Bull. Nov. Comp. Center, Comp. Science, 48 (2024), 1–12

Software system for cellular automata modeling of gas flows

E.K. Burnyshev

Abstract. The paper presents a new software system for cellular automata modeling of gas flows. The developed software modules of this system and the algorithms used in them are described. Testing of the developed modules and system was carried out on several basic gas dynamics problems. The comparison with the known results is made.

Introduction

Modern science provides a wide range of approaches for modeling natural processes. One of the most promising methods for modeling spatial-dynamic processes is cellular automata (CA) [1]. This approach is particularly useful for describing systems in which the process under investigation is modeled by nonlinear or discontinuous functions. The cellular automata approach has been extensively applied in the modeling of gas dynamic phenomena. The main advantages of cellular automata include a high degree of agreement between simulation results and analytical solutions for gas dynamics problems, as well as their natural parallelism. Due to these factors, scientists are increasingly resorting to the use of cellular automata models in their research.

Among the promising methods for two-dimensional modeling of gas flows is the FHP-MP model [2]. These cellular automata are a generalization of the classical boolean FHP model [3]. However, in the implementation of the FHP-MP cellular automata, there is a need to support the following functionalities: defining and adjusting the geometry of the modeling domain; setting boundary conditions for the problem; utilizing tools to transfer computational models from other software packages.

This paper is dedicated to the description of a software system designed for modeling gas flows, alongside its main modules and the algorithms employed within their operation. Specifically, the paper describes: an algorithm for converting a raster image into the hexagonal grid of the FHP-MP model and an algorithm for converting the geometry of a finite element model into the FHP-MP cellular automata. The general appearance of the software system is presented, as well as the architecture of the developed software modules. Results of system testing on several basic gas dynam-

ics problems are provided. A qualitative assessment of the results obtained during the computational experiment is given.

1. Software system architecture

1.1. Overview of the software system. The structure of the software system for modeling gas flows is described below. The system's architecture is illustrated in Figure 1.

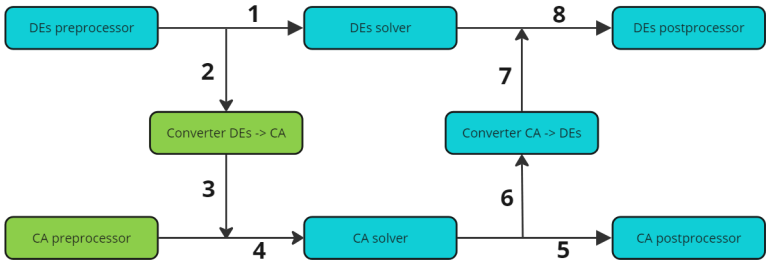


Figure 1. Scheme of the software system for modeling gas flows

In Figure 1, the preprocessor, solver, and postprocessor of differential equations (DEs) are represented by the corresponding modules of the LOGOS Aero-Hydro software suite [4]. The solver and postprocessor for cellular automata are part of the FHP-MP software implementation developed at the Laboratory of Parallel Program Synthesis of the Institute of Computational Mathematics and Mathematical Geophysics (ICMMG) of the Siberian Branch of the Russian Academy of Sciences (SB RAS) [2]. The converters between the differential and cellular automata representations are located at the center of the diagram in Figure 1. These converters are essentially standalone software modules whose primary function is to convert the computational model from one format to another.

Subsystems 1–8 of the software system, illustrated at the top of Figure 1, are the most popular, understandable, and familiar to researchers [5]. Subsystems 4–5 of the software system, located at the bottom of Figure 1, represent a classical modeling scheme adapted to cellular automata.

The main idea of this work is to combine two modeling approaches: the mesh models of the LOGOS Aero-Hydro software suite and the FHP-MP cellular automata. This will attract the attention of researchers and applied engineers using finite element modeling packages to the cellular automata approach. Specifically, paths 2-3-5 and 2-3-6-7 (see Figure 1) will allow a user unfamiliar with the cellular automata approach to utilize the cellular automata solver. As part of this work, a converter module for transforming differential representations into cellular automata, as well as a cellular automata preprocessor, were developed. These modules are marked in green in Figure 1.

1.2. Preprocessor and converter. The cellular automata preprocessor is a standalone software module implemented in C++ using the CATlib library [6]. Its primary function is to automatically convert the initial and boundary conditions from a raster image representing the computational domain into an input file for the FHP-MP cellular automata solver. The CA-preprocessor has the following features:

- A bitmap image in the specified format with a description of the subject area is submitted to the input;
- The cell types (wall, environment, etc.), initial concentration, and boundary conditions are set using RGB channel values of the input raster in the specified format;
- As an output, a data file is generated with a description of the modeling area in the format of the input file for the solver of the FHP-MP software implementation.

Similarly, the converter from differential representation to cellular automata is a separate program. The computer implementation of the converter takes as input the mesh of the model under study, obtained from the LOGOS Aero-Hydro package in the ".ngeom" format [7]. As output, it generates a data file describing the modeling domain in the format required by the FHP-MP software solver. The resulting data file is binary and its format is completely regulated by the CATlib library.

Two new algorithms serve as the basis for the developed software modules: an algorithm for converting a raster image into a cellular array, and an algorithm to convert mesh geometry into a cellular array. Descriptions of these algorithms are provided in the following section.

2. Employed algorithms

To further describe the developed algorithms, it is necessary to clarify the basic concepts of the FHP-MP model.

The cellular automata of the FHP-MP model will be denoted as a triple of objects (W, A, N), where $W = \{w_1, w_2, \ldots\}$ is the set of cells. Each cell $w \in W$ is assigned a corresponding finite automata A. For each cell $w \in W$, there is a defined ordered set

$$N(w) = \{N_i(w)\colon N_0(w) = w,\ N_i(w) \in W\ \&\ d(w, N_i(w)) = 1,\ i = 1, \ldots, b\};$$

the elements of which are neighbors of cell w and are called its adjacent cells or neighbors. Furthermore, $d(w_1, w_2)$ is the distance between w_1 and w_2, where $w_1, w_2 \in W$. The constant b is the number of neighbors of cell [2].

In the FHP model (Figure 2), the cells can be one of four types: environment, wall, inlet, outlet.

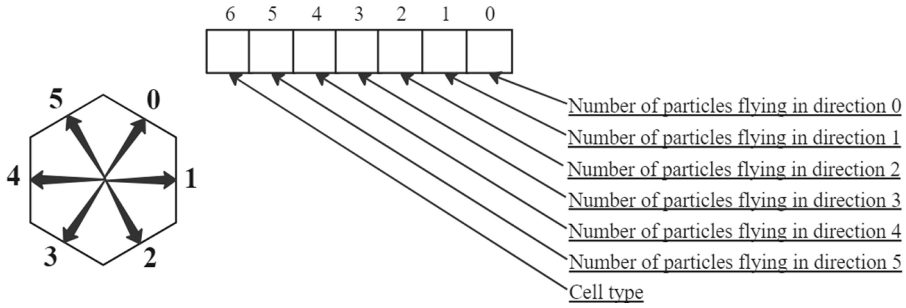


Figure 2. Cell of the FHP-MP model

2.1. Algorithm for converting a raster image into a cellular array. To implement the cellular automata preprocessor module of the FHP-MP model, an algorithm was developed for converting a raster image into a hexagonal model grid. The algorithm consists of the following steps.

1. A hexagonal mesh is superimposed on the original raster, and a rectangle is described around each cell, as shown in Figure 3.
2. Within the area of the rectangle, nodes of an orthogonal grid are established with a step size of $K_X/\texttt{gridSteps}$ (see Figure 3), where K_X is the size of the hexagonal cell, and `gridSteps` is the number of partitions in the orthogonal grid.
3. An iteration is performed over all nodes of the orthogonal grid within each rectangle. For each node, its geometric membership within the hexagonal cell, which is circumscribed by this rectangle, is determined. In Figure 3, nodes belonging to the cell are marked in blue, while those outside are marked in red.
4. During the rectangle traversal, the type of the model cell is determined: environment, wall, inlet, or outlet. The number of internal nodes for the cell is calculated by their types. The type of the pixel where the node is located determines its membership to a specific type.
5. After traversing the nodes of the rectangle, the hexagonal model cell is assigned the type of the most common type of nodes trapped inside (environment, wall, inlet, outlet).

In the above-mentioned algorithm, information regarding the type of pixel and initial concentration is derived based on the values of the RGB channels in the original raster image. The red channel is utilized for determining the type of pixel, while the other channels are employed to determine the initial concentration of particles. In the software implementation of this algorithm, the correspondence between the pixel types and the values of the RGB red channel is set in the configuration file (Figure 4).

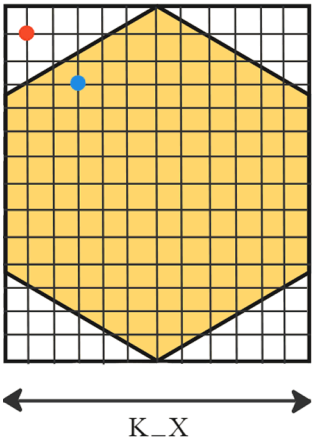


Figure 3. Model cell with superimposed orthogonal grid (`gridSteps` = 12)

```
[imageParametrs]
path_input_file ../input.jpg
K_X 1.00
gridSteps 12
countTypes 16
type0 192,255
type1 64,127
type2 128,191
type3 0,0
type4 0,0
type5 0,0
type6 0,0
type7 0,0
type8 0,0
type9 0,0
type10 0,0
type11 0,0
type12 0,0
type13 0,0
type14 0,0
type15 1,63
```

Figure 4. Example of a CA preprocessor configuration file

In the configuration file shown in Figure 4, there are 16 types of cells, for the first type (`type0`) the range of values of the red RGB channel is from 192 to 255. For the second type (`type1`) — from 64 to 127. For the third one (`type2`) — from 128 to 191 and for the last one (`type15`) — from 1 to 63. The remaining cell types (from `type3` to `type14`) are reserved and are fictitious. This is done so that, with further expansion of the system's functionality, it would be possible to add new types of cells and use them in modeling. Here, `type0`, `type1`, `type2`, and `type15` correspond to the cells of the environment, the inlet, the outlet, and the wall, respectively.

At each iteration, after calculating the characteristics of the cell, data about it is stored in the corresponding element of the cell array. The result of the algorithm is a representation of the computational domain in the form of a cellular array.

2.2. Algorithm for converting mesh geometry into cellular array. The input data for the algorithm is a computational finite element mesh of the model in the ".ngeom" format (Figure 5). This format for storing mesh geometry is widely used in the LOGOS Aero-Hydro software package.

The ".ngeom" format defines the structure of a text file consisting of several interrelated blocks. Let us consider the blocks used in the algorithm.

1. A block containing the finite element numbers and their corresponding type (a label indicating the finite element's belonging to a specific group) will be referred to as the "element-type" block.
2. A block containing the finite element numbers, the number of nodes in the mesh, and the node numbers that belong to a specific finite element will be referred to as the "element-node" block.

```
//_____________________________________________________________
// block node-coordinates
1 -6.0000000000000000 -6.0000000000000000 +0.0000000000000000
2 +1.0000000000000000 -6.0000000000000000 +0.0000000000000000
3 +0.9999999999954881 +6.0000000000000000 +0.0000000000000000
4 -6.0000000000000000 +6.0000000000000000 +0.0000000000000000
...
5964 +17.5298559672517700 +0.0019794938534883 +1.0000000000000000
//_____________________________________________________________
// block element-node
1 4 9 12 363 239
2 4 153 2143 3608 200
3 4 239 363 3608 2143
4 4 154 2144 3607 201
...
11622 4 595 19 20 626
//_____________________________________________________________
// block element-type
5761 11419 plane
5762 11420 plane
...
5811 11469 in
5812 11470 in
...
5821 11479 Wall-0-4
5822 11480 Wall-0-4
...
5861 11519 Wall-0-5
5862 11520 Wall-0-5
...
5901 11559 out
5902 11560 out
...
```

Figure 5. Example of a file in ".ngeom" format

3. A block containing the numbers of the mesh nodes and their coordinates. Let us denote this block by "node-coordinates".

The algorithm for translating from a finite element representation to a CA can be divided into two functional stages. At the first stage, data is read from the ".ngeom " file, and at the second stage, the mesh geometry is directly transformed into the geometry of a cellular automata.

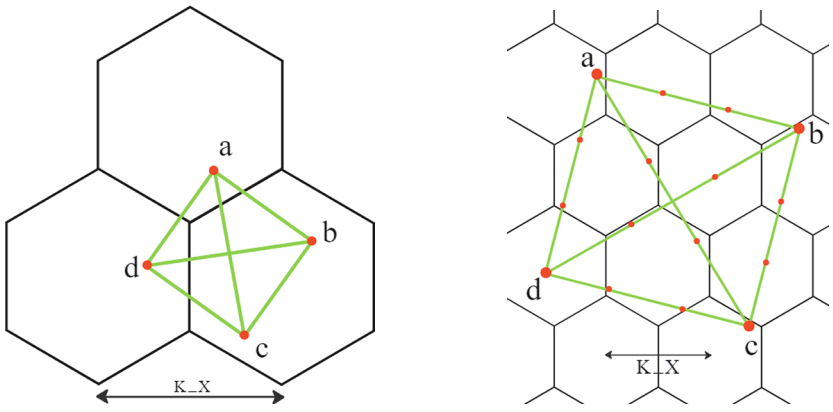
First stage:

1. Read and store data from the "element-type" block.
2. For each finite element, find the corresponding entry in the "element-node" block.
3. For each node, find the corresponding coordinates from the "node-coordinates" block.

After the first stage of the algorithm, a structured data set is obtained. Each element of this dataset contains the identifier of a finite element, its type and the coordinates of its nodes. This dataset will be referred to as the "mesh".

Second stage:

1. In the "mesh", find the minimum x and y coordinate values of the nodes ($xMin$, $yMin$ respectively). Increase the coordinate values of all nodes in the "mesh" by the absolute values of $xMin$ and $yMin$ for x and y respectively. At the same time, the z coordinate is discarded for each node, since the FHP-MP model is two-dimensional. Thus, the projection of the "mesh" onto the XY plane is obtained and simultaneously normalized.
2. For each specific finite element of the "mesh", segments are constructed, the vertices of which are the nodes of this element.
3. Divide the length of each segment by the size of the model's hexagonal cell: `countSteps` $= L/K_X$. Here, `countSteps` is a positive integer, L is the segment length, K_X is the size of the model cell.
4. If `countSteps` $= 0$ (Figure 6a), then for the ends of the segment, determine the cells in which they are geometrically located. Assign to these cells the type of the finite element from the "mesh" to which the segment belongs. If `countSteps` is non-zero (Figure 6b), then it determines the number of splits of the segment. Divide the segment into `countSteps` equal "sub-segments". For each vertex of the "sub-segments", determine which model cell it geometrically falls into and assign the corresponding type to this cell.



a b

Figure 6. Square finite element of the "mesh" and topology of the FHP-MP model: a — element sides are smaller than the size of the model cell; b — element sides are larger than the size of the model cell

3. Testing of the software system

3.1. Problem of sudden expansion in gas pipeline flow. Several well-known gas dynamics problems were selected to test the preprocessor module, including the problem of sudden expansion in gas pipeline flow (Figure 7).

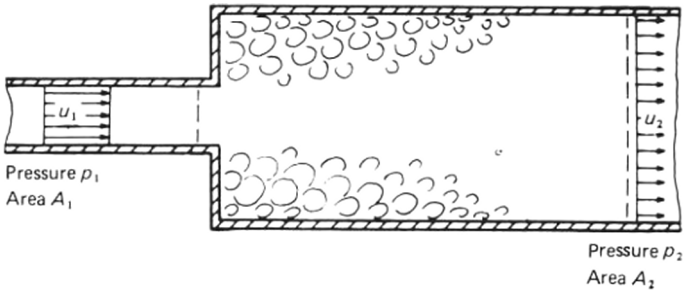


Figure 7. Sudden expansion in gas pipeline flow (taken from [8, Fig. 7.8, p. 261])

The preprocessor input was a raster image, created using a graphics editor, which represented the geometry of the flow expansion problem. Due to the symmetry of the problem, the gas pipe was modeled as expanding only in the top half. The initial image had a resolution of 514 by 232 pixels. The K_X parameter of the configuration file, which indicates the size of the model cell, was set to 1. The cellular automata model parameters that determine the number of particles emitted by the inlet and absorbed by the outlet per iteration were set to 10 and 3 respectively. The simulation was conducted for 2,000 iterations of the cellular automata. The following transformation of the computational domain in the cellular automata was obtained (Figure 8).

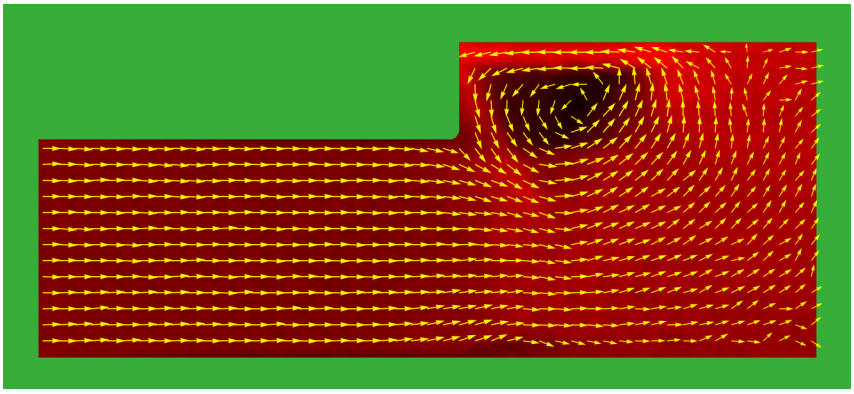


Figure 8. CA-region of the gas pipeline flow expansion problem after 2,000 computational iterations

The FHP-MP model postprocessor generates an output raster describing the computational domain according to the following rules: high-pressure areas are bright (bright red), low-pressure areas are dark (black); particle velocity vectors are indicated by yellow arrows, with the magnitude of the velocity being proportional to the length of the arrow.

As seen in Figure 8, a low-pressure area forms in the upper part of the gas pipeline during flow expansion. In this zone, flow separation from the pipeline walls occurs, leading to the formation of a vortex. Such behavior of the gas flow within the computational domain fully corresponds to the known analytical solution [8].

3.2. Problem of flow around a circular cylinder. As one of the test cases for verifying the functionality of the conversion module, the problem of flow around a circular cylinder was selected. The simplicity of the geometry of the study region, the known analytical solution, and the high level of interest from researchers in the field of gas and hydrodynamics were key factors in choosing this problem as a test.

The geometry of the problem was defined in the LOGOS Aero-Hydro package (Figure 9). The finite element mesh was saved in the "karman.ngeom" file and fed into the converter module. Four types of elements and their corresponding types of cellular automata in the FHP-MP model were defined as parameters in the converter's configuration file: Inlet-1, Outlet-2, Wall-3, Symmetry-4.

It can be seen that the conversion of the computational domain's geometry is performed correctly (Figure 10).

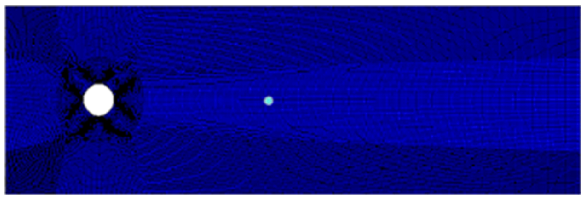


Figure 9. Geometry of the computational domain of the circular cylinder flow problem in the LOGOS Aero-Hydro package

Figure 10. Geometry of the computational domain of the circular cylinder flow problem in the FHP-MP model

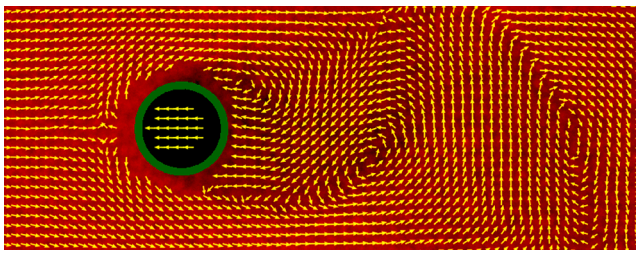


Figure 11. The result of calculating the circular cylinder flow problem in the FHP-MP model after 18,000 iterations

It is worth noting that in the cellular automata model, the types of cells for the outer contour and the inner part of the circular obstacle differ: inside are "environment" type cells, and outside are "wall" type cells. In the LOGOS Aero-Hydro computational model, the cylindrical obstacle is represented only by an outer layer of elements (Wall-3). This discrepancy in computational models is due to the nature of the conversion algorithm's operation and does not affect the calculation results.

The problem was solved in both cellular automata and finite element formats. Using the FHP-MP cellular automata, a turbulent flow pattern of gas streams around the circular obstacle was successfully simulated (Figure 11).

The cellular automata array size was 2,566 by 721 cells. At least 10,000 iterations of the cellular automata transition function were required to complete the transient processes. The final calculation result, shown in Figure 11, was obtained after 18,000 iterations.

The final simulation image obtained in the LOGOS Aero-Hydro package, as well as the result of a real experiment captured by a stationary camera, are presented in Figures 12 and 13, respectively.

As a result of the testing, successful transfer of the computational domain's geometry from a mesh representation to a cellular automata format was achieved. Qualitative similarity of gas flow patterns was observed in both the LOGOS Aero-Hydro package and the FHP-MP model, as well as with the results of the real experiment.

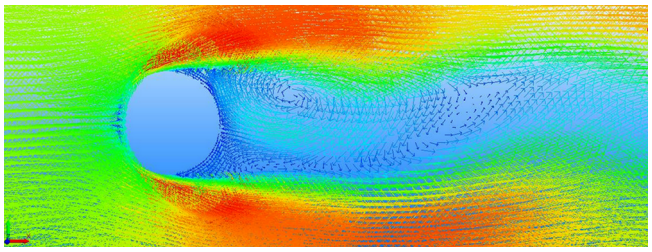


Figure 12. The result of the calculation of the circular cylinder flow problem in the LOGOS Aero-hydro package

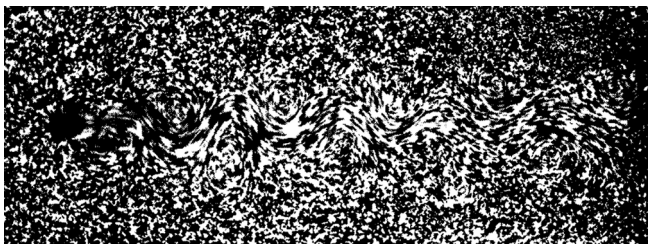


Figure 13. A double row of alternating vortices behind a circular cylinder (taken from [9, Fig. 31, p. 68])

Conclusion

The following components were developed: an algorithm for converting raster images into a cellular automata array in the FHP-MP model and an algorithm for converting the mesh geometry of a finite element model from the LOGOS Aero-Hydro package into the FHP-MP cellular automata. These algorithms were implemented in C++ as standalone software modules of a preprocessor and converter.

The developed modules, along with the solver and postprocessor of the FHP-MP model, form an integrated system for gas flow modeling (Figure 1). Through the converter, this system can be integrated with the LOGOS Aero-Hydro package.

The developed modules and the overall modeling system were tested on several basic gas dynamics problems. The results of the computational experiments exhibited qualitative agreement with the analytical and numerical benchmarks.

Future work on the gas flow modeling software system is planned. One area of development is to expand the functional capabilities of the converter module. It is intended to consider not only the geometry of the computational model but also some of its quantitative characteristics (e.g., velocity, pressure). This enhancement will allow for the assessment of not only qualitative similarities and differences between solutions in the cellular automata and finite element models but also enable correspondence between their quantitative characteristics.

References

[1] Theory of Self-Reproducing Automata / J. Von Neumann. — Urbana: University of Illinois Press, 1966.

[2] Medvedev Yu.G. An extension of the cellular-automaton FHP-I flow model to the FHP-MP multiparticle model // Vestnik Tomskogo gosudarstvennogo universiteta. — 2009. — No. 1(6). — P. 33–40 (In Russian).

[3] Frisch U. Lattice-Gas automata for Navier-Stokes equations / U. Frisch, B. Hasslacher, Y. Pomeau // Phys. Rev. Lett. — 1986. — No. 56.

[4] Betelin V.B., Shagaliev R.M., Aksenov S.V., et al. Mathematical simulation of hydrogen-oxygen combustion in rocket engines using LOGOS code // Acta Astronautica-96. — 2014. — P. 53–64.

[5] Basov K.A. ANSYS for Designers. — Moscow: DMK Press, 2009 (In Russian).

[6] Medvedev Yu.G. Architecture of the Cellular Automata topologies library // Bull. Novosibirsk Comp. Center. Ser. Computer Science. — Novosibirsk, 2022. — Iss. 46. — P. 27–41.

[7] Lopatkin A., Loginov I. LOGOS software package. Integration of some file formats for data storage // Proc. ST conference "Molodej v nauke". — Sarov, 2011 (In Russian).

[8] Mechanics of Fluids / B. S. Massey, J. Ward-Smith. — 7th ed. — Taylor & Francis, 1998.

[9] Aerodynamics / T. Karman. — McGraw-Hill Paperback, 1963.