pp. 1–11 (2018)

# Fast tetrahedral mesh generation and segmentation of an atlas-based heart model using a periodic uniform grid

E. Vasilev*, D. Lachinov*, A. Grishin*, and V. Turlapov*

**Abstract** — A fast procedure for generation of regular tetrahedral finite element mesh for objects with complex shape cavities is proposed. The procedure like LBIE-Mesher can generate tetrahedral meshes for the volume interior to a polygonal surface, or for an interval volume between two surfaces having a complex shape and defined in STL-format. This procedure consists of several stages: generation of a regular tetrahedral mesh that fills the volume of the required object; generation of clipping for the uniform grid parts by a boundary surface; shifting vertices of the boundary layer to align onto the surface. We present a sequential and parallel implementation of the algorithm and compare their performance with existing generators of tetrahedral grids such as TetGen, NETGEN, and CGAL. The current version of the algorithm using the mobile GPU is about 5 times faster than NETGEN. The source code of the developed software is available on GitHub.

**Keywords:** FE-mesh generation, tetrahedral mesh, regular mesh, segmentation, human heart modelling, fast generation

**MSC 2010:** 65M50

This research addresses the problem of fast generation of regular three-dimensional tetrahedral mesh with complex shape cavities. Current existing open-source software packages do not have such capabilities. For digital medicine, an important example of objects with cavities having a complex shape is the human heart.

For a qualitative simulation of the electrical activity of the human heart and obtaining reliable results, a reasonably detailed model of the heart should be used, which will accurately approximate the real heart.

Nowadays, there is a number of finite element models (FE models) of the heart for modelling electrical signals. For example, in some systems of heart electrical modelling, the rabbit heart model is used [1, 3]. The rabbit heart model was constructed from a high resolution MRI dataset, with the use of an intensity based level-set filter.

Early 3D models of cardiac anatomy were the simplistic models based on the geometric shapes. This approach is still in use for applications where the anatomical validity is not important for the purpose of the model [5, 12, 15]. Currently, researches use the simple geometric shapes and make parameterized models based on the segmented heart images [8].

*Lobachevsky State University of Nizhny Novgorod, Nizhny Novgorod 603950. E-mail: eugene.unn@gmail.com

The research was supported by the Ministry of Education and Science of the Russian Federation (Contract No. 02.G25.31.0157).

In many articles, studies are based on the models built on the segmentation of CT [2, 4] or MRI [9, 10] scans. The models of this type are close to real heart geometry, but describe only one heartbeat phase, rather than the whole cycle.

At present, the most common and popular open source mesh generators are: TetGen, Ani3D, MFEM, NETGEN, GMSH, LBIE.

TetGen is a very popular software for generation of the three-dimensional tetrahedral meshes and three-dimensional Delaunay triangulations [13]. It is used in Wolfram Mathematica and GMSH software and can be included as a plugin in OpenCascade and other CAD systems.

Ani3D is a software package for generation of the unstructured tetrahedral meshes and adapting them isotropically and anisotropically [16]. Ani3D can be used to generate quasi-optimal meshes.

NETGEN is an automatic 3D tetrahedral mesh generator based on the abstract rules [11]. The main idea is to transform the front triangle to a local coordinate system and apply the most appropriate abstract rule. NETGEN developers offer generation rules for tetrahedron generation for 27 different situations.

MFEM is an open-source library for finite element methods, including a parallel generator of meshes [17], and now it is gaining popularity.

Open source LBIE-Mesher [19, 20] deserves special mention, because its goal is very close to our one. The goal of LBIE (Level Set Boundary Interior and Exterior) Mesher is to generate tetrahedral/hexahedral Finite Element (FE) meshes for the volume interior to an isosurface, the volume exterior to an isosurface, or an interval volume between two isosurfaces, and to do this directly from the volumetric imaging data (CT or MRI tomography). Working capacity of LBIE in [19] is demonstrated on the example of the human heart, which is especially interesting for us. The development of LBIE was supported by the authors in 2003–2006. The main drawback of the LBIE, in our opinion, was its rigid binding to the reconstruction of the isosurface. In practice, the task of reconstructing the isosurface according to the tomography of human organs is the source of many artifacts. Today, in world practice, the organ surfaces (instead of isosurfaces), are reconstructed enough successfully, and this is usually done with the help of machine learning methods.

## 1. Source data

As the initial data, anatomical segments of heart surface from the Plastic boy anatomy (Plastic boy store) were used. The model was segmented and supplemented by the heart conduction system (Sinoatrial node, Atrioventricular node, His bundle branches, Bachmann's bundle). We have several types of meshes (see Fig. 1): the outer surface of the heart; atria and ventricles; fibrous tissue; heart conduction system. All surfaces must be polygonal and defined in STL-format.

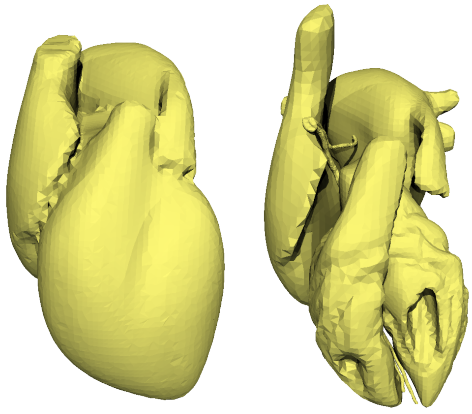


**Figure 1.** The outer surface of the heart and internal heart surfaces.

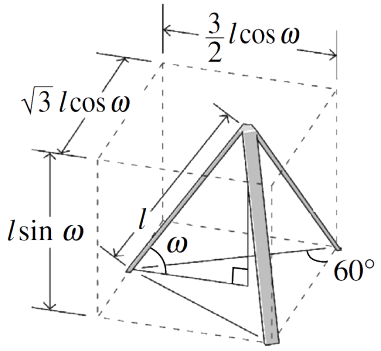



**Figure 2.** Regular tetrahedron formula [18].

## 2. Regular mesh generation

The algorithm consists of several parts: regular mesh generation; classification of atria and ventricles; removal of cavities and protruding parts.

### 2.1. Initial mesh generation

The main idea is to calculate all vertices independently of each other using a regular grid and write a highly parallel code. This is of particular importance for the calculations performed on a finite element mesh of the heart. Nowadays, such calculations on a supercomputer can take hours. For the performance of finite element calculations, it is also important that the regular grid has a grain, formed by a number of tetrahedra, which allows to organize a periodic and hierarchical structures. When adapting such a grid to the surface of a given shape, it is more interesting to preserve the grid structure, if possible, by changing only the positions of the vertices.

To generate regular mesh, we use aspect ratios of a tetrahedron (see Fig. 2). The optimal angle $\omega$ is about 54.7356°, the edge length $l$ can be specified arbitrarily by the user. The position of each vertex in parallelepiped can be calculated using the

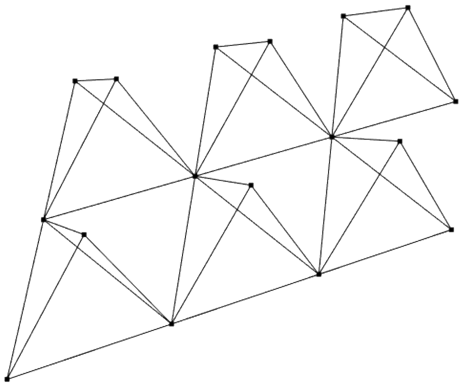


**Figure 3.** One layer of tetrahedra.

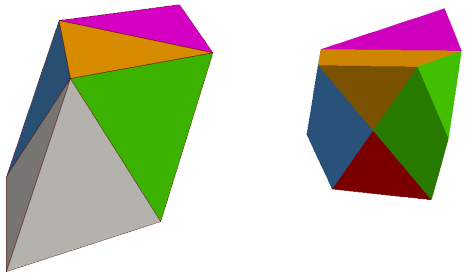


**Figure 4.** The kernel of the regular tetrahedral mesh structure proposed here (left) and its cross-section (right).

formulas:

$$\begin{aligned} x &= i_x \cdot \sqrt{3}\, l \cos(w) + i_y \cdot \frac{\sqrt{3}}{2}\, l \cos(w) + i_z \cdot \frac{\sqrt{3}}{2}\, l \cos(w) \\ y &= i_y \cdot \frac{3}{2}\, l \cos(w) + i_z \cdot \frac{1}{2}\, l \cos(w) \\ z &= i_z \cdot l \sin(w) \end{aligned} \tag{2.1}$$

where $x$, $y$, and $z$ are the vertex coordinates and $i_x$, $i_y$, and $i_z$ are the indices in a 3-dimensional array.

The volume cannot be filled with identical tetrahedra because tetrahedron is not a self-similarity figure, so there is some free space left between the tetrahedra (see Fig. 3). The free space forms some octahedra, and we divide each of them into 4 tetrahedra.

To assess the mesh quality, we use aspect ratio ($L_{\max}/(2 \cdot \sqrt{6} \cdot r_{\text{in}})$), where $L_{\max}$ is the length of the longest edge of a tetrahedron and $r_{\text{in}}$ is the radius of the sphere inscribed in a tetrahedron) [6,14]. Using the angle $\omega = 54.7356^\circ$ we have the aspect ratio 1.0 for the main tetrahedron and 1.5236 for the tetrahedra formed from the octahedron.

As a result, after creating the vertices and the tetrahedra, we have a regular mesh in the form of an inclined parallelepiped (see Fig. 4).

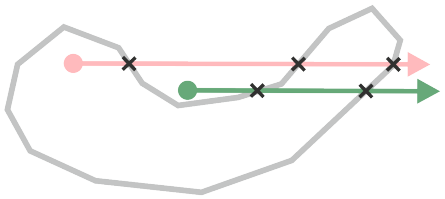


**Figure 5.** Even-rule algorithm (2D schematic). If the point is on the inside of the polygon then it will intersect its edge an odd number of times, and if it is on the outside, an even number of times.

By changing the formulas, we can make a number of figure shapes bounding the mesh (such as a cube, a cylinder, etc.). In the general case, the bounding shape is not so important, because at the next stage we have to remove the vertices not belonging to the heart.

An important advantage of mesh generation by this method is that there is no need to store the full mesh during generation on the GPU. We can generate mesh layers independently, and move mesh layers from the GPU after generation to free the GPU memory for a new mesh layer. This approach allows one to built the meshes of large dimensions.

## 2.2. Mesh cutting

The next step is to remove vertices that extend beyond the bounding surface. To remove them, we need to mark vertices for deleting. We mark the vertices located inside tissue meshes (see Fig. 1) using the ray casting algorithm. A two-dimensional version of the algorithm is shown in Fig. 5.

We start the ray from the vertex being tested in any fixed direction and calculate the number of intersections with heart tissue shells. If the point is outside the shell, the ray will intersect it an even number of times. If the point is inside the shape, then the ray will intersect the edge an odd number of times. Having tested vertices with all shapes, we can delete tetrahedra containing the marked vertices.

After marking, we know which vertices we want to exclude from our mesh. We also need to exclude the tetrahedra incident to these vertices.

The main idea of the cutting algorithm is generating vector match pairs ‘old vertex index’—‘new vertex index’. We create a new vertex array by removing the vertices from the old one and moving the ‘surviving’ vertices to the places of deleted vertices. For each tetrahedron we check all its vertices: if at least one vertex is not contained in the ‘new vertex index’, then we delete it, otherwise we change old indices to new indices.

A more complex option is to remove the marked vertices adjacent to the unmarked vertices. In this case, an incidence matrix is constructed, and before deleting we check whether the unmarked neighbour has a vertex.

After deleting the vertices we obtain the model shown in Fig. 6.

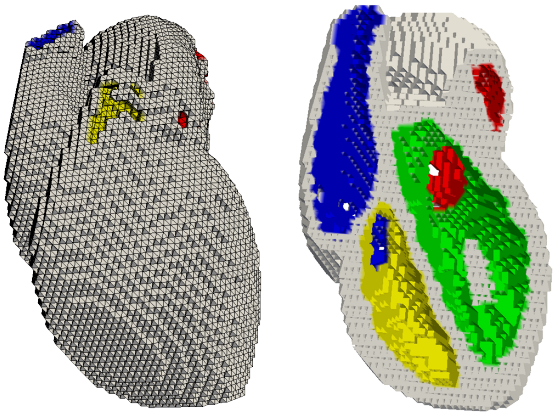


**Figure 6.** Figure after fast cutting operation.

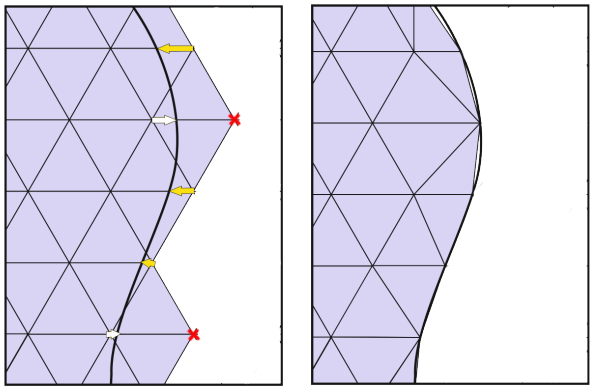


**Figure 7.** Smoothing algorithm (2D schematic).

## 2.3. Mesh surface smoothing

After the cutting stage, we get an unsmoothed mesh (see Fig. 6), the boundaries of which do not coincide with the boundaries of the surface, which we meshed with.

The smoothing algorithm is labelled as Algorithm 2.1. We calculate the distance from the vertex to the surface. If this distance is more than 0.4 of edge length, then we remove this vertex and incident tetrahedra, and move the adjacent vertices to the boundary. If this distance is less than 0.4 of edge length, then we move this vertex to the boundary (see Fig. 7). Applying algorithm to one layer shown in Fig. 8: top pictures show compaction of tetrahedra layer, bottom pictures show stretch of one tetrahedron layer.

As a result of all operations we have a regular mesh, and the regularity is only violated at the boundary (see Fig. 9).

In some cases the final check-aspect-ratio step may accidentally delete some internal tetrahedra leaving holes inside the domain, especially in parts with high curvature boundary and respectively big edge lengths. While the vertices are projected to the surface, they change the aspect ratios of both boundary and internal

**Algorithm 2.1** Mesh surface smoothing

**for** each outside vertex **in** mesh **do**
  calculate distance between vertex and boundary
  **if** distance $>$ 0.4 $\cdot$ edge length **then**
    delete this vertex
    add neighbours to special vertices list

  **else**
    calculate new vertex position as projection of vertex onto a shape and replace old vertex

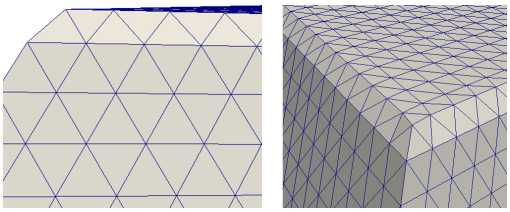
  **end if**

**end for**
**for** each vertex **in** special vertex list **do**
  move this vertex to the boundary

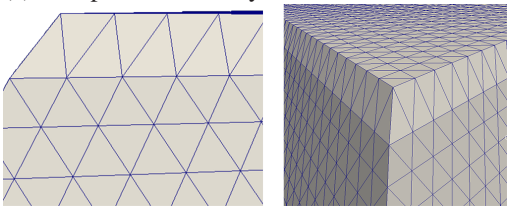
**end for**
**for** each tetrahedron incident to special vertices **do**
  check aspect ratio
  **if** aspect ratio $>$ 3.0 **then**
    delete tetrahedron

  **end if**

**end for**



(a) Compacted mesh layer



(b) Stretched mesh layer

**Figure 8.** Shifting operations with the boundary layer of tetrahedra.

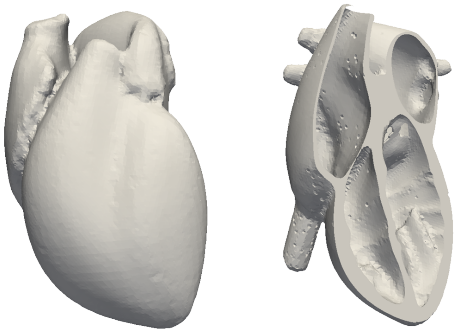


**Figure 9.** Resulting mesh.

**Table 1.** Measurement results.

| Parameter | NETGEN | TetGen | CGAL | Our approach |
|---|---|---|---|---|
| Number of points | 343862 | 892420 | 2326140 | 386797 |
| Number of tetrahedra | 1920978 | 4608171 | 7152246 | 2220448 |
| Average aspect ratio | 1.38692 | 3.37772 | 2.92473 | 1.35397 |
| Maximum aspect ratio | 4.91937 | 148.163 | 9858.46 | 7.53282 |
| Average edge length | 0.062730 | 0.082872 | 0.039632 | 0.114907 |
| Maximum edge length | 0.160629 | 1.24866 | 2.27701 | 0.1589 |
| time | 290 sec. | 195 sec. | 909 sec. | 59 sec. |

tetrahedra. Also, the algorithm can make a low-quality mesh for walls that are smaller than the size of the tetrahedron.

## 3. Performance measurement

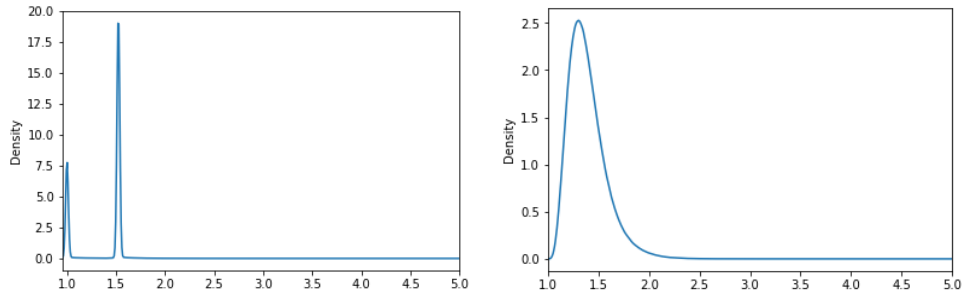
To measure the performance, we created tetrahedral meshes using four different mesh generators with the same heart surface. It consists of 15350 triangular faces. We compared 4 mesh generators: TetGen, NETGEN, CGAL and our approach.

To measure the performance, a computer with the following characteristics was used: CPU Intel Core i7 4710HQ@2.50GHz and NVIDIA GeForce GTX 850M GPU with 2 GB of DDR3 memory, 128 bit interface, 902 MHz, 640 CUDA cores, operating system Windows 8.1 x64.

Measurement results are given in Table 1.

The current version of the algorithm using the mobile GPU is about 5 times faster than NETGEN. The total time of the mesh generation consists of the following parts:

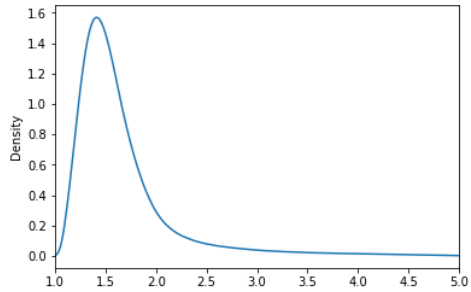
- Mesh generation: 0.752 seconds;
- Mesh marking: 7.416 seconds;
- Mesh cutting: 6.668 seconds;
- Mesh surface refinement: 44.063 seconds;



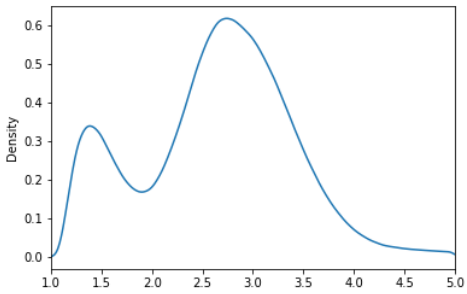

(a) Our approach aspect ratio distribution




(b) NETGEN aspect ratio distribution




(c) TetGen aspect ratio distribution




(d) CGAL aspect ratio distribution

**Figure 10.** Comparison of tetrahedra quality for different mesh generators.

**Table 2.** Algorithm GPU part measurements.

| Parameter | GTX 850M | GTX 680 |
|---|---|---|
| Architecture | Maxwell GM107 | Kepler GK104 |
| Number of cores | 640 | 1536 |
| Memory bandwidth | 32 GB/s | 192.3 GB/s |
| Mesh generation time | 0.721 sec. | 0.594 sec. |
| Mesh marking time | 6.839 sec. | 1.559 sec. |

It can be seen that three-quarters of the time is occupied by the smoothing operation of the surface. Currently, it is sequential algorithm, but after creating a parallel version for the GPUs, its execution time will be reduced by 80%. The mesh cutting operation is also sequential at present, but by parallelizing all operations we expect an acceleration of up to 15x with low-end GPUs, and up to 60x with hi-end GPUs to be achieved.

The comparison of tetrahedra quality for different mesh generators is presented in Fig. 10.

Performance comparison of the GPU part of the algorithm on two different video cards is shown in Table 2. We see the superior performance of a more powerful GPU. Tasks are performed faster due to a higher frequency and a greater number of GPU cores. In the computationally complex part of the algorithm, the performance gap between GPUs is even more evident.

It can be concluded from the analysis of this comparison results that our approach is well suited for generating tetrahedral meshes with a small step size.

## 4. Conclusion

We described and implemented a finite-element mesh constructing algorithm based on the trimming procedure of periodical regular tetrahedral mesh. The main advantage of the algorithm is that the tetrahedra in the periodical regular tetrahedral mesh are generated independently of each other at the generation stage, therefore this procedure is much better parallelized compared to wave propagation algorithms. At the cutting and smoothing stages of the algorithm, the tetrahedra are also processed independently of each other, so these stages do not slow down the algorithm.

The vertices are simply projected to the boundary surface, therefore the sharp edges of the initial domain will not be always preserved. This is not a big issue for cardiac simulations, since cardiac domains do not have sharp edges.

The proposed algorithm can be best applied to create meshes with cavities and dense meshes consisting of small-size tetrahedra. The algorithm offers excellent parallelization opportunities for architectures with a large number of computing processors like GPU.

Source codes of the developed software are available on GitHub: https://github.com/FenixFly/Parallel-Mesh-Generator.

## References

1. H. J. Arevalo, P. M. Boyle and N. A. Trayanova, Computational rabbit models to investigate the initiation, perpetuation, and termination of ventricular arrhythmia. *Progress Biophys. Molecul. Biology* **112** (2016), No. 2, 185–194.
2. O. V. Aslanidi, T. Nikolaidou, J. Zhao, B. H. Smaill, S. H. Gilbert, A. V. Holden, T. Lowe, P. J. Withers, R. S. Stephenson, J. C. Jarvis, J. C. Hancox, M. R. Boyett, and H. Zhang, Application of micro-computed tomography with iodine staining to cardiac imaging, segmentation, and computational model development. *IEEE Trans. Medical Imaging* **32** (2013), No. 1, 8–17.
3. R. Bordas, K. Gillow, Q. Lou, I. R. Efimov, D. Gavaghan, P. Kohl, V. Grau, and B. Rodriguez, Rabbit-specific ventricular model of cardiac electrophysiological function including specialized conduction system. *Progress Biophys. Molecul. Biology* **107** (2011), No. 1, 90–100.
4. D. Deng, P. Jiao, X. Ye, and L. Xia, An image-based model of the whole human heart with detailed anatomical structure and fiber orientation. *Comp. Math. Methods Medicine* (2012). Article ID 891070, 16p.
5. P. C. Franzone, L. Guerri, M. Pennacchio, and B. Taccardi, Spread of excitation in 3D models of the anisotropic cardiac tissue. II. Effects of fiber architecture and ventricular geometry. *Math. Biosciences* **147** (1998), No. 2, 131–71.
6. P. J. Frey and P.-L. George, *Mesh Generation: Applications to Finite Elements.* Hermes Science Publishing, Oxford–Paris, 2000.
7. M. Kremer, D. Bommes, and L. Kobbelt, OpenVolumeMesh—a versatile index-based data structure for 3D polytopal complexes. *Proc. 21st Int. Meshing Roundtable* (2013), 531–548.
8. P. Lamata, M. Sinclair, E. Kerfoot, A. Lee, A. Crozier, B. Blazevic, S. Land, A. J. Lewandowski,

D. Barber, S. Niederer, and N. Smith, An automatic service for the personalization of ventricular cardiac meshes. *J. Royal Society Interface* **91** (2013), No. 11, 62–72.

9. A. Lopez-Perez, R. Sebastian, and J. M. Ferrero, Three-dimensional cardiac computational modelling: methods, features and applications. *Biomed. Engrg. Online* **14** (2015), Art. 35.
10. M. Plotkowiak, B. Rodriguez, G. Plank, J. E. Schneider, D. Gavaghan, P. Kohl, and V. Grau, High performance computer simulations of cardiac electrical function based on high resolution MRI datasets. *Computational Science ICCS 2008*. pp. 571–580.
11. J. Schöberl, NETGEN—an advancing front 2D/3D-mesh generator based on abstract rules. *Comput. Visualiz. Science* 1997, 41–52.
12. M. Sermesant, P. Moireau, O. Camara, J. Sainte-Marie, R. Andriantsimiavona, R. Cimrman, D. L. Hill, D. Chapelle, and R. Razavi, Cardiac function estimation from MRI using a heart model and data assimilation: advances and difficulties. *Medical Image Analysis* **10** (2006), No. 4, 642–656.
13. H. Si, TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Software* **41** (2015), No. 2, Article 11, 36 p.
14. C. Simpson, C. D. Ernst, P. Knupp, P. P. Pébay, and D. C. Thompson, *The Verdict Library Reference Manual.* Sandia National Laboratories, April 2007.
15. F. A. Syomin and A. K. Tsaturyan, Mechanical model of the left ventricle of the heart approximated by axisymmetric geometry. *Russ. J. Numer. Anal. Math. Modelling* **32** (2017), No. 5, 275–291.
16. Yu. V. Vassilevski and K. N. Lipnikov, An adaptive algorithm for quasioptimal mesh generation. *Comput. Math. Math. Phys.* **39** (1999), No. 9, 1468–1486.
17. P. S. Vassilevski, Sparse matrix element topology with application to AMG(e) and preconditioning. *Numer. Linear Algebra Appl.* (2002), No. 9, 429–444.
18. H. N. G. Wadley, Multifunctional periodic cellular metals. *Phil. Trans. R. Soc. A* **364** (2005), 31–68.
19. Y. Zhang, Tetrahedral/hexahedral finite element meshing from volumetric imaging data: proposal for a dissertation. *Inst. Comput. Engrg Sci. The Univ. Texas at Austin (USA)*, 2003.
20. Y. Zhang and C. Bajaj, Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. *Computer Methods Appl. Mech. Engrg. (CMAME)* **195** (2006), No. 9-12, 942–960.