Computer Assisted Mechanics and Engineering Sciences, **17**: 137–145, 2010.

The efficiency of the application of the heap lists to the algorithm of mesh generation

J. Kucwaj
Cracow University of Technology
Institute of Computer Science
Department of Physics, Mathematics and Computer Science
Warszawska 24, 31-155 Cracow, Poland
e-mail: jkucwaj@pk.edu.pl

The paper presents an analysis of the efficiency of the application of heap lists data structures to the 2D triangular mesh generation algorithms. Such efficiency is especially important for the frontal methods for which the size of the generated mesh is controlled by a prescribed function in the considered domain. In the presented approach two advancing front procedures are presented: first for points insertion and the second for the Delaunay triangulation. If the heap lists are applied to the minimal size of frontal segment selection, a better quality mesh is obtained.

Keywords: tree and lists data structure, frontal methods, Delaunay triangulation, grid generation, mesh adaptation.

1. Introduction

The subject of the paper is the application of heap lists [7] algorithms to the computer code of grid generation [3]. The algorithm uses as an input a mesh size function defined in the computational domain. The function uniquely determines the whole mesh. The idea of the frontal method of the mesh generation is based on fixing the initial front of generated line segments on the boundary and then successive insertion of the points over chosen frontal segments with simultaneous front update. It turns out, that it is important how the current frontal segment is chosen [11]. The numerical experience [8, 10] indicates that the choice of the smallest front segment results in meshes with the best quality. For the fast choice of the smallest front element the heap lists data structures algorithms are applied.

A topology and appropriate data structure for a plane domain triangulation are defined by specifying different types of curves which are parts of the boundary [4]. In the present implementation the following types of curves are build in:

- a straight line segment,
- an arc of a circle,
- a B-spline curve.

A curve is represented by its ends and the type. Therefore, all the available curves having a representation in the computer data structure are topologically equivalent to the straight line segment. It is assumed, that the domain $D \subset \mathbb{R}^2$ is multiconnected, that is, k-connected with finite number of internal contours. The boundary of the domain is represented as a union of closed curves, without multiple points. Every loop is represented as a sequence of curves, appearing in the order such that the end of any curve is the beginning of the next one.

2. Heap lists data structures

Heap lists are well-known in computer science binary tree data structures [7, 9]. They allow one for a fast removal or insertion of elements from them in such a way that the smallest element is still at the top of the list. The heap lists, in their structure, are also the binary trees. This duality of their structure causes that appropriate algorithms of insertion and removal are very fast. In addition, by the presence of natural order in which a son is the next to the father and the son from the left is first to the son on the right, they can be treated as lists (see Fig. 1). If we store the segments of the front while performing the advancing front procedure, the first step is to arrange the heap list in such a manner so that the smallest element could be placed at the origin (location 1 in the tree in Fig. 1). It is not necessary to arrange the whole array in an increasing order. The appropriate algorithm is explained in the example shown in Figs. 2, 5. For example, let's take into consideration letters of the word "METHOD". The letters in this word are, in a natural way, ordered in the alphabetical order, the "smallest" is "D". The idea of the algorithm is explained in Figs. 2–5.

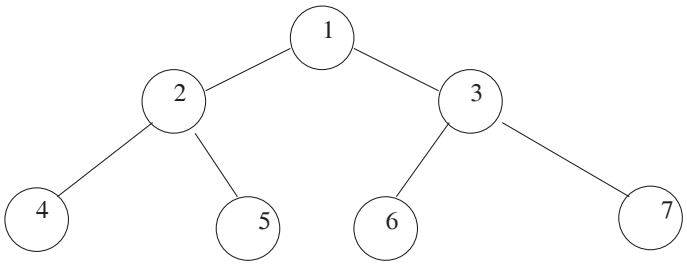

Fig. 1. Binary tree as heap list.

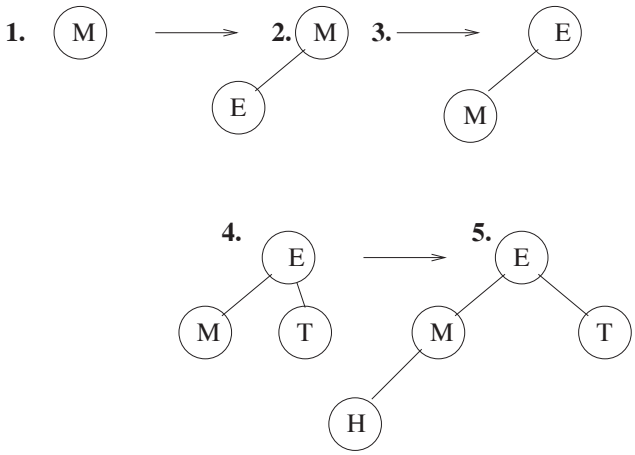

Fig. 2. Binary tree as heap list – insertion.

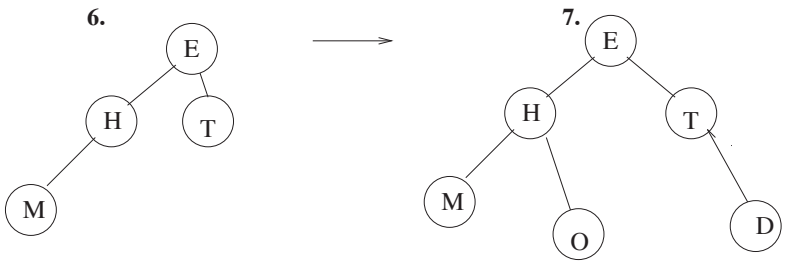

Fig. 3. Binary tree as heap list – insertion continuation.

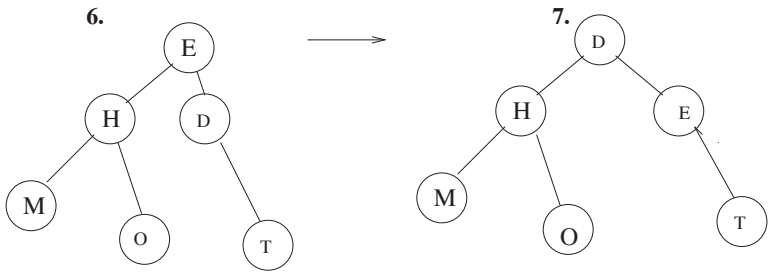

Fig. 4. Binary tree as heap list – deletion.

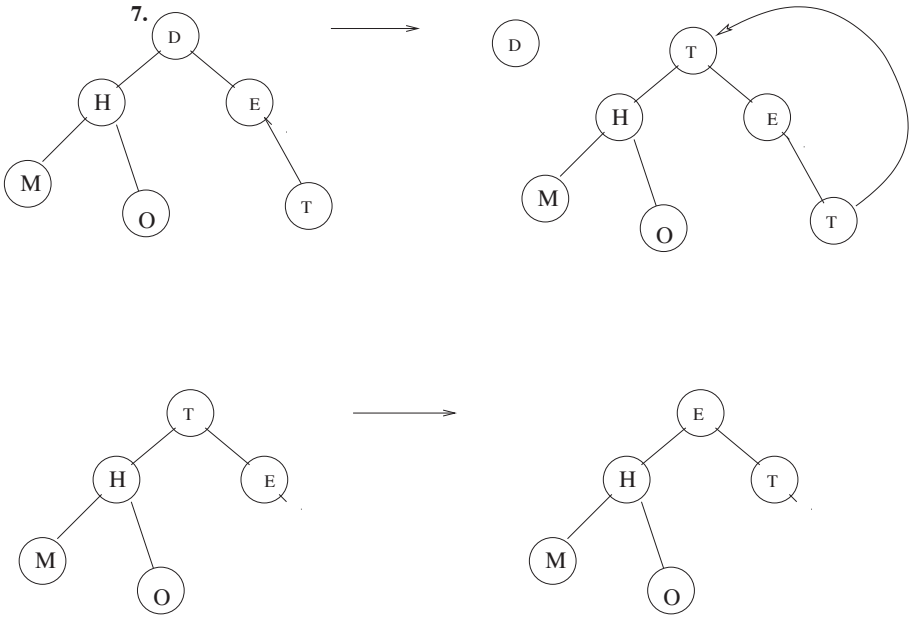

Fig. 5. Binary tree as heap list – deletion continuation.

The main idea of the algorithm is to keep the rule that the father is preceding the son. Such an approach needs only to exchange the elements in the branch. In the next steps of the algorithm of the advancing front technique it is necessary to modify the heap list to insert or to remove an element in such a way that the lowest one is still at the top of the tree. This is done by the algorithms – "INSERTION" and "DELETION" described in [7]. Both algorithms are similar to the approach presented above. In case when the new element is added it is inserted in the last position on the heap list and then, if necessary, interchanged with the father and so on. It would appear that if it is the lowest element it will come to the top of the binary tree (heap list). In case of the removal of the father, the first son takes the place of the father, the second son the place of the first son, unless it is the last element on the list. For the algorithm description, the following integer variables are introduced:

SONPOS – the position of the son on the heap list,

FATHPOS – the position of the father on the heap list,

NHEAP – length of the heap list,

HEAP (1:NHEAAP) – heap list,

SON – the segment number of the son,

FATH – the segment number of the father.

It is assumed that:
SON=HEAP(SONPOS).
FATH=HEAP(FATHPOS)
The algorithm of "DELETION" uses analogous variables. The following formulas for a new locations of sons follow from the binary tree structure [13]:
SONPOS1=2*FATHPOS,
SONPOS2=2*FATHPOS+1.

The "INSERTION" algorithm of adding a new element is as follows [7]:

Algorithm 1.

```
1. NHEAP=NHEAP+1,

2. Place the new segment FNEW at the end of the list
   HEAP(NHEAP)=FNEW,

3. SONPOS=NHEAP,

4. FATHPOS=SONPOS/2 (integer division),

5. SON=HEAP(SONPOS),

6. FATH=HEAP(FATHPOS),

7. IF (the length of the FATH < the length of the SON) THEN

   (a) VAR=HEAP(SON),
   (b) HEAP(SON)=HEAP(FATH),
   (c) HEAP(FATH)=VAR,
   (d) SONPOS=FATHPOS,
   (e) IF (SONPOS>1) THEN go to 4. OTHERWISE finish the algorithm.
       ENDIF

   ENDIF
```

The algorithm "DELETION" of an element is as follows [7]:

Algorithm 2.

```
1. Remove the segment at the top of the list:
   OUT=HEAP(1).

2. Put the segment being at the end of the list to the top:
   LHEAP(1)=LHEAP(NHEAP).

3. Lower NHEAP: NHEAP=NHEAP-1.

4. Set the position of the father FATHPOS in the heap list to:
   FATHPOS=1.

5. Set the positions of the two sons:
   SONPOS1=2*FATHPOS,
   SONPOS2=2*FATHPOS+1.
```

```
6. Denote the segments of the father and the sons associated with these
   positions:
   SON1=HEAP(SONPOS1),
   SON2=HEAP(SONPOS2),
   FATH=HEAP(FATHPOS).

7. Determine which son needs to be interchanged:
   IF (LENGTH(FATH)<LENGTH(SON1),LENGTH(SON2)) THEN VAR=0 ENDIF.
   IF (LENGTH(FATH)>LENGTH(SON1)>LENGTH(SON2)) THEN VAR=SONPOS2 ENDIF.
   IF (LENGTH(FATH)>LENGTH(SON2)>LENGTH(SON1)) THEN VAR=SONPOS1 ENDIF.
   IF (LENGTH(SON1)>LENGTH(FATH)>LENGTH(SON2)) THEN VAR=SONPOS2 ENDIF.
   IF (LENGTH(SON2)>LENGTH(FATH)>LENGTH(SON1)) THEN VAR=SONPOS1 ENDIF.


8. IF (VAR ≠ 0) THEN exchange father and the son (with VAR position) positions:

  (a) Interchange the segments stored in HEAP.
  (b) Set FATHPOS=VAR.
  (c) IF (2*FATHPOS ≤ NHEAP) go to 5.
      ENDIF.

   ENDIF
```

3. Algorithm of grid generation

The main idea of grid generation is based on the algorithm of the advancing front technique and generalization of Delaunay triangulation for [2] arbitrary 2D domains. It is assumed that the domain is multiconnected with arbitrary number of internal loops. The boundary of the domain may be composed of the curves mentioned in Sec. 1.

In the case of the advancing front technique combined with the Delaunay triangulation, the point insertion and triangulation can be divided into the following steps [3]:

1. Points generations on the boundary components of the boundary of the domain.
2. Internal points generation by the advancing front technique.
3. A Delaunay triangulation of the previously obtained set of points.
4. A Laplacian smoothing of the obtained mesh.

The algorithm for boundary points generation depends upon the type of the boundary curve. The appropriate techniques of points generation on straight-line segments, circles and B-spline segments are presented in [5].

4. Mesh quality factors

To find the criterion for assessing the quality of the mesh [1], the shape and size of the triangles is taken into account. For similar triangles the quality measure should be the same. The following [6] mesh quality measures are taken into account for a triangle $\triangle ABC$:

$$\alpha(\triangle ABC) = 2\sqrt{3}\frac{||\mathbf{AB}\times\mathbf{AC}||}{||\mathbf{AB}||^2+||\mathbf{BC}||^2+||\mathbf{CA}||^2}, \tag{1}$$

$$\beta(\triangle ABC) = 2\frac{r}{R}, \tag{2}$$

where R – is the radius of the circle circumscribed over triangle $\triangle ABC$, r – is the radius of the circle inscribed in triangle $\triangle ABC$. The coefficient $2\sqrt{3}$ in the formula (1) is the normalization factor to obtain the quality equal to 1 for the equilateral triangle, the same situation is for β-quality in formula (2). It can be proved that $\alpha(\triangle ABC)$ achieves its maximum value for an equilateral triangle and varies in the interval $(0, 1]$, the same is true for the $\beta(\triangle ABC)$ quality of the triangle.

It is rather difficult to define properly a quality measure for the whole mesh. In this paper we take into account the average of qualities of all triangles and minimal quality of all triangles in the triangulation.

Remark 1. *It can be mentioned that mesh qualities of the right-angled isosceles triangle are equal to:*

$$\begin{aligned} \alpha &= \frac{\sqrt{3}}{2} = 0.866025, \\ \beta &= \frac{2}{1+\sqrt{2}} = 0.828427. \end{aligned} \tag{3}$$

On the other hand, for the right-angled triangle with acute angles 30° *and* 60° *these qualities are equal to:*

$$\begin{aligned} \alpha &= \frac{6}{8} = 0.75, \\ \beta &= \frac{2}{1+\sqrt{3}} = 0.73050. \end{aligned} \tag{4}$$

5. Numerical examples

In all of the presented examples the computational domain is taken from fluid mechanics applications [13]. It is a square $[-1, 2] \times [-1, 2]$ with the hole modeled by aircraft profile NACA0012 [5].

In the first two examples the mesh generator creates two meshes with the same mesh size function given by formula (5), first with the choice of minimal frontal element, and the second with random choice of frontal element. Figures 6 and 7 show the meshes without and with minimal frontal segment choice, respectively. The mesh density function was taken as follows:

$$\rho(x,y) = \begin{cases} 0.2\sqrt{(x+0.05)^2+y^2}+0.02, & \text{if } (x<0.5 \text{ and } y<0.15), \\ 0.2\sqrt{(x+.05)^2+y^2}+0.05, & \text{if } (x<0.5 \text{ and } y>0.15), \\ 0.2\sqrt{(x-1.05)^2+y^2}+0.02, & \text{if } (x>0.5 \text{ and } y<0.15), \\ 0.2\sqrt{(x-1.05)^2+(y-0.3)^2}+0.1, & \text{otherwise.} \end{cases} \tag{5}$$

In the next two examples two meshes are generated with the mesh size function given by formula (6). Figure 8 shows the mesh obtained without the heap list application algorithm, Fig. 9 presents the mesh obtained with an application of the heap data structure algorithms. Table 1 presents mesh quality parameters for meshes from Figs. 6–9. The meaning of variables in Table 1 is as follows: **TRI** – number of triangles, **NOD** – number of nodes. The mesh density function is as follows:

$$\rho(x,y) = \begin{cases} 0.2\sqrt{(x+0.05)^2+y^2}+0.01, & \text{if } (x<0.5 \text{ and } y<0.15), \\ 0.2\sqrt{(x+.05)^2+y^2}+0.01, & \text{if } (x<0.5 \text{ and } y>0.15), \\ 0.2\sqrt{(x-1.05)^2+y^2}+0.010, & \text{if } (x>0.5 \text{ and } y<0.15), \\ 0.2\sqrt{(x-1.05)^2+(y-0.3)^2}+0.01, & \text{otherwise.} \end{cases} \tag{6}$$

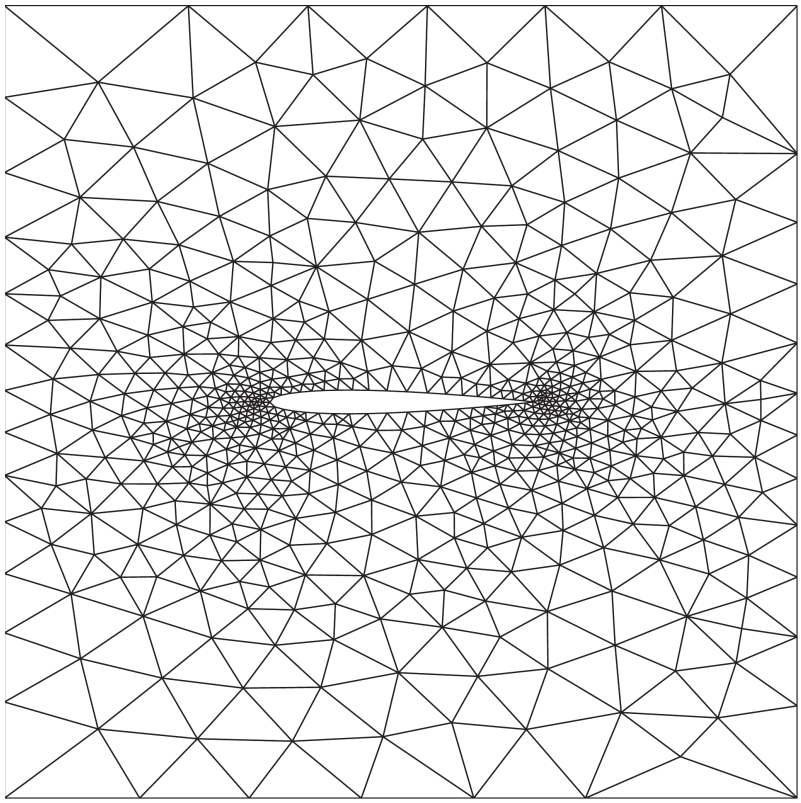

Fig. 6. A mesh generated without heap list data structure application with mesh size function given by formula (5).

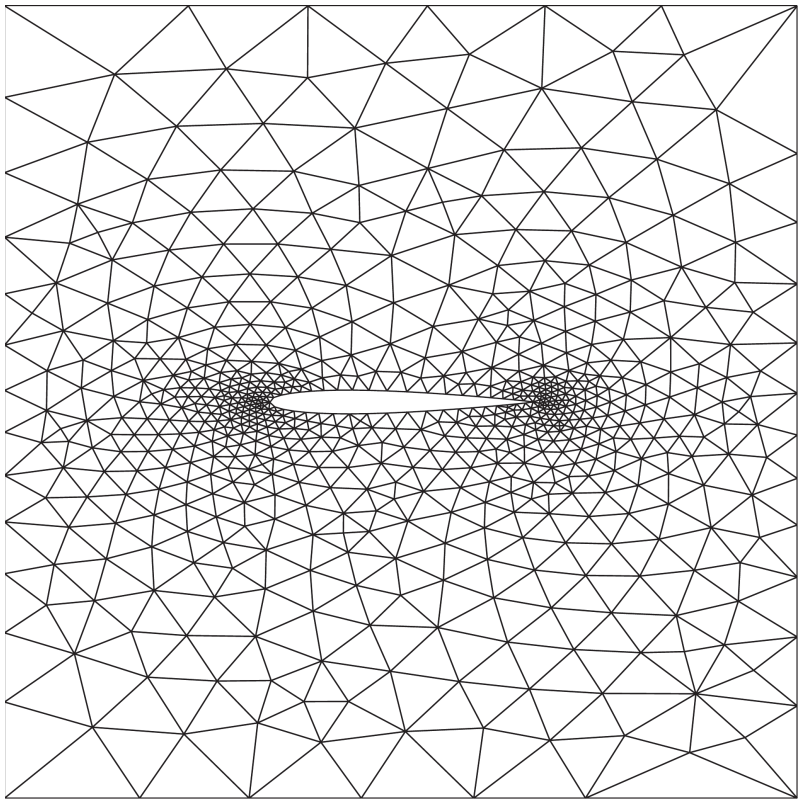

Fig. 7. A mesh generated with heap list data structure application with mesh size function given by formula (5).

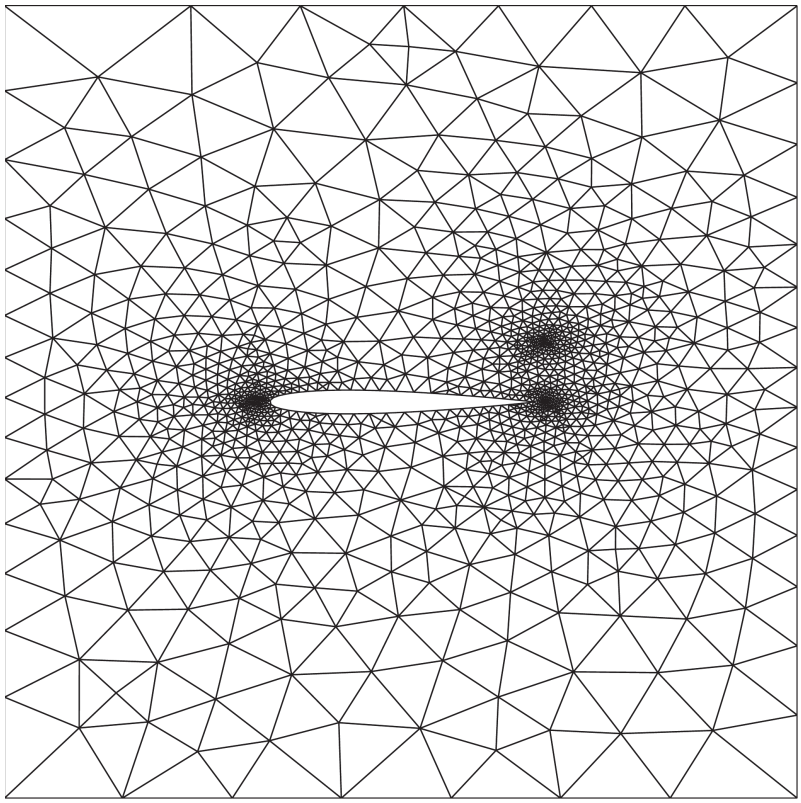

Fig. 8. A mesh generated without heap list data structure application with mesh size function given by formula (6).

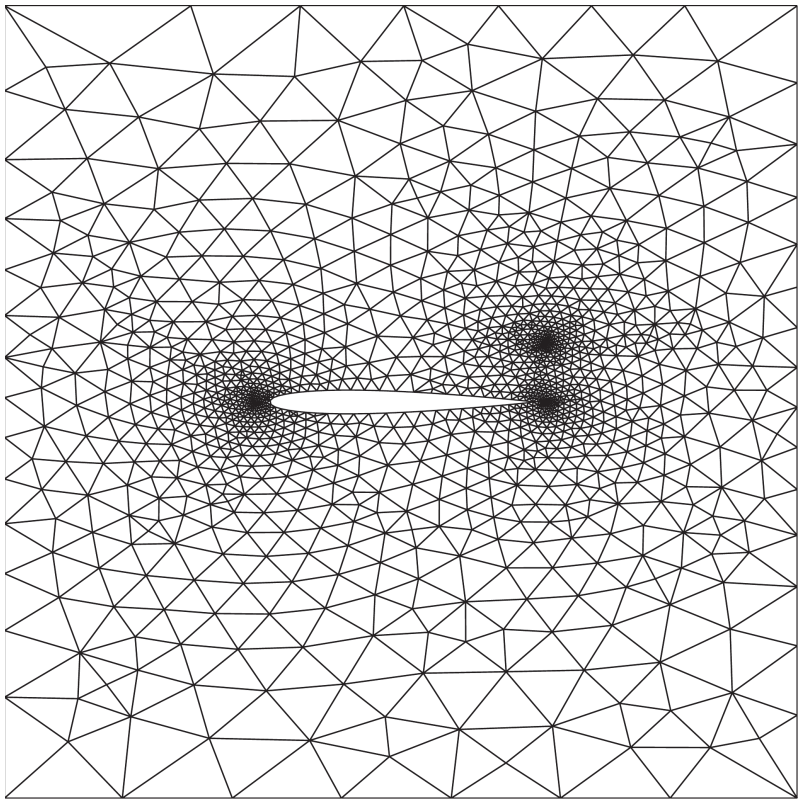

Fig. 9. A mesh generated with heap list data structure application with mesh size function given by formula (6).

Table 1. Parameters of meshes.

	TRI	NOD	$\alpha_{\min}$	α_{aver}	$\beta_{\min}$	β_{aver}
mesh from Fig. 6	1197	637	0.484811	0.948219	0.404435	0.941107
mesh from Fig. 7	1283	680	0.503969	0.959738	0.459709	0.953240
mesh from Fig. 8	2310	1202	0.494314	0.958332	0.465855	0.952391
mesh from Fig. 9	2520	1307	0.492145	0.960055	0.474445	0.954270

6. Conclusions

The results presented in Table 1 suggest that quality parameters for meshes generated with the application of the heap lists data structure are better than for the meshes generated with random choice of frontal segment. In both cases the requirements of the mesh size function are more or less satisfied and, visually, the meshes are similar. In comparison with other implementations [12] the influence of these data structure applications is less meaningful, but it is always important to have the best triangles qualities thus the heap lists application is always recommended. Moreover, in the case of the binary tree algorithms application the meshes have about 8% less nodes, provided that the same mesh size function is used, what suggests satisfaction for better mesh size function requirements. Mesh qualities with heap lists application are usually 7–10% better. It seems that this follows from the way of the insertion of a new point over the current front segment (taking into account the mesh size function). Application of the heap lists is very inexpensive. For example for $2^{14} = 16384$ elements, we have at most 14 (length of the branch of the tree) comparisons to complete insertion or deletion algorithms, whereas the length of the heap list is 16385.

References

[1] S. Chalasani, D. Thompson. Quality improvements in extruded meshes using topologically adaptive generalized elements. *International Journal for Numerical Methods in Engineering*, **60**: 6, 1139–1159, 2004.

[2] J. Kucwaj. Delaunay Triangulation of Surfaces. *ZAMM*, **76**: S.3, 249–250, 1996.

[3] J. Kucwaj. The Algorithm of Adaptation by Using Graded Meshes Generator. *Computer Assisted Mechanics and Engineering Sciences*, **7**: 615–624, 2000.

[4] J. Kucwaj. The Application od the Adaptive Algorithm to the Potential Problems. *Annales UMCS Informatica*, AI, **2**: 37–45, 2004.

[5] J. Kucwaj, Numerical investigations of the convergence of a remeshing algorithm on an example of subsonic flow. *Computer Assisted Mechanics and Engineering Sciences*, **2–4**: 147–160, 2010.

[6] S.H. Lo. Delaunay triangulation of non-convex planar domains. *Int. J. Num. Meth. Engng.*, **28**: 2695–2707, 1989.

[7] R. Loehner. Some useful data structures for the generation of unstructured grids. *Comm. Appl. Numer. Methods*, **4**: 123–135, 1988.

[8] R. Loehner. A parallel advancing front grid generation scheme. *Int. J. Num. Meth. Engng.*, **51**: 663–678, 2001.

[9] S.J. Owen, S. Saigal. Surface sizing control. *Int. J. Num. Meth. Engng.*, **47**: 497–511, 2000.

[10] S.J. Owen, S. Saigal. Formation of pyramid elements for hexahedra to tetrahedra transition. *Comp. Meth. Appl. Mech. Engng.*, **190**: 4505–4518, 2001.

[11] S. Pippa, G. Caligiana. GradH-Correction: guaranteed sizing gradation in multi-patch parametric surface meshing. *Int. J. Num. Meth. Engng.*, **62**: 495–515, 2005.

[12] A. Szotko, R. Loehner. Three-dimensional parallel unstructured grid generation. *Int. J. Num. Meth. Engng.*, **38**: 905–925, 1995.

[13] J.F. Thompson, B.K. Soni, N.P. Weatherwill. Handbook of Grid Generation. CRC Press, Boca Raton, London, New York, Washington, D.C., 1999.