TRIANGULAR MESH DECIMATION AND UNDECIMATION FOR ENGINEERING DATA MODELLING

Sebastian Krivograd[1], Borut Žalik[1], Franc Novak[2]

[1]University of Maribor, Faculty of Electrical Engineering and Computer Science, Maribor, Slovenia

[2]Jožef Stefan Institute, Ljubljana, Slovenia

Key words: engineering data modelling, triangular meshes, mesh decimation and undecimation, hash table.

Abstract: Triangular mesh decimation is the process that uses local operations on geometry and topology to reduce the number of triangles in a triangle mesh. Triangular meshes are used in many engineering applications where simple interpolation of discrete data replaces continuous and complex model of reality. Furthermore, triangular meshes are standard input to numerical analysis tools based on Finite Element Method. Manipulation with large triangular meshes is a bottleneck in engineering applications hence appropriate simplifications are needed. Apart from intuitive manual techniques, mesh decimation process is an attractive alternative providing optimal computer based solutions. The paper presents a fast algorithm for decimation of triangular meshes using vertex elimination approach. To speed-up the search for the vertex to be removed, a hash table is applied. Presented solution runs in linear time and is suitable for different applications in practice. Its usefulness is increased by an introduction of an undecimation, i.e. a reverse process restoring gradually the initial triangular mesh. An illustrative example from the analysis of a power line electric field is given.

Modeliranje inženirskih podatkov s poenostavljanjem in rekonstrukcijo trikotniških mrež

Ključne besede: modeliranje inženirskih podatkov, trikotniške mreže, poenostavljanje in rekonstrukcija trikotniških mrež, sekljalna tabela.

Izvleček: Trikotniške mreže pogosto uporabljamo v inženirskih aplikacijah, predvsem ko želimo interpolirati diskretne odbirke kompleksnih zveznih procesov. Trikotniške mreže so prav tako standarden vhod za različne numerične analize, ki temeljijo na metodi končnih elementov. Manipulacija z zelo velikimi trikotniškimi mrežami pa predstavlja ozko grlo pri inženirskih aplikacijah, zato iščemo enostavnejše a še zmeraj dovolj verne trikotniške mreže. Procesu, ko z uporabo lokalnih operacij nad geometrijo in topologijo podane trikotniške mreže zmanjšamo število vozlišč in trikotnikov, pravimo poenostavljanje trikotniške mreže. Postopek iskanja najprimernejše trikotniške mreže je običajno prepuščen uporabniku in temelji na njegovi intuiciji, zato predstavlja računalniško podprt proces poenostavljanje zanimivo alternativo. V članku predstavimo učinkovit algoritem za poenostavljanje trikotniških mrež, ki temelji na odstranjevanju oglišč. Da bi pohitrili odločitev, katero izmed oglišč je najprimernejšo za odstranitev, uporabimo sekljalno tabelo. Predstavljena rešitev deluje v linearnem času in je primerna za različne aplikacije v praksi, predvsem tam, kjer potrebujemo hiter odziv sistema. Uporabnost pristopa povečamo z možnostjo postopnega vračanja odstranjenih oglišč - z rekonstrukcijo. Delovanje algoritma prikažemo z ilustrativnim primerom poenostavljanja trikotniške mreže električnega polja daljnovoda.

1 Introduction

Although natural phenomena are continuous, engineers normally measure their values only at some discrete measurement positions. The values at other positions are then calculated by interpolation. In the applications where scalar values (e.g., temperature, see level, electric field, tension) are measured in a plane, the most suitable interpolation results in a triangular mesh /1/. Of course, the denser the mesh, the better interpolation can be constructed. Unfortunately, a huge number of measured points may cause problems in manipulation with the corresponding triangular meshes resulting in slow response time and considerable computer memory requirement. This is especially critical, when a triangulation mesh is used as an input into numerical analysis based on FEM (Finite Element Method) /2/. Furthermore, by the widespread use of the internet, large triangular meshes require long transfer time between collaborating parties. Because of that, triangulation meshes are frequently simplified to made an acceptable compromise between the accuracy and the system limitations. The main idea stems from the fact that a triangular mesh can be simplified in regions with small or no variation of the scalar values. This task is usually performed manually using experience and intuition of a user. An alternative is the use of the so-called mesh decimation algorithms, which simplify the triangular mesh automatically.

Triangular mesh decimation approaches, developed so far, can be classified according to the elements they are taking from the mesh (i.e., vertices, edges, or triangles) /3, 4/:

- **Vertex decimation methods** are the most frequently used and are based on Schroeder simplification algorithm /5/. The vertices are evaluated, and they are incrementally removed from the mesh according to their importance. Various techniques have been proposed, and they differ on how vertices are evaluated and what type of triangulation is required (see /3/ for an overview).

- **Edge decimation methods** eliminate edges, which have been evaluated already /6/. The edge being removed is replaced by a vertex. Triangles, which degenerate to edges, are removed. One of the best edge decimation methods is based on quadric error metrics /7/.
- **Triangle decimation methods** eliminate one or more triangles. Although theoretically feasible, practical approaches using this possibility have not been reported.

In this paper we present an algorithm that combines two mesh decimation approaches. Franc and Skala used a hash table in a parallel environment for speeding-up the search of the most suitable vertex to be removed /8/. They combined the vertex and edge removal in the following way. At first the most suitable vertex is selected, and then among the edges incident to that vertex, the shortest one is contracted. Our algorithm uses the pure vertex decimation similar to the one proposed by Schroeder /5/, but using the hash table as the acceleration technique. The heuristic approach for creation of a hash table for engineering applications have been also developed. Beside that, our algorithm is equipped by an undecimation feature, i.e. by an incremental reconstruction possibility of the original mesh. The algorithm is suitable for engineering data modelling. Low time and space complexity make it even candidate for embedded system applications.

2 The decimation algorithm

The proposed algorithm for triangular mesh decimation is briefly sketched as follows:

1. Evaluate all vertices according to a chosen evaluation criterion and arrange them into a hash table.
2. Select the most suitable vertex using the hash table (for example, vertex v_i in Figure 1a).
3. Remove the vertex from the triangular mesh.
4. Remove all triangles incident on the removed vertex (Figure 1b).
5. Triangulate the area from where the triangles have been removed (see Figure 1c).
6. Re-evaluate vertices incident to the removed vertex (vertices v_j, v_k, v_l, v_m, v_n in Figure 1c).
7. Return to step 2 until the termination criterion is met.

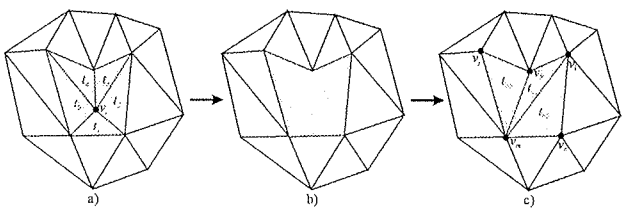

Figure 1: Vertex decimation

2.1 Evaluation of vertices

Before the decimation process starts, the vertices have to be evaluated. Let us take a look at Figure 2 where the triangles with small changes of scalar values in their vertices can be reduced without much spoiling the presentation.

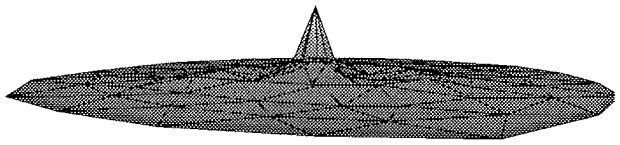

Figure 2: A triangular mesh

The evaluation of vertices is done by investigating the neighbourhood of the vertex under consideration. Different criteria can be applied. Let us mention just two of them:

- Vector $v_{i,j}$ connecting the examined vertex v_i and its neighbouring vertex v_j is formed. The angle between this vector and xy plane is calculated. The average value of all angles defined by vertex v_i is used as the evaluation value ev_i.
- The average difference of scalar values between the examined vertex v_i and the neighbouring vertices is used as the evaluation value ev_i.

Better results are obtained by the first criterion. That can be confirmed by observing Figure 3a and Figure 3b, where the scalar value against the two neighbours is the same on both figures. If the first evaluation criterion is applied, the situation in Figure 3a gives bigger evaluation value ev_i than in Figure 3b. Applying the second criterion gives the same evaluation value ev_i. In practice, however, the criterion with average difference of scalar values is normally used.

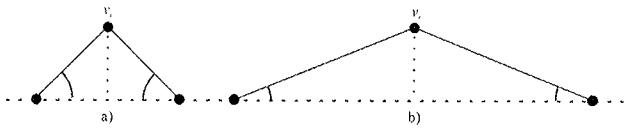

Figure 3: Evaluation of vertices

2.2 Selection of a vertex to be removed

The strategy is to remove vertices that cause the smallest change in data representation. Therefore, the vertex with the smallest evaluation value ev_i is selected and removed from the mesh. The evaluation values ev_i of the neighbouring vertices are changed and must be estimated again. In the next iteration, the algorithm searches for the next vertex with the smallest ev_i. It can be selected easily by walking through the set of the remaining vertices, and selecting the one with the smallest ev_i. Unfortunately, this method works in $O(n^2)$ time and considerably slows down the algorithm. The second possibility, sorting at first all vertices according to ev_i and adjusting their position in the sorted array according to the changed ev_i works in expected $O(n \log n)$ time. The constant expected time complexity can be achieved by introducing a hash-table /8/. However, in this case, the condition of selecting the vertex with the smallest ev_i has to be slightly relaxed. The vertices are

organised in the hash table according to their evaluation values ev_i. Figure 4 schematically shows the structure of the hash table. Vertices in each interval are stored in a FIFO queue.

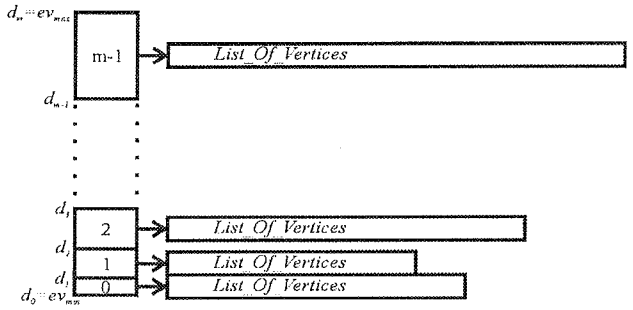

Figure 4: Arranging vertices into hash table

The hash table is usually organised according to the expected distribution of the input data. In our application, the heuristics for linear, quadratic and exponential expected data distributions are prepared. It is important to note that during the decimation process new evaluation values of ev_i can become greater than maximal evaluation value determined before the hash table has been formed. Therefore, we have to add additional entry to the hash table accepting such possible cases. Having the hash table, the next vertex to be removed from the triangular mesh is now obtained easily. The algorithm always selects the first vertex from the lowest non-empty FIFO queue and removes it. The neighbouring vertices of the removed vertex are evaluated again and inserted in the corresponding interval at the end of the FIFO queues. In that way it is prevented to perform decimation only locally. The hash table assures constant time complexity of this part of the algorithm. By storing a list of neighbouring vertices at each vertex of the mesh, the neighbouring vertices are accessible without any search (see Figure 5). As each vertex has l neighbouring vertices, in general $l << n$, the update of the estimation values is realised in $O(l) \approx O(1)$ time.

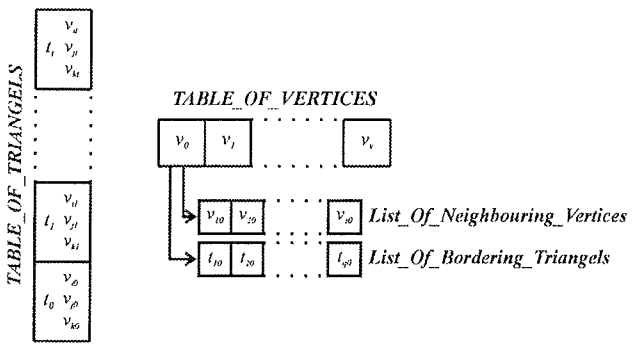

Figure 5: Direct access of the vertex neighbours

It may be desired that the border vertices of the region are eliminated from the decimation process. In this case, they are not inserted into the hash table. Similarly, we can test the shape of the newly created triangles according to the application specific recommendation. For FEM analyses it is desired, for example, that the rate of the triangle sides does not fall below 1:1:10. If that happens, the considered vertex is not removed.

2.3 Triangulation of polygon area

After removing a vertex from the mesh, all the triangles incident to it are eliminated (shaded part in Figure 1). The empty region has to be filled by the new triangles. Franc and Skala applied here a clever solution by selecting the shortest edge from the removed vertex to their neighbours. The shortest edge is contracted pulling all the edges defined by the removed vertex to the opposite vertex of the shortest edge /4/. However, this elegant method works only when the obtained gap forms a convex polygon which is not always the case. Therefore we applied in our solution a classical polygon triangulation algorithm. There are many ways how a polygon can be triangulated: algorithms based on diagonal insertion, restricted Delaunay algorithms, and the algorithms using Steiner points (see /9/ for an overview). In our approach, the well-known ear-cutting algorithm proposed by ElGindy et. al is used /10/.

3 Undecimation

Returning the removed vertices into the mesh in the reverse order of their elimination is an extremely useful feature in practice, giving the user the opportunity to experiment with the mesh. The user may return step by step only a few vertices instead of processing the whole set of vertices again and trying different termination criteria. This feature (denoted as *undecimation*) can be realised easily and very efficiently by the proper data structure as described below.

At the beginning we have an array of vertices and an array of triangles. The position of vertices remains the same, they are just pulled-out from the mesh. The removed vertices are marked by flags. When a vertex is removed, triangles incident to it are removed, too. This is indicated in the triangle array by a flag. There are always less new triangles than original and they occupy the memory locations of the old ones. Some of the locations (at least one) are marked as empty. Figure 6a shows the state of the data structure before and Figure 6b after removing vertex v_0 in the example shown in Figure 1.

The undecimation process requires the knowledge how the process of decimation was executed and what changes in the triangular mesh occured at each step. The easiest solution would be to store a topology of each obtained mesh. This would involve file operation and would slow down the whole process. However, by the proper organisation of data, the shape of the mesh could be easily restored by a tolerable amount of additional memory. Figure 7 explains our solution. Two additional one-way linked lists are introduced at each removed vertex. The first list stores the indices of the removed triangles. It contains only two

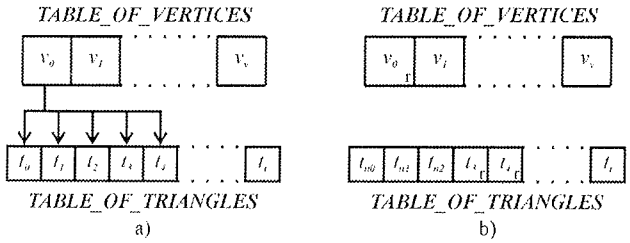

Figure 6: The state of the data structure before (a) and after (b) removing a vertex

indices, because the third one is the removed vertex itself. The second list stores the indices of the new triangles. The process of undecimation is now extremely easy. The vertex, which is going to be returned to the mesh, set-ups the flag indicating that it belongs to the mesh again. Triangles, which have been added by the polygon triangulation process, are removed and the old triangles are restored using the information from the list of the removed triangles.

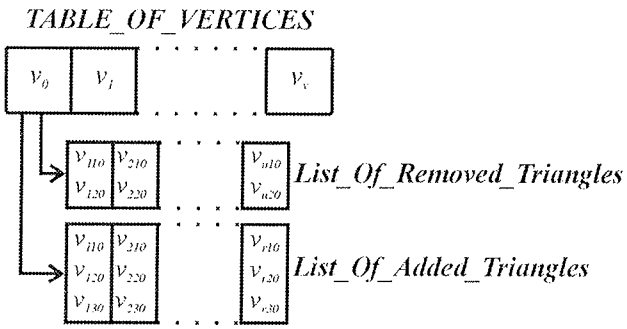

Figure 7: Additional lists by removed vertex

4 Results

4.1 Theoretical time and space complexity

The proposed algorithm for mesh decimation consists of the steps with the following complexity:

- evaluation of vertices is done in $O(n)$, where n is the number of the input vertices,
- removal of a vertex is realised in constant time $O(1)$,
- triangulation of polygon using the ear-cutting is performed in $O(l_i^2)$, where l_i is the number of neighbouring vertices of the removed vertex vi. However, $l_i << n$, and therefore this step can be considered as being done in constant time regarding n.
- re-evaluation of the neighbouring vertices of the removed vertex is done in constant time.

If k is the number of all vertices that are removed during the decimation process and $k < n$, the required time complexity becomes $O(k) + O(n) = O(n)$. The same estimation is obtained for the process of undecimation.

Let us investigate the space complexity of the algorithm. At the beginning, the space for n vertices and $2n$ triangles is allocated (it is well-known that each triangulation consists of at most $2n - h - 2$ triangles, where h is the number of vertices forming the convex hull of the given set of polygons /11/). At each vertex vi being removed, l_i records the removed triangles and $l_i - a$, $0 < a \leq l_i$, $l_i << n$, records about the added triangles are needed. In this way, we obtain linear space complexity $O(n)$.

4.2 Practical results

Tests have been performed on various sets of engineering data and on artificially generated data. As a case study, consider the electric field of a power line borrowed from /12/. In Figure 8a the initial triangular mesh consisting of 11213 vertices and 22010 triangles is shown. The shaded triangular mesh is shown in Figure 8b.

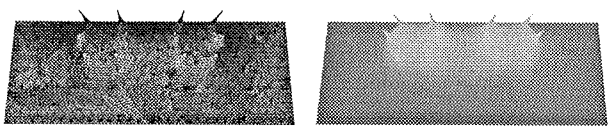

Figure 8: Initial triangular mesh (a) shaded initial triangular mesh (b)

After that, we start the decimation process. At each step, 10% of vertices are removed from the triangular mesh. This process is shown in Figure 9. Notice that despite the smaller number of triangles, the shaded pictures do not differ noticeably until only 20% of the initial vertices remain. In this case, we obtained 2253 vertices and 4090 triangles.

To show how efficient the proposed algorithm is, we arranged the vertices in a regular grid, and triangular meshes are constructed from them. The scalar values in the vertices have been set randomly. 99% of the vertices have at first been removed, and then, all of them are returned (undecimated). The results are shown in Table 1 where CPU time for mesh decimation and undecimation is given. As seen from Figure 10, the proposed algorithm works indeed in linear time. Experiments have been performed on a PC with Celeron 600 MHz processor and 384 MB of RAM.

Table 1: Times needed for mesh decimation and mesh undecimation

INPUT	no. of vertices (x1000)	10	40	90	160	250	360	490	640	810
OUTPUT	no. of vertices	100	400	900	1600	2500	3600	4900	6400	8100
	no. of triangles	188	775	1772	3167	4959	7156	9748	12742	16121
TIME (s)	decimation	0,140	0,651	1,583	2,774	4,357	6,340	8,742	11,526	14,681
	undecimation	0,140	0,611	1,442	2,583	4,086	5,928	8,202	10,816	13,840

5 Conclusion

Huge surface meshes are produced in different fields like volume visualisation in architecture, GIS, industrial design, etc. In electronics, triangular meshes are used for model-

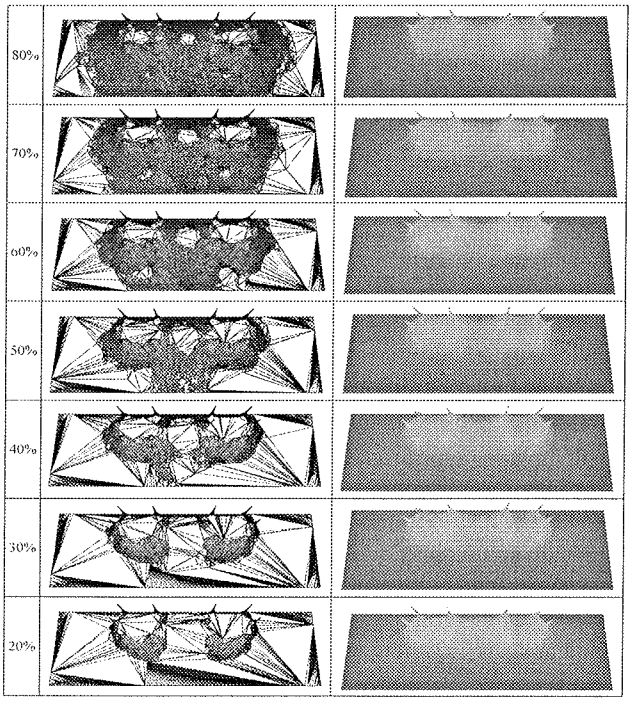

Figure 9: Decimation process

ling temperature distribution and mechanical properties of devices, as well as for visualisation of electrical parameters. Even recent workstations face problems in interactively displaying huge data sets often composed of more than a million of triangles. Reducing the complexity of surface meshes is therefore imperative for engineering applications. This hot topic motivated development of mesh decimation algorithms based on different criteria following specific objectives in practice. The algorithm proposed in this paper combines the approaches of Schroeder /5/ and Franc & Skala /8/. The resulting decimation process is performed in linear time. The usefulness of the algorithm has been demonstrated in a case study of modelling of electric field of a power line. Its performance is characterised by an example of constructing artificial triangular meshes. Low time and space complexity and efficient undecimation feature illustrate the strengths of the algorithm and make it a promising solution for modelling engineering data.

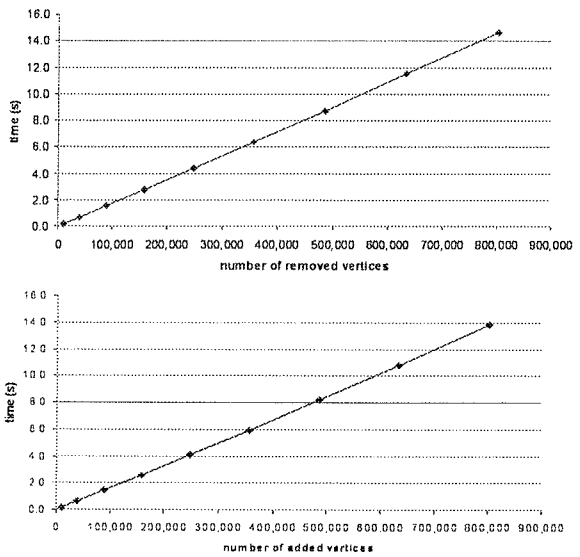

Figure 10: Graph of times needed for mesh decimation (above) and undecimation (below)

Acknowledgement

The authors would like to thank prof. dr. Mladen Trlep from Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia, for providing us with the empirical data used in the paper.

Literature

/1/ M. Lamot, B. Žalik, "Software Tool for the Support of On-line Thermal Monitoring of Microelectronic Systems". Midem, 2000, vol. 30, no. 3, pp. 144-147.

/2/ M. Trlep, A. Hamler, B. Hribernik, "The analysis of complex grounding systems by FEM". IEEE transaction on magnetic, 1998, vol. 34, no. 5, pp. 2521-2524.

/3/ M. Garland, P.S. Heckbert, "Fast Polygonal Approximation of Terrains and Height Fields", technical report, 1995 http://www.cs.cmu.edu/~garland/scape

/4/ M. Franc, V. Skala, "Triangular Mesh Decimation in Parallel Environment", EUROGRAPHICS Workshop on Parallel Graphics and Visualization, Girona, Spain, 2000, pp. 39-52

/5/ W. J. Schroeder, J. A. Zarge, W. E. Lorensen, "Decimation of Triangle Meshes", Computer Graphics, 1992, vol. 26, no. 2, pp. 65-70

/6/ H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, "Mesh Optimization", Proceedings of the 1993 ACM SIGGRAPH conference, 1993, pp. 19-26.

/7/ M. Garland, P. S. Heckbert, "Surface Simplification Using Quadric Error Metrics", SIGGRAPH Conference Proceedings, 1997

/8/ M. Franc, V. Skala, "Parallel Triangular Mesh Decimation Without Sorting", SCCG Proceedings, Budmerice, 2001, pp. 69-75

/9/ M. Lamot, B. Žalik, "A Contribution to Triangulation Algorithms for Simple Polygons". Journal of Computing and Information Technology - CIT, 2000, vol. 8, no. 4, pp. 319-331.

/10/ H. ElGindy, H. Everett, G. Toussaint, "Slicing an Ear in Linear Time", internal memorandum, School of Computer Science, McGill University, 1989

/11/ M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, "Computational Geometry: Algorithms and Applications", Springer 1997.

/12/ M. Trlep, B. Hribernik, "Unified approach to solving a steady-state electromagnetic field. IEEE transaction on magnetic, 1997, vol. 33, no. 2, pp. 1974-1977.

izr. prof. dr. Borut Žalik, univ. dipl. inž.
Sebastian Krivograd, univ. dipl. inž.
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko, Smetanova ulica 17, 2000 Maribor
tel: (02) 220 74 71, fax: (02) 251 11 78
e-mail: zalik, sebastian.krivograd@uni-mb.si

izr. prof. dr. Franc Novak, univ. dipl. inž.
Institut "Jožef Stefan"
Jamova 39, 1000 Ljubljana
tel: (01) 477 33 86, fax: (01) 251 93 85
e-mail:franc.novak@ijs.si

Prispelo (Arrived): 16.04.2002 Sprejeto (Accepted): 28.06.2002