithms. Such estimation method should:

- correspond to capabilities of chosen algorithms;
- take into account properties of a given input object set;
- work fast;
- provide maximal reliability.

As an input set of objects a typical object set representing a city has been taken. This set contains as relative simple objects corresponding to a majority of buildings as complicated ones representing roads, parks, rivers, lakes (figs. F3.1-6).

There are very many triangulation algorithms (see [MZ00]), but from the point of view of the robustness they can be divided to the following two big groups:

1) algorithms which can't process objects with holes;
2) algorithms which can do it.

In accordance with D2.1.2 and E2.1.1, algorithms of group (1) have the less robustness and the greater speed than algorithms of group (2). Let's explore the given object set from the point of view of hole existence. The results are shown in tables T3.1, T3.2.

As we can see, the share of objects with holes is very small, therefore the total performance of processing the whole object set doesn't depend essentially on efficiency of processing such objects. Thus, it is reasonable to consider holes as a disturb factor and "cut out" them at a post-processing step.

So, let's implement a multi-algorithm strategy for objects without holes. There are several methods proposed [TS91, YH08] to make efficient object shape complexity estimation, but they are too costly for massive processing objects. At the same time, to implement one of the described above multi-algorithm strategies we are interested only in estimation of the object complexity from the point of view of D2.1.1. Thus, we need to estimate a probability per vertex of occurrence a complex topological case that requires a costly routine or that can lead to an error. It is clear, that a regular polygon even with a very large number of vertices has the complexity near the zero. At the same time the complexity of an object represented by only several vertices can be very high if in it there are extreme small values of edge lengths, angles, vertex-edge distances. Because we deal with processing of large object sets, for our purpose can be usable the following statistical observation: in general, the probability of occurrence of a problem case increases with a deviation of the

shape of a given object from its convex approximation. Also, many of existing algorithms are sensitive to the number of concave vertices. Using this number as an object complexity indicator is proposed first in [HM83]; in [TS91] in spite of criticism of using it for object shape estimation, the author advises using this number to select (from [KET90, TS91]) the most suitable algorithm to process a given object. On the base of these things it is reasonable to take the ratio of the number of concave vertices ( $n_{cv}$ ) to the total vertices number ( $n$ ) of the object boundary as the complexity indicator. Let's call this ratio the concavity factor ( $f_{cv}$ ) of a given object:

$$f_{cv} = \frac{n_{cv}}{n} \quad \text{(E3.1)}$$

Let's explore from the point of view of this indicator the input object set. For it we distribute the objects into $N = 10$ groups; the group with a given index $i$ ($1 \le i \le N$) consists of objects with the $f_{cv}$ in the following range: $\frac{i-1}{N} \le f_{cv} < \frac{i}{N}$. The results of this distribution are shown in tables T3.3, T3.4. From these results follows that for the considered kind of data the concavity factor in general correlates with the main parameters of the object shape complexity in general assumption. Also, we can see that the most part of the set consists of quite simple objects.

## 4. MAIN APPROACH

Thus, for the given object set we have the introduced above estimation method which is fast but isn't absolute reliable. It corresponds to the case of the PSO strategy. For its implementation the following three algorithms have been chosen (in the ascending order of their robustness):

1) an algorithm to triangulate an object by its recursive binary division (see details in 5.1);
2) a version of the well known "'cutting ear" algorithm [KET90];
3) an algorithm that works by inserting edges in an existing triangulation (see details in 5.2).

Algorithm (1) is the weakest and the fastest. It is intended to process the most part of the given set that is represented by simple objects. This algorithm can be considered as a generalized variant of the "cutting ear" algorithm, its greater speed of work is achieved by giving the robustness up. Algorithm (2) is intended to process the remaining after application of algorithm (1) objects, which aren't too complicated. The algorithms described in [CH91, SE91, TS91], in spite of the fact that the asymptotic behavior of their costs is better than the same behavior of [KET90], use complex auxiliary data structures, so their cost of initialization seems too big to process such kind of objects. In [TS91] using the algorithm described in [KET90] is explicitly recommended when the number of concave vertices is small. Algorithm (3) is used as the strongest one; also it intended to implement the post-processing step of "cutting out" holes. Taking into account, that the total amount of work for such algorithm is quite small, it is reasonable neither to make different implementations for these two tasks nor to implement a complicated algorithm such as [SE91].

The results of processing the input object set by the chosen algorithms are shown in tables T4.1, T4.2.

Let's determine the beginning algorithm for each of the defined in section 3 object groups. As we can see, for groups (1, 2) it should be algorithm (1). For groups (3, 4), in spite of the fact that algorithm (1) can't process all objects of these groups, it is reasonable to use this algorithm as the beginning because it keeps the cost advantage and the share of unsuccessfully processed objects is very small. For group (5) as the beginning should be chosen algorithm (2). And the remaining groups should be processed by algorithm (3).

Thus, the final approach (let's call it "smart") works as it shown in scheme S4.1. The results of processing the object groups by this approach are shown in table T4.3.

If a processed object contains a hole(s), it has been "cut out" after triangulation of the area inside the object external boundary. For this purpose a variant of algorithm (3) is used.

The results of processing the whole object set are shown in table T4.4. As we can see, the "smart" approach provides essential reducing the cost of processing with comparison of using algorithms (2) and (3).

## 5. IMPLEMENTATION DETAILS

### 5.1 The "dividing" algorithm

This algorithm belongs to the group of algorithms based on diagonal inserting [HM83, KET90, CH91, SE91, TS91]. The crucial operation of any such algorithm is choosing a proper diagonal to divide.

**Definition D5.1.1** Let's call a given diagonal of a polygon *basically correct* if the diagonal is within the internal angles of both its vertices.

**Definition D5.1.2** For a given polygon and a given its diagonal let's call the polygon concave vertices which aren't the diagonal ones and aren't connected with them by the polygon edges the *spoiling vertices* of the polygon concerning the given diagonal (figs. F5.1.1, F5.1.2).

The spoiling vertices are the only factor that can make a basically correct diagonal topologically wrong. In addition, the cost per vertex of the "cutting ear" algorithm described in [KET90] linearly depends on the averaged number of such vertices considered at each step. Thus, a natural way to reduce the cost of similar algorithms is minimization of this number. From this point of view the best diagonal is that eliminates the maximal number of concave vertices from the consideration at the current and the following steps. The maximal possible value of this number is 6. In this case a diagonal connects two concave vertices with elimination of their concavity for the child polygons and have four concave vertices as the nearest neighbors on the polygon boundary (fig. F5.1.1).

In this way a polygon is divided by such "ideal" diagonal or its approximation to two child ones no one of them in general is an "ear". Checking correctness of such diagonal is a more sophisticated task than the checking an "ear" one. There are several conditions to make it absolute reliable, but such test is too costly. At the same time, the following condition of correctness can be formulated:

**Condition C5.1.1** A diagonal is considered correct, if for each child polygon all inherited spoiling vertices of the parent one are in the internal half-plane of the child polygon edge that is this diagonal (fig. F5.1.1).

This condition is only sufficient but not necessary (fig. F5.1.2). In accordance with the results shown in table T5.1.1, approximately 40% of diagonals considered by this condition as wrong in fact are good. But the ratio of such wrong results to the total number of tested diagonals is quite small for groups (1 - 4). Of course, the robustness of the algorithm based on this condition has some probabilistic character, but, as it follows from the results shown in tables T4.1, T4.2, it is justified if objects aren't too complicated and absolute reliability isn't required. In general the algorithm has the same cost dependence as the cost dependence of algorithm (2): $T = cn_{cv}n$ but the value of constant $c$ is less because the number of considered concave vertices decreases faster during the recursive division.

### 5.2 The "edge insertion" algorithm

The used edge insertion algorithm in comparison with a majority of existing ones (one of the most recent is described in [DM04]) doesn't use any auxiliary algorithm to re-triangulate areas at the place of a new edge insertion. It works using only the two following basic operations: the point insertion and the "edge flip", that allows making their deep optimization.

In general case the algorithm works in 5 steps:

1. We determine points of intersection of an edge to be inserted with existing edges. For each segment of the edge bounded by such points the midpoint is determined (white points in fig. F5.2.1).
2. The determined previously midpoints with the endpoints of the edge are inserted into the triangulation (fig. F5.2.2).
3. By the "edge flipping" the obtained at the previous step triangulation is reduced to a state when no one edge crosses the inserted one (fig. F5.2.3).
4. By the same "edge flipping" we obtain a topology when all triangles from the same side of the inserted edge have a common vertex (fig. F5.2.4).
5. Finally, the midpoints and their edges are removed (fig. F5.2.5).

In the presented work the described above algorithm is used to insert as edges of the external boundary of an object as edges of an object hole. After insertion all triangles lying outside the object are removed.

## 6. CONCLUSION

The presented work in fact combines the incremental improvements of the existing ideas and algorithms to obtain the approach that works well in practice. Its high performance and reliability have been proved by working in an existing GIS (figs. F3.1-6 are screenshots of its work). Of course, the idea to use different complexity algorithms according to the complexity of a problem to be resolved is quite obvious and already has been used in several areas of Computer Graphics, but in a majority of cases without an explicit formalization. In the paper such formalization has been proposed. The formulated processing strategies have been introduced in the generalized form and therefore can be applied to any kind of problems in Computer Graphics, not only triangulation ones.

With the processing strategies the conditions of their effective usage have been formulated. The object set, which is typical for GIS, has been explored from the point of view of the chosen object complexity indicator. It has been shown that such object sets meet the formulated conditions and therefore can be effectively processed by the proposed strategies.

Within the framework of the proposed strategies, efficiency of using a fast algorithm, that is too weak to work on one's own, is shown. In many of cases such algorithm can be derived from existing "fully fledged" one by substitution strong conditions and tests with weak ones. The developed example of such algorithm has been introduced in the paper. The complexity of its success prediction makes the presented multi-algorithm approach, which uses this algorithm, a little closer to stochastic methods.

## 7. REFERENCES


[HM83] S. Hertel, K. Mehlhorn. Fast triangulation of simple polygons. Proc. FCT, LNCS 158, 1983, 207-215.

[KET90] X. Kong, H. Everett and G. Toussaint. The graham scan triangulates simple polygons. Pattern Recognition Letters, 11 (1990), 713-716.

[CH91] B. Chazelle. Triangulating a simple polygon in linear time. Discrete Computational Geometry, 6 (1991), 485-524.

[SE91] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. Computational Geometry: Theory and Applications, 1 (1991), 51-64.

[TS91] G. T. Toussaint. Efficient triangulation of simple polygons. The Visual Computer, 7 (1991), 280-295.

[MZ00] M. Lamot, B. Zalik. A Contribution to Triangulation Algorithms for Simple Polygons. Journal of Computing and Information Technology - CIT 8, 2000, 4, 319–331.

[DM04] V. Domiter. Constrained Delaunay triangulation using plane subdivision. Proceedings of the 8th Central european seminar on computer graphics, Budmerice, Slovaška, 19.-21. April, 2004, 105-110.

[YH08] Y. Chen, H. Sundaram. Estimating Complexity of 2D shapes. Arts Media Engineering, Arizona State University, Tempe, AZ 85281, AME-TR-2005-08.


## 8. ANKNOWLEGEMENTS


This work is supported by the Russian Foundation for Basic Research, projects 08-07-00399, 08-07-00468, 08-07-00469.


## About the author


Alexander Emelyanov, PhD, is a researcher of Institute of Computing for Physics and Technology, Protvino, Moscow reg., Russia. His contact email is aiem@rambler.ru.

| total number of objects | number of objects with holes | total number of holes | maximal number of holes in an object |
|---|---|---|---|
| 237295 | 1211 | 4811 | 42 |

**Table T3.1:** Properties of the object set concerning holes.

| number of vertices of the external boundary | number of holes | number of vertices of a hole boundary |
|---|---|---|
| 7.94 | 0,02 | 8.73 |

**Table T3.2:** Properties of the "**a**veraged" object of the set.

| group index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| ratio of the number of objects in the group to the total object number, % | 56 | 12 | 16 | 11 | 4.9 | 0.36 | 0.061 | 0.057 | 0.057 | 0.031 |
| ratio of the total number of object boundary vertices in the group to the total number of such vertices, % | 26 | 8.2 | 12 | 13 | 14 | 1.6 | 0.25 | 0.23 | 0.32 | 0.34 |
| ratio of the total number of the object boundary vertices to the total vertices number in the group, % | 100 | 99 | 98 | 97 | 96 | 86 | 84 | 91 | 94 | 95 |

**Table T3.3:** Statistical properties of objects in the groups.

| group index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| number of the object boundary vertices | 4.89 | 7.13 | 7.76 | 12.82 | 29.34 | 46.02 | 42.52 | 42.23 | 58.51 | 113.97 |
| number of holes | 0.0009 | 0.0086 | 0.017 | 0.044 | 0.15 | 0.81 | 0.78 | 0.57 | 0.26 | 0.45 |
| number of a hole boundary vertices | 8.02 | 8.11 | 7.55 | 8.43 | 8.86 | 9.21 | 14.54 | 8.12 | 14.66 | 14.09 |

**Table T3.4:** Properties of the "averaged" objects of the groups.

| time of processing (s) by the algorithm: | group index | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| "dividing" (1) | 0.77 | 0.23 | **0.34** | **0.44** | **1.0** | **0.22** | **0.049** | 0.055 | **0.051** | 0.098 |
| "cutting ear" (2) | 1.52 | 0.41 | 0.52 | 0.63 | 1.01 | 0.21 | 0.032 | 0.038 | 0.059 | 0.087 |
| "edge inserting" (3) | 1.94 | 0.66 | 1.04 | 1.32 | 1.53 | 0.183 | 0.031 | 0.029 | 0.041 | 0.045 |

**Table T4.1:** The costs of processing the groups of objects by the chosen basic algorithms;
for the "dividing" algorithm results are marked bold if in these cases not all objects have been successfully processed.

| group index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| S | 1 | 1 | 0.99997 | 0.9998 | 0.9983 | 0.9941 | 0.9862 | 1 | 0.9851 | 1 |

**Table T4.2:** Successfulness (S) of the "dividing" algorithm for the object groups.

| group index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| beginning algorithm index | 1 | 1 | 1 | 1 | 2 | 3 | 3 | 3 | 3 | 3 |
| time, s | 0.78 | 0.24 | 0.34 | 0.46 | 0.72 | 0.14 | 0.031 | 0.031 | 0.042 | 0.045 |

**Table T4.3:** Results of the "smart" approach application.

| algorithm | (2) | (3) | "smart" |
|---|---|---|---|
| time w/o holes, s | 4.48 | 7.02 | 3.15 |
| time full, s | 4.91 | 7.45 | 3.56 |

**Table T4.4:** Results of processing the whole object set.

| group index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| ratio of the number of diagonals which have been considered wrong by C5.1.1 to the number of really wrong diagonals | 1.9 | 1.7 | 1.8 | 1.5 | 1.6 | 1.7 | 1.9 | 1.8 | 2.2 | 6.5 |
| ratio of the wrong result number to the total number of tested diagonals | 0.025 | 0.025 | 0.017 | 0.035 | 0.18 | 0.31 | 0.36 | 0.34 | 0.26 | 0.55 |

**Table T5.1.1:** Results of application of condition C5.1.1 for the testing diagonals.

**Figure F3.1**

**Figure F3.2**

**Figure F3.3**

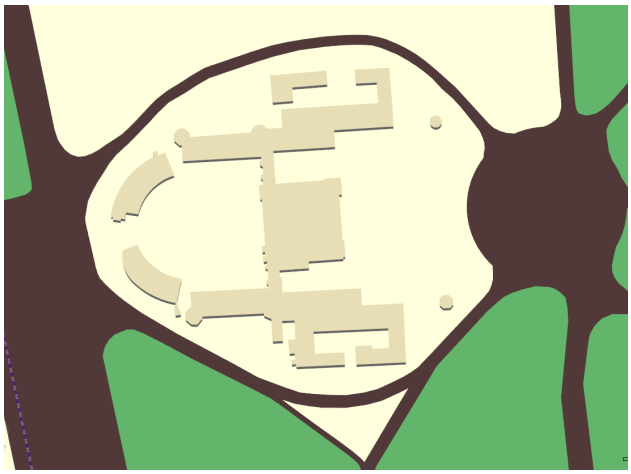


**Figure F3.4**

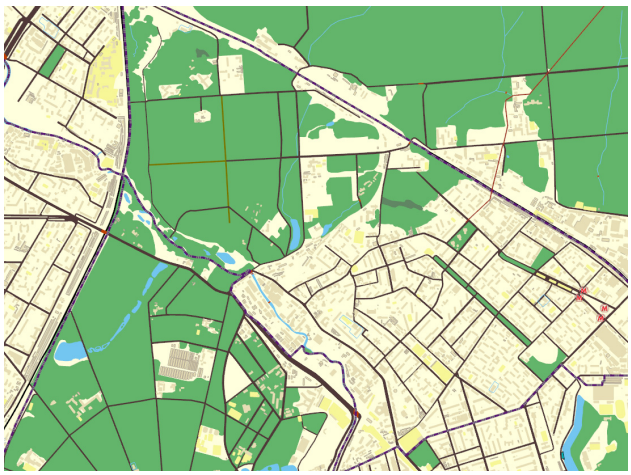
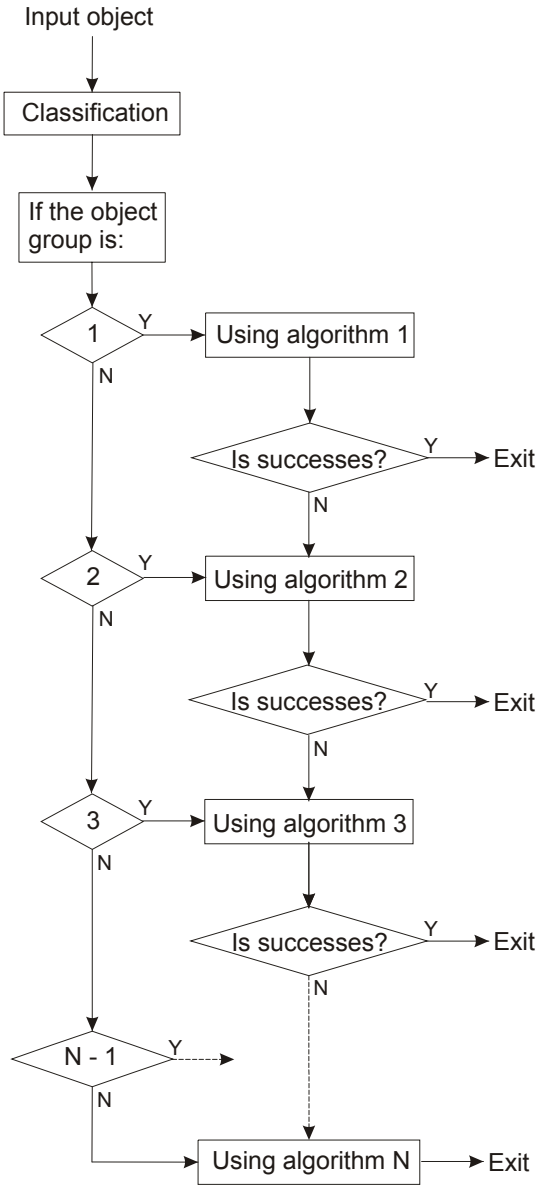
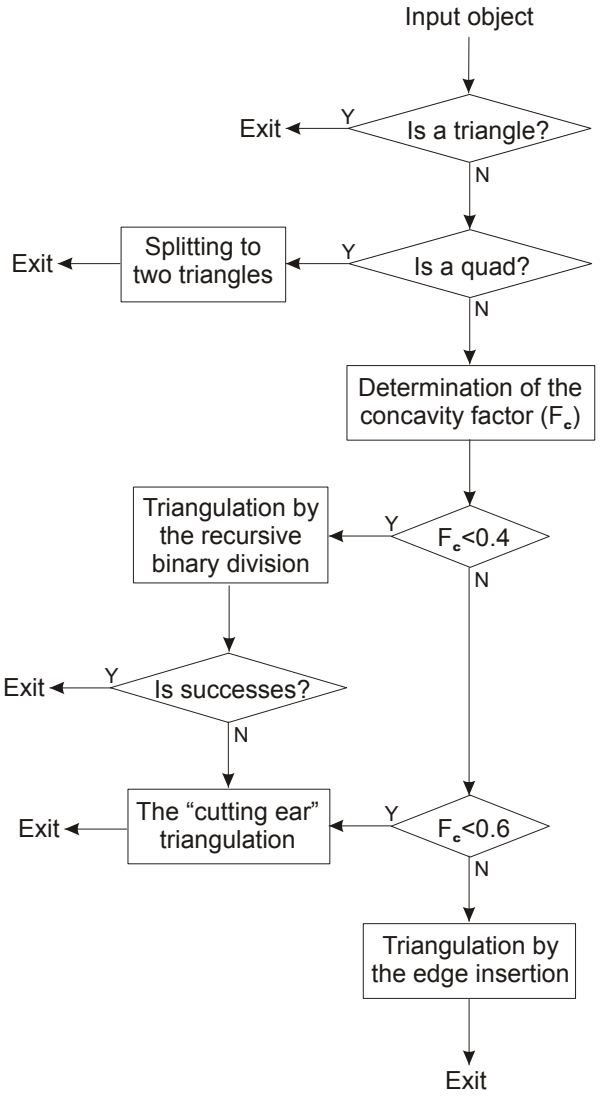


**Figure F3.5**

**Figure F3.6**




**Scheme S2.4.1:** The PSO strategy.
The algorithm with a given index is the beginning for the object group with the same index.




**Scheme S4.2:** The used implementation of the PSO strategy.

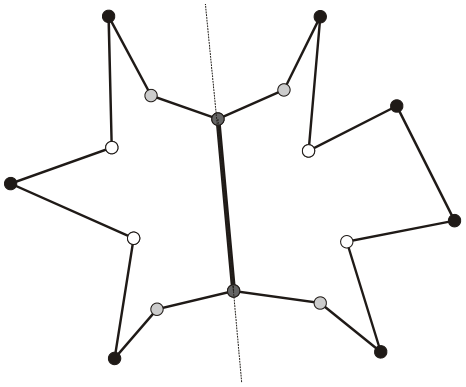


**Figure F5.1.1:** A diagonal that eliminates the maximal number of concave vertices from the consideration.

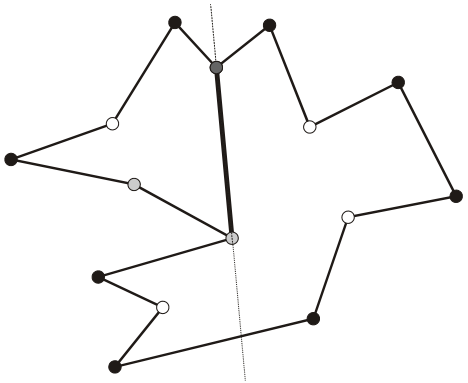


**Figure F5.1.2:** A diagonal that is considered wrong by condition C5.1.1 but in fact is correct.

In these figures different kinds of vertices are denoted by: (●) - convex; (●) - concave which are converted to convex; (●) - concave which are currently skipped ; (○) - spoiling.

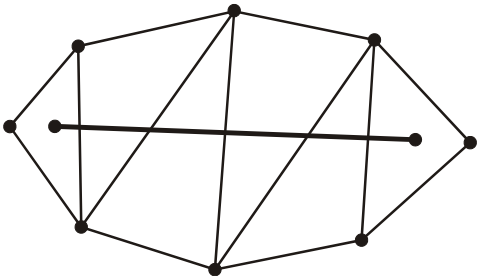


**Figure F5.2.1**

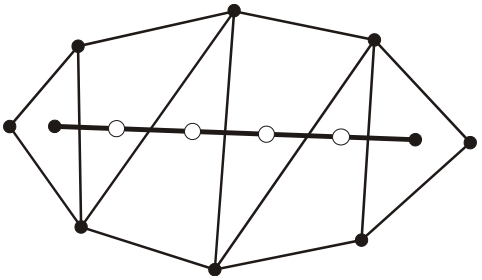


**Figure F5.2.2**

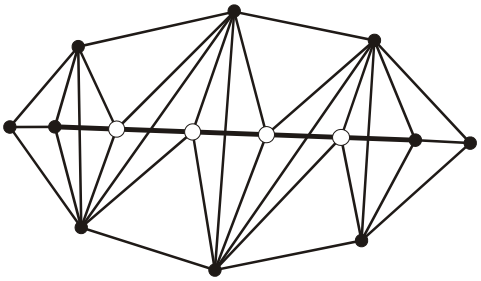


**Figure F5.2.3**

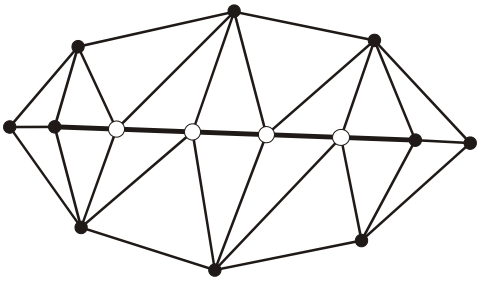


**Figure F5.2.4**

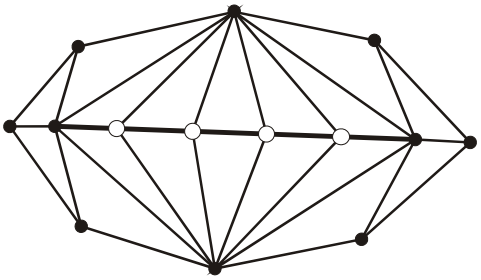


**Figure F5.2.5**

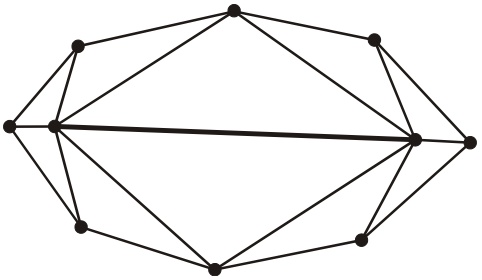


**Figure F5.2.6**