INTERNATIONAL CONFERENCE ON

COMPUTER VISION AND GRAPHICS

With Special Session for Young Scientists
and
Summer School on Image Processing

Zakopane, Poland, September 25-29, 2002

Conference Proceedings
Volume One

Organizers:

Association for Image Processing
Poland

Silesian University of Technology
Gliwice, Poland

Institute of Theoretical and Applied Informatics,
PAS, Gliwice, Poland

Programme committee:

S. Ablameyko (BY)
P. Bhattacharya (US)
E. Bengtsson (SE)
A. Borkowski (PL)
D. Chetverikov (HU)
L. Chmielewski (PL)
J. Chojcan (PL)
R. Choras (PL)
S. Dellepiane (IT)
M. Domanski (PL)
U. Eckhardt (DE)
A. Gagalowicz (FR)
E. Grabska (PL)
H. Heijmans (NL)
J.M. Jolion (FR)
A. Kasinski (PL)
R. Klette (NZ)
W. Kosinski (PL)
R. Kozera (AUS)
H. Kreowski (DE)
Z. Kulpa (PL)
M. Kurzynski (PL)
W. Kwiatkowski (PL)
G. Levina (RU)
L. Luchowski (PL)
R Lukac (SK)
V. Lukin (UA)
W. Malina (PL)
A. Materka (PL)
H. Niemann (DE)
M. Nieniewski (PL)
L. Noakes (AUS)
M. Orkisz (FR)
M. Paprzycki (US)
J. Piecha (PL)
K. Plataniotis (CND)
H. Palus (PL)
D. Paulus (DE)
J. Roerdink (NL)
P. Rokita (PL)
R. Sara (CZ)
V. Skala (CZ)
B. Smolka (PL)
J. Soldek (PL)
G. Stanke (DE)
R. Tadeusiewicz (PL)
V. Valev (BG)
T. Vintsiuk (UA)
J. Zabrodzki (PL)
M. Zaremba (CND)

Local organizing committee:

A. Bal (PL), D. Bereska (PL), P. Lagodzinski (PL), A. Matulewicz (PL), H. Palus (PL), B. Smolka (PL) M. Szczepanski (PL), G. Zajączkowski (PL)

Invited lectures:

A. Gagalowicz
P. Kiciak
R. Kozera
K. Myszkowski
M. Orkisz,
A. Śluzek

Editor

Konrad Wojciechowski
President of Association for Image Processing

Hlavaty,T., Skala,V.: The Brute Force Generator of Triangulations with Required Properties, ICCVG2002 Int.Conf., Poland, ISBN 839176830-9, pp.325-336, 2002

Tomas HLAVATY, Vaclav SKALA
Department Computer Science and Engineering
University of West Bohemia
Univerzitni 22, 306 14 Plzen, Czech Republic
thlavaty@kiv.zcu.cz, skala@kiv.zcu.cz

THE BRUTE-FORCE GENERATOR OF TRIANGULATIONS WITH REQUIRED PROPERTIES[1]

Abstract.
At present many heuristic algorithms searching for triangulations by a given criterion exist. The problem of the heuristic methods is in generating a triangle mesh that approximates the exact solution with an error. The error would be a good property for quality evaluation of this method but the exact size of the error cannot be calculated without the knowledge of the exact solution. Generally the problem of finding the exact solution, which represents a triangle mesh globally optimizing the criterion, has non-polynomial time complexity (NP problem). In this paper we present an algorithm generating a triangle mesh globally optimizing the general criterion. Unfortunately the time complexity is still non-polynomial but some new program techniques are presented here that are used for decreasing the time of the computation.

keywords : *global optimum, hash table, NP problem, triangulation*

1 INTRODUCTION

On a set of points in a plane it is possible to construct a triangulation. The design of the triangulation can vary. Many combinations for connecting the points by lines exist. Of course the condition that the lines do not cross each other is still true. For the choice of the ideal triangulation the, so called, *evaluating function*, which evaluates the design of the individual triangulations is used. In practice the triangulation optimizing the given evaluating function that represents the needed criterion there is selected. The evaluating function can be defined as the function maximizing the minimal angle of triangles (Delaunay Triangulation), sum of the lengths of the edges (Minimum Weight Triangulation) and many others.

[1] This work was supported by the Ministry of Education of the Czech Republic – project MSM 23500005

Hlavaty,T., Skala,V.: The Brute Force Generator of Triangulations with Required Properties, ICCVG2002 Int.Conf., Poland, ISBN 839176830-9, pp.325-33[illegible]02

Searching for the algorithm that constructs triangle meshes optimizing the given criterion is a hard problem. At present, the algorithms are not known for many criteria yet. Generally, it is possible to deal with this problem using brutal force, where all possible triangulations are generated and then the best for the given criterion is selected. Unfortunately, this method has non-polynomial time complexity (NP problem) and so it is not possible to find the solution for a large set of points. In case of a large set of points the heuristic methods are used that approximate the exact solution with some error. This error can be used as a very good quality factor, which characterizes the given heuristic method. Unfortunately, it is not possible to compute without the exact size of the error the knowledge of the optimal solution and so we need to use the brutal force again.

That was the main reason why we started to be interested in the problem of designing the optimal algorithm that generates triangle meshes by the brutal force. We would like to create a tool, which provides sets of triangle meshes optimizing the general criterion. Those meshes can be used as a start point for the quality evaluation of heuristic methods.

2 TRIANGLE INSERTING METHOD

[1] presents several methods for generating all possible triangulations on a set of points in a plane, which uses different attitudes. The *Triangle Inserting method* is suggested that seems to be the best from all. This paper deals with the improvement of the method in question. Therefore we begin with its short description.

In the Triangle Inserting method a triangulation is seen as a set of triangles. Two basic points have been stressed in the algorithm's description. The algorithm has to generate all the possible triangulations that could be constructed on the input set of points. If any triangulations were omitted we could not be sure that the found triangulation is optimal for the selected criterion. On the other hand the algorithm does not have to generate duplicities. Any duplicate triangulations decrease the time delay and because the algorithm falls into the category of NP problems, it will be reflected on the time of the computation greatly.

The starting point of the algorithm is to find the convex hull. It is no problem because many algorithms with polynomial time complexity exist (Incremental algorithm, Gift Wrapping Algorithm, Divide and Conquer algorithm, QuickHull algorithm). In our case the edges of the convex hull represent a polygon (the, so called, *boundary polygon*) surrounding a region into which triangles have to be inserted for creating a correct triangulation. The procedure of the algorithm is very simple. An edge is chosen from the boundary polygon and then the, so called, *empty triangle* that contains the selected edge and that is inside of the boundary polygon is inserted. The empty triangle means a triangle whose vertices are any points from the input set and which contains no other points from this set. By inserting the triangle, the boundary polygon will be changed and will demark the original region without the region of the inserted triangle. From this new polygon an edge is selected again and another empty triangle, which contains that selected edge and which is included inside the new region, is selected. That procedure is repeated until the correct triangulation is created. It is in the same moment as the boundary polygon is an empty triangle.

It could seem that only one triangulation will be created by the described procedure. But it is not right. Determinism is hidden in the procedure. When an edge is selected from the

Hlavaty,T., Skala,V.: The Brute Force Generator of Triangulations with Required Properties, ICCVG2002 Int.Conf., Poland, ISBN 839176830-9, pp.325-336, 2002

boundary polygon then an empty triangle shall be inserted. There can be more of these empty triangles (see Figure 1). Different results will be obtained by selecting the individual triangles.

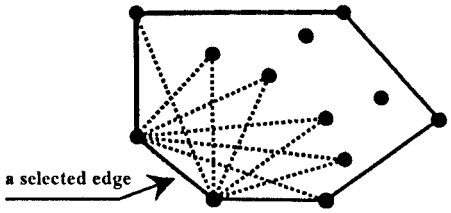

Fig. 1 An example of more combinations for inserting an empty triangle

Because we need to generate all the possible combinations of triangulations, we have to use all the potential empty triangles. The number of branches is the same as the number of the empty triangles that can be inserted. The resulting structure describing the algorithm can be represented as a tree. On the Figure 2 a simple example of such tree is shown.

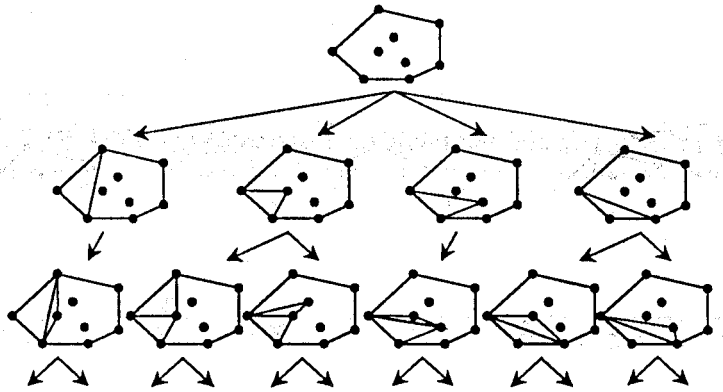

Fig. 2 An example of more combinations for inserting an empty triangle

The root node of the tree is only represented by the edges of the convex hull and the leaves of the generated tree are represented by the required triangulations. The number of inserted triangles in the generated triangulation corresponds to the level of the given node in the tree. Because the number of triangles in an arbitrary triangulation is always the same and equal to the constant N_T, therefore the maximal number of levels in the tree can be N_T.

$$N_T = 2 \cdot (N-1) - N_{ch}, \tag{1}$$

where N is the number of the points in the input set of points and N_{ch} is the number of points (or edges) in the convex hull.

The goal of the algorithm is only to select the best triangulation for the given criterion, which is represented by the evaluating function. Therefore it is not necessary to know all the triangulations. The triangulations can be evaluated continuously and we must only remember the best one. This approach is advantageous because it has very small memory requirement.

The algorithm uses a depth-first based searching for the global optimum (see [6], [7]) where the tree is generated from the left side to the right side. A breadth-first based searching algorithm is possible to use too, but in our case this algorithm is not ideal. It has bigger memory requirement and an exploration of nodes in the tree has no advantage because it is not possible to minimize the needed tree by this approach. We have to generate the same tree as in the origin method.

Hlavaty,T., Skala,V.: The Brute Force Generator of Triangulations with Required Properties, ICCVG2002 Int.Conf., Poland, ISBN 839176830-9, pp.325-3 02

3 IMPROVEMENT OF THE METHOD

In [1] and [2] techniques for speeding up the Triangles Inserting method are presented. These techniques are based on a distribution of the algorithm or on using a preprocessing and the basic complexity of the algorithm is not changed.

In this chapter, an improvement of the method is presented, which just decreases the time complexity.

3.1 The general view

Let's try to look at the nodes of the generated tree by the Triangle Inserting method from a different view. The node represents a part of a generated triangulation (see Figure 3) an, so called, *incomplete triangulation* that can be divided into two regions by a, so called, *boundary polygon P*:

- $\Omega_{complete}$ – the completed part of the triangulation (the area of the inserted triangles).
- Ω_{empty} – the region that still has to be triangulated in order to create the triangulation of a input set of points.

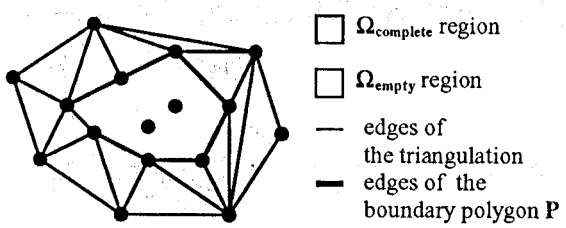

Fig. 3 An incomplete triangulation

The boundary polygon P is the set of edges that divides the generated triangulation to $\Omega_{complete}$ and Ω_{empty} regions.

The root and the leaves of the tree are special cases. The root of the tree represents as incomplete triangulation, in which no triangle has been inserted yet. There are only the edges of the convex hull and, because this edges surround the region that could be triangulated, it is possible to say that the edges of the convex hull are the edges of the boundary polygon P. The $\Omega_{complete}$ region is empty (no triangles have been inserted) and the Ω_{empty} region is the inner area of the convex hull.

In the leaves of the generated tree is the opposite situation. The leaves represent the correct triangulations of the input set of points. Using the terminology defined above there is only the $\Omega_{complete}$ region (the inner area of the convex hull). The Ω_{empty} region is an empty area and the boundary polygon P contains no edges.

From this view all the nodes (incomplete triangulations) are divided into two regions by the given boundary polygon. If we try to generate a tree we can notice that some nodes have the same boundary polygon and both the regions (see Figure 4).

From the point of view of the Triangle Inserting method these nodes can be regarded as roots of sub-trees. These sub-trees always have the same structure because the Ω_{empty} region and the boundary polygon P is the same too.

Hlavaty,T., Skala,V.: The Brute Force Generator of Triangulations with Required Properties, ICCVG2002 Int.Conf., Poland, ISBN 839176830-9, pp.325-336, 2002

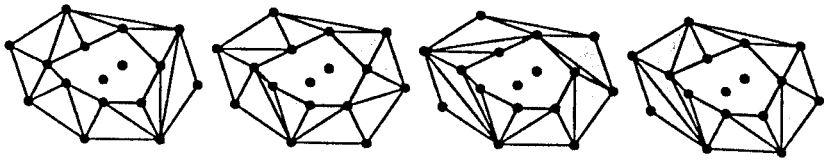

Fig. 4 Incomplete triangulations with the same boundary polygon and separation into regions

What is the point of this fact in our case? The main goal is to find a triangulation T that optimizes the evaluating function $w(T)$ globally. We can write it as:

$$T_{opt} = opt(w(T_i))_{\forall i}, \tag{2}$$

where T_i is the set of edges of the generated triangulation, w is the evaluating function and opt is the function, which returns the set of edges T_i with the best evaluation.

Let us suppose that the next condition is valid for the evaluating function w corresponding to the required criterion of the resulting triangulation:

$$w(T) = w(T_1) + w(T_2), \tag{3}$$

where T_1 and T_2 are sets of edges with the following properties:

$$\begin{aligned} T_1 \cup T_2 &= T, \\ T_1 \cap T_2 &= 0. \end{aligned} \tag{4}$$

Then we can say that if triangulation T was divided into two sets of edges T_1 and T_2 the sum of evaluations T_1 and T_2 is the same as the evaluation of the whole triangulation T.

We can use this property of the evaluation function for the triangulation of the boundary polygon P from Figure 4. Searching for the global optimum, where the starting point is an incomplete triangulation can be described as:

$$T_{opt} = T_{complete} \cup opt\left(w\left(T^i_{empty}\right)\right)_{\forall i}, \tag{5}$$

where the functions are the same as in the previous equation, $T_{complete}$ is the set of edges included in the original incomplete triangulation and T_{empty} is the set of edges found by the triangulation of the boundary polygon P of the original incomplete triangulation.

Since the started condition is valid the result of *opt* function has to be the same in case of all the incomplete triangulations from Figure 4. This fact can be used during the generation the whole tree. If the edges of the best triangulation of the boundary polygon P were remembered, this result can be used in following nodes with the same boundary polygon P (see Figure 5). In all the occurrences of this polygon P, it is not necessary to generate the whole sub-tree to find the triangulation optimizing the required criterion. The sub-tree can be replaced by one new node in which all the edges found in the first case of the triangulation of the given boundary polygon P are inserted.

We still work with one polygon but in the tree there are many different shapes of the boundary polygons. Remembering all the triangulations of individual polygons, we could decrease the robustness of the tree. In the following text the term *local optimum* will stand for the set of edges, which triangulates a boundary polygon P and which optimizes the triangulation of the polygon by the required criterion.

Remembering and using local optimums is the key idea of improving the Inserting triangles method. Any unneeded triangulations that cannot be the optimal solution are omitted (based on the knowledge of the local optimums) and so the time of the calculation is decreased.

Hlavaty,T., Skala,V.: The Brute Force Generator of Triangulations with Required Properties, ICCVG2002 Int.Conf., Poland, ISBN 839176830-9, pp.325-330, 2002

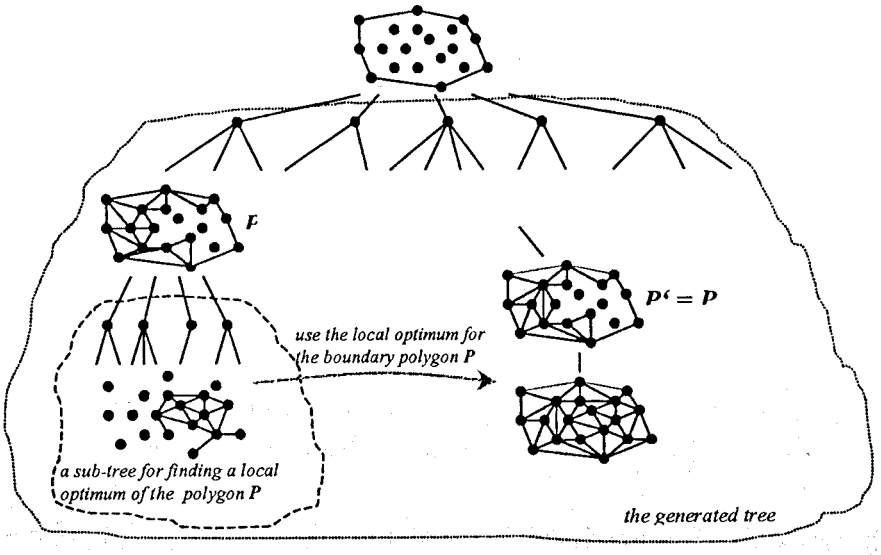

Fig. 5 An example using a local optimum in the generated tree

Let us note that in practice there are not just so simple trees as it is shown on Figure 5. The boundary polygon P can surround more non-continuous regions (see Figure 6). If we explored the Triangle Inserting method in more detail we find that the similar situation has been solved there. The boundary polygon P is viewed as a set of the polygons that surround the individual continuous regions.

This view is not change. It is means that we do not try to find a local optimum of a whole boundary polygon but we only work with the edges enclosing individual continuous regions.

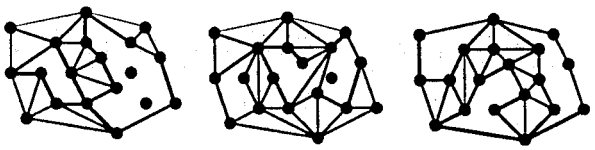

Fig. 6 An examples of the non-continuous boundary polygons

3.2 Hash table

Unfortunately, the boundary polygons that will appear multiply times in the generated tree cannot be determined in advance. When a local optimum of a boundary polygon P is found, we cannot say, if we should remember it. A good idea is remember all the local optimum found before but this approach has any pitfalls.

The first problem is in the number of local optimums. Every local optimum is connected with a node of the generated tree. In the worst case (in the tree there are no duplicated boundary polygons) the number of the local optimums can maximally be the same as the number of nodes. Because generally the number of the nodes in the tree grows exponentially by inserting points into the input set we get huge sets of the local optimums.

From the reason of quick searching in the set of all the local optimums, which were found, the proper data structure has to be selected. On the ground of the results in [3] we used a hash table. The local optimums separate into clusters. The cluster in which given local optimum is inserted is nominated by a hash function that returns the index of the given cluster. The basic structure of the hash table is shown on the Figure 7.

Hlavaty,T., Skala,V.: The Brute Force Generator of Triangulations with Required Properties, ICCVG2002 Int.Conf., Poland, ISBN 839176830-9, pp.325-336, 2002

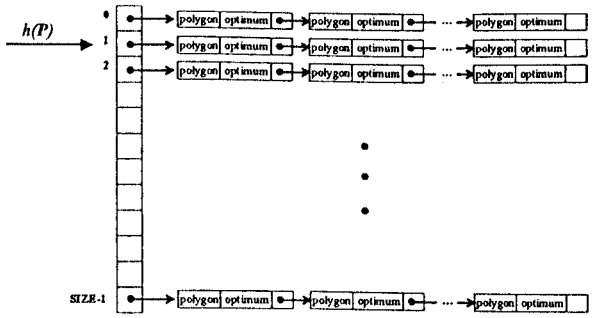

Fig. 7 The basic structure of the hash table

It is obvious from the picture that the saved local optimums are connected by a one-way linear list in the clusters and the number of the clusters is the same as the range of the hash table (variable *SIZE*). A quick approach is needed to all the save items so the clusters has to fulfill the next condition:

The clusters have so short length as it is possible.

The whole range of the hash table is used for saving the local optimums.

The choice of the hash function influences the both conditions. If an unsuitable hash function would be selected the needed time searching in hash table could be very long. Our selected hash function is on the Figure 8 (written in C):

```
int hash (unsigned int *bin_P, int numP) {

        unsigned int index_hash = 111 * numP;
        for (i = 0; i < numblocks; i++) {
              index_hash = _rotl(index_hash,1);
              index_hash += bin_P;
        }
        unsigned int tmp_hash = index_hash;
        for (i = K; i < 32; i += K)
              tmp_hash ^= index_hash >> i;
        return (tmp_hash & MASK_VAL);
}
```

Fig. 8 The hash function

The input parameters of the function are the number of edges in boundary polygon P (*numP*) and a binary vector represents the edges in polygon P (*bin_P*). The binary vector means the vector of bits where each bit is connected with one edge from all, which are possible to construct. The value of bit means if the edge is in polygon or is not (see Fig. 9).

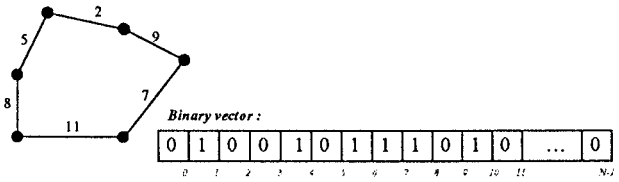

Fig. 9 An example of the polygon representation by the binary vector

The number of bits in the binary vector is equal the maximal number of (non-oriented) edges N that can be constructed on the input set of edges:

$$N = \frac{n \cdot (n-1)}{2}, \tag{6}$$

where n is number of points. In hash function that binary vector is represented by an array of unsigned integers.

Hlavaty,T., Skala,V.: The Brute Force Generator of Triangulations with Required Properties, ICCVG2002 Int.Conf., Poland, ISBN 839176830-9, pp.325-336, 2002

The algorithm of the hash function is composed from two cycles. Step by step first cycle takes the blocks of 32 bits (type *unsigned int*) representing boundary polygon and calculates from them a result value in range of *unsigned int* type (32 bits) by an arithmetic summation and a left rotation 1 bit far with a carry (the function *_rotl*).

Second cycle transfers the result value of first cycle to the range from *0* to *SIZE-1* by the operation *exclusive or* (operator ^). The variable *SIZE* has to assume the value:

$$SIZE = 2^k, \qquad k = 1,2,...32, \tag{7}$$

The variable *K* in hash function means the same as in the previous equation and the variable *MAX_VAL* is equal value *SIZE – 1*.

This hash function was designed on the base of our knowledge and many experiments. May be our function has not ideal character but to found a better function is very hard. On the ground of experiments (they will be discussed in a chapter with some results of our experiments) we can say that it is sufficient.

3.3 Memory requirement

It was said this problem has a non-polynomial complexity. The number of nodes in the generated tree grows exponentially and because the number of found local optimums is connected with the number of the nodes we can expect that it will grow exponentially too. On the other hand the memory capacity is limited. Therefore we have to assume that all the local optimums cannot be saved in hash table.

It is need to be interested in a mechanism helping to determine any local optimums that could be removed from the hash table. Unfortunately it is not possible to say about the local optimum if could be used during generating next nodes of the tree. Therefore we can use only a mechanism based on a general method.

We selected a method based on a chronological order in our case. It means the found local optimums are arranged in a list by time inserting into the hash table. If a memory is needed to release the first items from the list are forgotten. The new local optimums are added on the end of the list. This list can be implemented as a one-way linear list (see Figure 10).

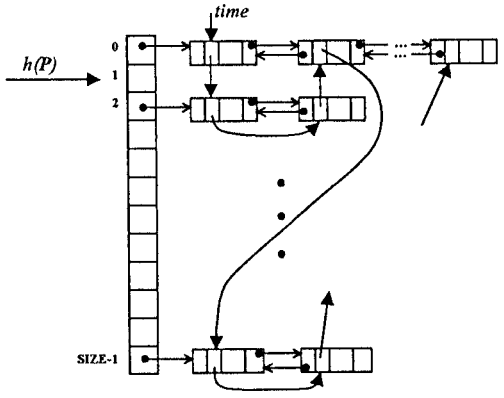

Fig. 10 The hash table with a list created according to time of inserting the local opt. into table

The advantage of chronological mechanism is possible to check in the next chapter with results.

Hlavaty,T., Skala,V.: The Brute Force Generator of Triangulations with Required Properties, ICCVG2002 Int.Conf., Poland, ISBN 839176830-9, pp.325-336, 2002

4 RESULTS

At first the quality of our hash function is need to check. We could get worse results if we have a bad function. The method was tested on a few distil sets of points by a generator of random numbers with the uniform distribution. We have to warn that the number of generated triangulations is dependent on the number of points and their topological positions in a plane. In two different sets with the same number of points we can construct different numbers of the triangulations and of course the time of computation is different. The results in the graphs only can be taken as orientation values.

Let us return to the evaluation of our hash function. The uniform distribution in hash table (the lengths of the clusters shall be the same approximately) and the usage of the whole range of the hash table have to be emphasized. Figure 14 (see Appendix) shows how the character of our hash function is changed for two different values of the parameter *SIZE*. All the clusters of the hash function are recorded on the x-axis. On the y-axis there is the number of the saved local optimums in the given cluster (see 14a) and there is maximal length of the clusters during the calculations (see 14b). The figure only shows a dependence on these two factors for one selected set of points. We of course tested many different sets and we can say (from the reason of space capacity no other graph are not shown here) that the characters of the graphs do not change significantly.

On the ground of Figure 14 we can tell more about hash function that it has a tooth-like character. Generally, it is very hard to select and specify the input parameters of the hash function and so this character of the hash function is sufficient in our case. The other significant effect is that the lengths of the clusters are decreasing when the size of the hash table is growing (see Figure 11).

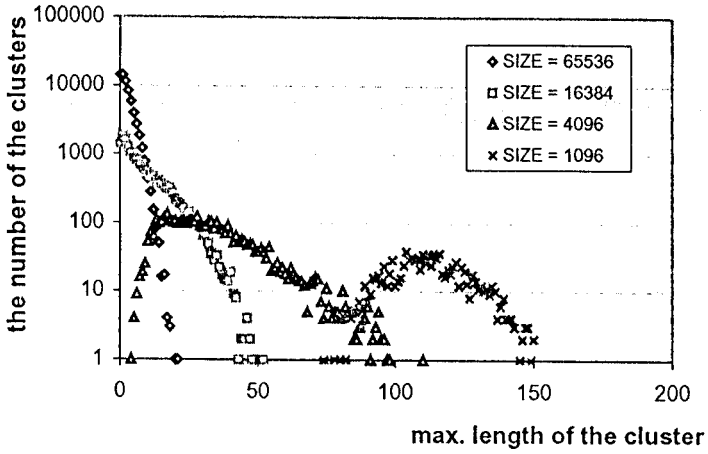

Fig. 11 The histogram showing the number of clusters with the given maximal length (a set with 18 points)

The figure shows a histogram describing the change of the cluster lengths for several different values of the parameter *SIZE*. Our goal is to comb the clusters very quickly therefore we need to get a short length of the clusters in hash table. We have to perceive that the results of a selected set with 18 points are only in the graph. The values in the graph will be changed for another different sets of points (the dependence on the number of points and on topological position of points is here again). In our case the important property is the lengths

Hlavaty,T., Skala,V.: The Brute Force Generator of Triangulations with Required Properties, ICCVG2002 Int.Conf., Poland, ISBN 839176830-9, pp.325-336, 2002

of clusters. The clusters are shorter when the size of the hash table is bigger. Therefore we would set the size of the hash table on the value so big as it is possible. On the other hand we have to perceive that some memory is needed for creating the empty hash table where the pointers on the clusters are saved. We have to combine these two facts and set the size on the constant value 2^{16} ($SIZE = 65536$).

The next thought that has to be given to the usage of the chronological mechanism removing the local optimums in the hash table. The variable MAX sets the maximal number of the local optimums, which can be saved in the hash table. The influence changing the value of this variable on the length of the clusters can say if the selected mechanism has good properties from the view of our hash table. We tested many sets of points and an example of the result is shown on Figure 12 (a set including 18 points).

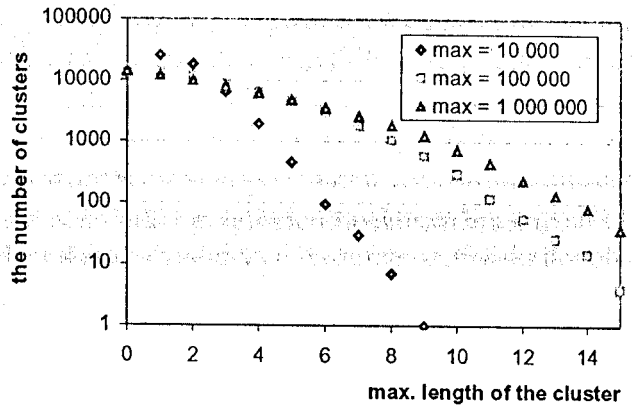

Fig. 12 The histogram showing the number of clusters with the given maximal length for different maximal number s of items in hash table (tested on a set with 18 points)

The mechanism of the chronological removing is only used in the case setting 10 000 and 100 000 maximally saved items on the figure. We can observe that the influence on the length of the clusters is very good. It is true for the other results too. Unfortunately the results are not possible to draw in a simple and comprehensive graph and so we only can say that all the curves have a similar character as on Figure 12.

The most important graph is a graph describing a dependence of the time of computation on the number points in an input set (see Figure 13). We have to perceive that the time is not only dependent on the number of points. It is dependent on the topological position of points in a plane too. Therefore we only have to look on the values in the graph as on some approximate values. They were computed as the approximation of the values, which were obtained by testing several sets with the same number of points on the computer DELL, PentiumIII, 2x650MHz, 1GB RAM, OS Windows 2000.

The time of computation for the original Triangle Inserting method is shown in the graph too. By comparing the resultant curves in the graph we can say that our improved method offers the results faster and so it is possible to use it for bigger sets of points. On the other hand the maximal number of the saved local optimums in hash table has an influence on the time too (see Figure 13). When the number of maximal local optimums (saved in the hash table) is growing the time of computation is decreasing. We need more memory with growing the maximal number of local optimums in the hash table (for example we need about 200MB for 1 000 000 saved local optimums in our representation).

Hlavaty,T., Skala,V.: The Brute Force Generator of Triangulations with Required Properties, ICCVG2002 Int.Conf., Poland, ISBN 839176830-9, pp.325-336, 2002

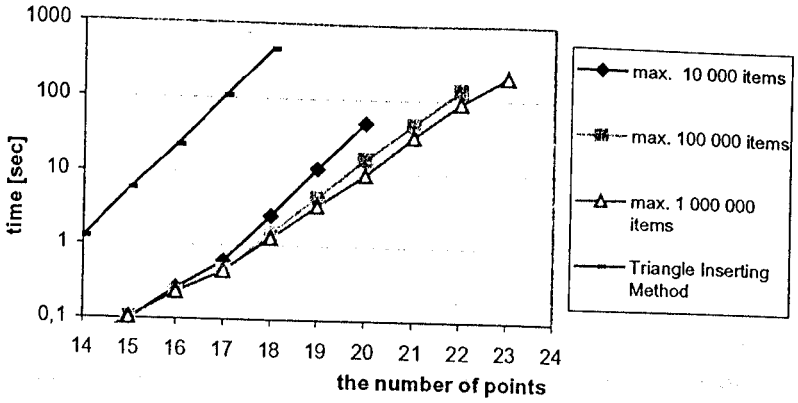

Fig. 13 The independence of the time (which is needed for computation) on the number of points

5 CONCLUSION

Improving Triangles Inserting method is presented in this paper. The main goal of the work has been to decrease the time of computation. From the Figure 14 we can obtain an opinion about our improved method. The limitation of the maximal number of points for which we can find a solution in real time has been moved up but unfortunately we cannot forget that it is still NP problem. Generally, it is not possible to break NP complexity. We only can try to move the limit of points more up by using some other techniques (as preprocessing and distribution - see [1]).

Perhaps the usage of this method is limited in practice. We only can use it when the input sets do not contain a lot of points. On the other hand the information that the found triangulations are exact and do not exist better result can be very important in some situations. Because practically this method can be used for a general criterion of the triangulation, the other possibility of the practical usage exists too. If a programmer designed a heuristic method for a triangulation with a special properties and no other method exists yet, our method can be used for getting exact results and so the programmer can get a feedback about quality of his or her heuristic method.

REFERENCES

[1] T. Hlavatý, *Generátor trojúhelníkových sítí zadaných vlastností brutální silou*, Diploma Thesis, supervisor: V. Skala, University of West Bohemia, Pilsen, 2000.

[2] T. Hlavatý, V. Skala, *Generátor trojúhelníkových sítí zadaných vlastností brutální silou*, Technical Report No. DCSE/TR-2002-09, University of West Bohemia, Pilsen, 2002.

[3] J. Hrádek, M. Kuchař, V. Skala, *Hash Functions and Triangular Mesh Reconstruction*, Computers & Geoscinces, submitted for publication.

[4] C. B. Barber, D. P. Dobkin, H. Huhdanpaa, *The Quickhull algorithm for convex hulls*, ACM Transactions On Mathematical Software, 22, 469-483, 1996.

Hlavaty,T., Skala,V.: The Brute Force Generator of Triangulations with Required Properties, ICCVG2002 Int.Conf., Poland, ISBN 839176830-9, pp.325-336, 2002

[5] T. Lambert, *Convex Hull Algorithms*, http://www.cse.unsw.edu.au/~lambert/projects.html.

[6] S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, 1995.

[7] Ch. J. Thorton, Artificial Intelligence Through Search, on-line publication, 1997.

APPENDIX

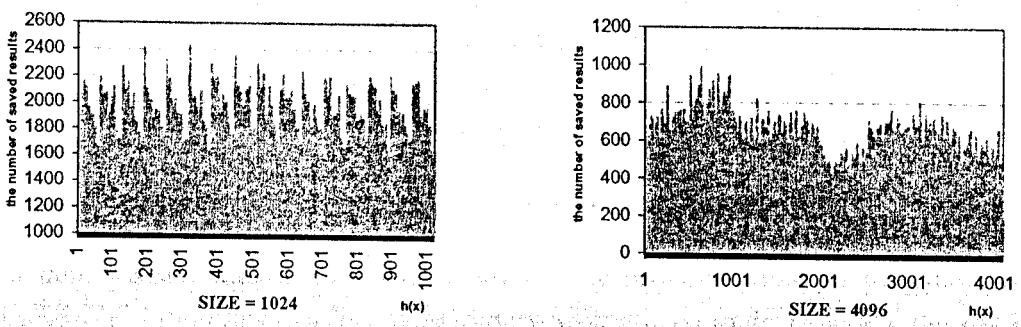

Fig. 14a The number of saved results in the individual clusters of the hash table (set of 18 points)

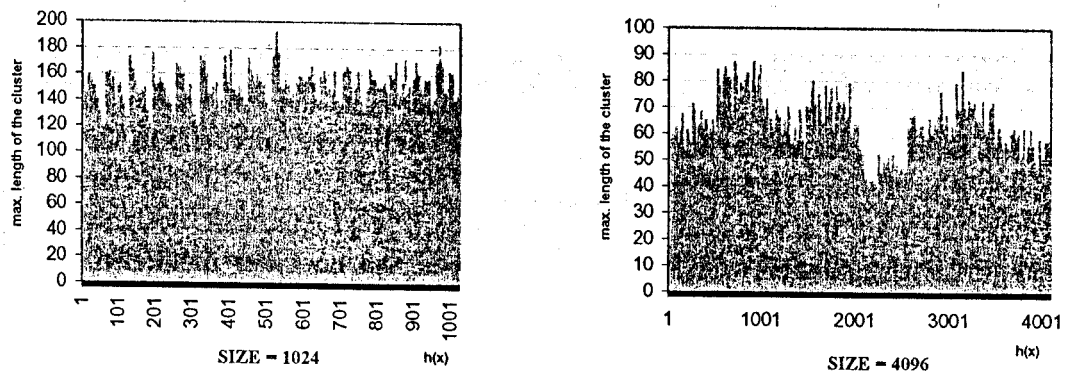

Fig. 14b The number of saved results in the individual clusters of the hash table (set of 18 points)