METALLURGY AND FOUNDRY ENGINEERING – Vol. 38, 2012, No. 1
http://dx.doi.org/10.7494/mafe.2012.38.1.43

Michał Szucki*, Józef S. Suchy**

A VOXELIZATION BASED MESH GENERATION ALGORITHM FOR NUMERICAL MODELS USED IN FOUNDRY ENGINEERING

1. INTRODUCTION

Numerical modeling has become one of the basic tools used in several fields of science and industry. The same is in the foundry engineering where computer simulation programs have been successfully applied for many years. Due to them, it is possible to assess technology correctness without the necessity of performing costly and time-consuming industrial tests. Thus, computer programs are used in order to analyze temperature changes, microstructure formation, and liquid metal motion inside the mould [1, 2]. Simulation software applied in the foundry industry is based on various physical and mathematical grounds and often on very different numerical solutions. A novelty in this scope is the application of the lattice Boltzmann method (LBM) – being developed by the authors, for modeling mould filling and casting crystallization processes [3, 4]. One of the main problems hindering the wider applications of these type of alternative solutions is the space discretization issue called the mesh generation process.

Typically, the extended algorithms which allow for the transformation virtual three-dimensional geometry (originated the most often from the CAD systems) into the cell grid form are implemented in commercial simulation software. This type of tools are often highly complex and able to create the mesh composed of several subdomains corresponding to various elements of casting technology (mould, cast, core, feeder etc.) and the possibility of its local refinement.

On the other hand, commercial mesh generators are characterized by a closed structure (the generated mesh can be used in a given simulation environment only) and the applied there solutions are not universally available. On account of these reasons, ende-avors to develop a complex method – allowing of virtual three-dimensional geometry to the cell form, undertaken in this study, seems to be fully justified.

* M.Sc.,** Prof., PhD., D.Sc.: AGH University of Science and Technology, Faculty of Foundry Engineering, Krakow, Poland; e-mail: mszucki@agh.edu.pl

The generation of computational grids is a very broad subject, which combined with the mentioned above fact of applying a different mathematical or numerical approach to the modeling of foundry processes, forces the necessity of narrowing this problem. The authors present, in the hereby paper, the solution allowing the generation of regular computational meshes with cubic cells (being applied in the lattice Boltzmann method and in e.g. the finite difference method), where as the initial data the three-dimensional virtual geometry (stored in the STL files) was assumed.

2. STL FILE FORMAT

The three-dimensional geometry (3D) is represented in the computer memory by means of flat objects, the so-called polygons (usually triangles). These polygons joined together form the surface, which 'surrounds' 3D objects. By writing three coordinates for each vertex of each triangle it is possible to store such virtual solids in the computer memory.

The STL file is one of the most popular data formats for storing three-dimensional geometry. It found its application in several domains where virtual 3D objects are used, including also foundry engineering. There are two types of STL files: binary and ASCII. In both cases these data structures store the list of vertex coordinates for each triangle and the coordinates of the normal vector to its surface (this vector is used in three-dimensional graphics, among others, for calculating the amount of light falling on this surface). The structure of the text (ASCII) STL file is presented in Table 1.

Table 1. Structure of the ASCII STL file

Keyword	Variables	Variables type
solid	*name*	*string*
Keyword "solid" is followed by the definition of individual triangles:		
facet normal	$n_x\ n_y\ n_z$	*float*
outer loop		
vertex	$v1_x\ v1_y\ v1_z$	*float*
vertex	$v2_x\ v2_y\ v2_z$	*float*
vertex	$v3_x\ v3_y\ v3_z$	*float*
endloop		
endfacet		
List of triangles is concluded with the keyword: "end solid":		
endsolid	*name*	*string*

Thus, the discussed here mesh generation method will operate based on the list of vertex coordinates of individual triangles imported from the STL file.

3. VOXELIZATION

Voxelization is the 3D virtual geometry transition process from the form of the polygonal surface to the regular mesh with cubic cells (Fig. 1). The term "voxel" refers to the computer 2D graphics where a picture constitutes a set of points of the proper color, called pixels. Thus, the voxel can be treated as the three-dimensional pixel characterized by its location in the space discretized – in three directions (x, y, z) – by equally spaced parallel planes.

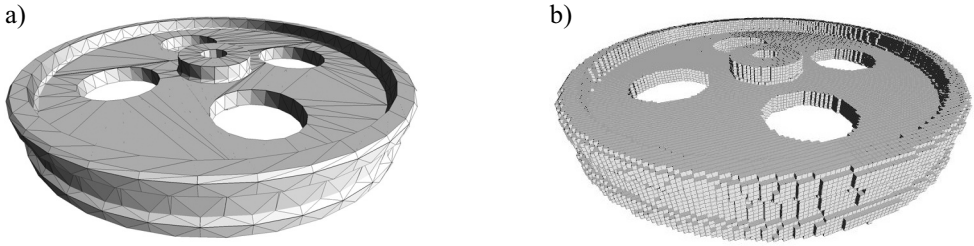

Fig. 1. *Comparison of three-dimensional solids before (a) and after the voxelization process (b)*

The grid created as the result of the voxelization process, at the proper assumptions, can be successfully used in the lattice Boltzmann method as also in other numerical schemes. In the literature usually the so-called GPU based voxelization algorithms, can be found [5, 6]. These methods operate on the grounds of the z-buffer (also known as the depth buffer) of the graphic card, which stores (in two-dimensional form) information on distances in between 3D objects and their mutual overlapping on the three-dimensional scene.

The advantage of the GPU based algorithms is first of all their high effectiveness (they operate practically in real time). On the other hand, the application of graphical solutions is closely related to the computer hardware configuration. In addition the adaptation of this type of method for mesh generation in computer modeling by improving their functionality, is very difficult.

4. CPU BASED VOXELIZATION ALGORITHM

As a point of departure for this study the authors applied the solution presented in paper [7], where a detailed description of the voxelization algorithm can be found. This method enables the effective generation of the cubic cells mesh without the necessity of taking any information from the graphic card (all calculations are CPU based). This allows, in practice, performing the voxelization process independent of the computer hardware configuration.

The main problem in this type of solutions is establishing whether the given cell belongs to the object (is inside it) or not (is outside the object). To this end the so-called raycasting method is used in the presented algorithm. Counting the number of times the casted ray (virtual straight line) intersects the triangles forming the three-dimensional solid, the proper classification of each cell is possible.

The discussed voxelization algorithm, in a general outline, consists of the following stages:

Determination of the so-called bounding box – the smallest rectangular prism into which the considered three-dimensional solid fits. This is done by the determination of the minimum and maximum values of the triangle vertex coordinates for each of the three directions x, y and z.

Partitioning of the bounding box with the use of the quadtree. This partitioning is being done on the x, y plane of the bounding box ($z = 0$). Each node of the quadtree corresponds to an axis-aligned box. These boxes are formed by the 'extrusion' of each leaf node to the height corresponding with the height of the bounding box. Each leaf of the quadtree contains a list of triangles that intersect the corresponding box. Partitioning of the analyzed space with the use of the quadtree is carried out up to the moment, when the number of triangles stored in each leaf node is smaller than the given value or the defined tree depth is reached.

Defining the grid parameters. The user determines the grid step (voxel size) Δx. On this basis the rectangular grid filled with cubic cells (of a side length Δx) surrounding the bounding box (determined in the first step), is created.

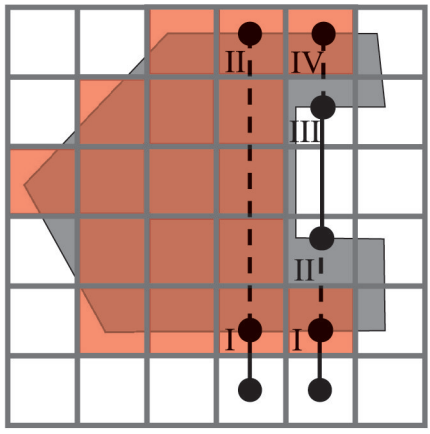

Fig. 2. *Scheme illustrating the way of determination whether the given grid cell belongs to the solid. Roman numerals determine the successive rays intersections with the object surface (gray). Grid cells, which are inside the solid, are marked red*

Determination whether the given grid cell belongs to the object – raycasting. The mentioned rays are projected, parallel to the z-axis, from the center each cells. In order to speed up calculations it is done only for one layer of cells in the x, y plane (with the lowest z value).

Thus, for the grid consist a number of cells equal to j, k, l in each direction, only $j \cdot k$ rays are casted. Knowing the coordinates x and y of the casted ray the leaf of quadtree, to which this ray belongs, can be localized. Then the ray intersections with the triangles stored in the given leaf of quadtree are counted. As long as this number is odd the successive cells of the grid being on the given ray path (in the direction of the z axis) belong to the virtual object, otherwise they lie outside (Fig. 2).

5. VIRTUAL 3D GEOMETRY ERRORS

The STL file structure does not guarantee that the object stored there is of the proper structure. Errors in the three-dimensional geometry, first of all a discontinuity on its surface, can cause that the obtained grid will not properly represent the original solid. This problem is especially essential in the case of applying the discussed solutions in the foundry engineering where the initial STL files originate not only from the CAD systems but also from 3D scanners or the computer tomography. In a case of this type of devices there is a high probability, that the virtual geometry surface will contain discontinuities (cracks), which should be removed in the proper software before starting the grid generation process (Fig. 3).

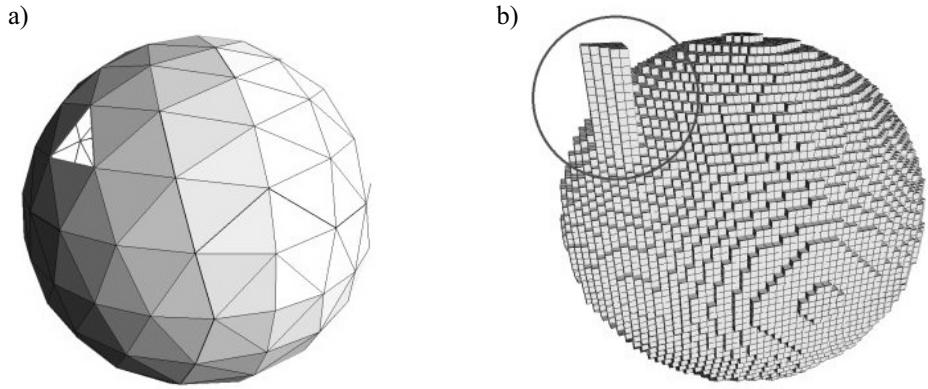

Fig. 3. *Comparison of the solid of a discontinuous structure (a) and the cell grid generated on its basis (b)*

Extending the presented algorithm with the possibility of controlling the correctness of the three-dimensional geometry is relatively simple and reduces to verify whether the total number of the intersections of each ray with triangles being on its path – is even. If this condition is fulfilled the geometry written in the STL file is of a continuous structure, while otherwise the triangle surface contains errors. This is related to the fact that each space fragment belonging to the object must be limited in an arbitrary direction by two walls, which is illustrated in Figure 4.

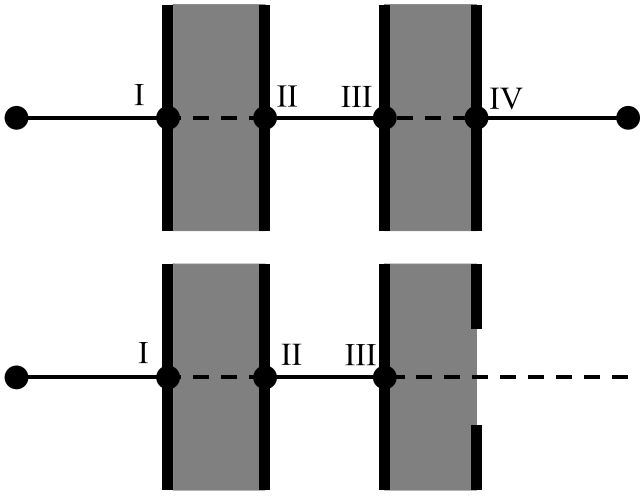

Fig. 4. *Schematic ray path for virtual geometry of the correct structure (upper Figure) and for the case when there are discontinuities on its surface (lower Figure). Roman numerals mark ray intersections with the object walls (triangles)*

6. MULTI-SUBDOMAIN APPROACH

In the case of the simulating processes occurring in the casting production the analyzed system is not homogeneous but consists of several subdomains such as: mould, cores, cast area etc., which should be treated separately (Fig. 5). Such a situation must be reflected in the computational grid, which should be also divided into several regions identified by the state of individual cells. In the typical simulation software applied in the foundry engineering each subdomains of the model system is read from a separate STL file. Tools of this type enable obtaining the coherent computational grid, which cells are assigned to the relevant category corresponding with the individual technological elements.

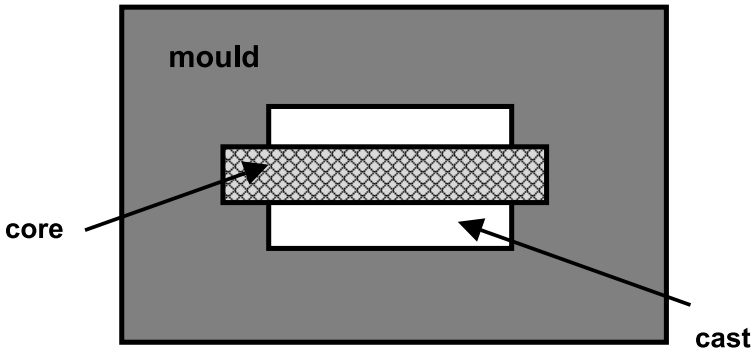

Fig. 5. *Schematic presentation of the model system consisting of several subdomains*

The simplest method of generating one computational grid for several subdomains is the formation of triangles lists located in the separated guadtree sets for each region. In such a case the bounding box should be also determined for each object separately. Next, the grid of cubic cells, which includes all taken into account subdomains, is determined. At the raycasting stage each ray intersection with triangles stored in all of the quadtree data structures are verified. Due to the knowledge of the bounding box coordinates for individual subdomain, the optimization of this algorithm is possible. Thus, only the quadtree which are located on the given ray path are analyzed.

In certain situations there is a theoretical possibility that after performing the algorithm the given cell will belong to more than one subdomain. In a such case it is necessary to assign it 'manually' to the proper category. An analogous procedure is applied when in between two adhering subdomains a crack of a one grid cell width (which is related to the representation accuracy of 3D geometry in the STL file) is formed, for which the classification to the given subdomain was not determined.

7. GRID REFINEMENT

One of the most important drawbacks of the regular computational grids is the necessity of an application of the relatively dense space discretization for the proper representation of virtual object details. Such a situation is directly related to the increase of computation time and higher demand for the operating memory. This problem is specially essential for the – mentioned in the introduction, lattice Boltzmann method, which requires storing in the computer memory a lot of information for each mesh cell. Several solutions, in which locally refined computational grids are recommended, can be found in the literature [8, 9]. In the case of the LBM based algorithms, as well as other numerical solutions, usually the cell size is selected in such a way as to satisfy the equation:

$$\Delta x_n = \frac{\Delta x}{n} \tag{1}$$

where:

Δx_n – cell size in the refined area,

n – parameters defining the ratio of the cell size between the coarse and fine grid.

The simplest way for achieving this result is to divide cells of the coarse grid into n parts in each axis, before starting the voxelization process. In order to do this the cuboidal area, in which the grid will be refined, must be determined. Its edges should coincide with the coarse grid lines. Unfortunately, in such a case the fine grid area is always in the rectangular form, which in many cases is the reason of generating additional unnecessary cells (Fig. 6).

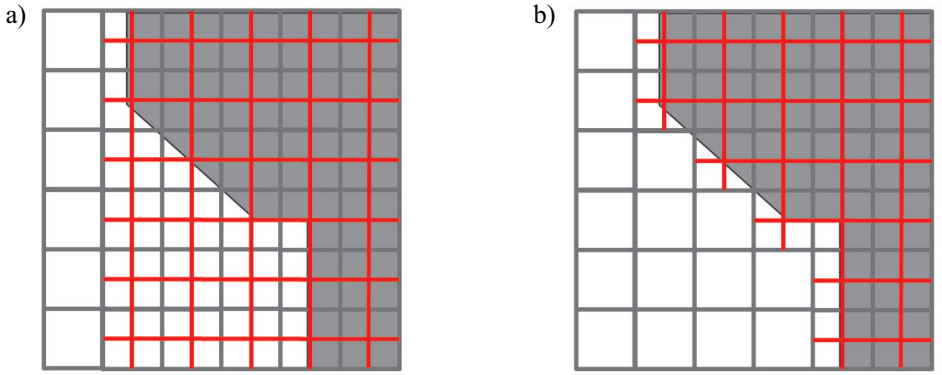

Fig. 6. *Schematic comparison of the grid refinement (red lines) methods in the rectangular area (a) and in the area defined based on the STL file (b). Grey color marks this part of the computational grid, which should be characterized by smaller cells*

A much more effective solution is the mesh refinement in the region limited by the 3D geometry loaded from the STL file.

In such a case the voxelization process should be divided into two stages:

The first stage is analogous as in the previously presented the generation method for multi-subdomain meshes. In this case the system area (loaded from the STL file), which should be characterized by a finer grid (smaller cells) will constitute the additional subdomain. Within this stage the grid cells belonging to this region are properly marked.

The second stage constitutes dividing grid cells, belonging to the refined subdomain to n parts in each of the three directions. Next, the raycasting process should be repeated for the newly formed cells in order to perform their classification.

This method allows practically for a free choice of the three-dimensional space discretization level. In order to increase the accuracy of 3D geometry representation a single cells layer surrounding the fine grid area should be also refined.

8. ANALYSIS OF ALGORITHM EFFICIENCY

The results of efficiency analysis for the presented method are given below. The time of the grid generated for three selected geometries stored in the STL files (Fig. 7) differing in the number of triangles, was compared (Tab. 2). This part of the study encompassed generating single STL file based grids, the formation of grids composed of three subdomains as well as locally refined grids. Calculations were performed on the Dell T7500 Workstation with Intel Xeon X5680 (3.33 GHz) processor and 24 GB RAM.

In the first case the generation time of the mesh consisting of 100^3, 150^3 and 200^3 cells, for three mentioned STL files, was compared (Fig. 8).

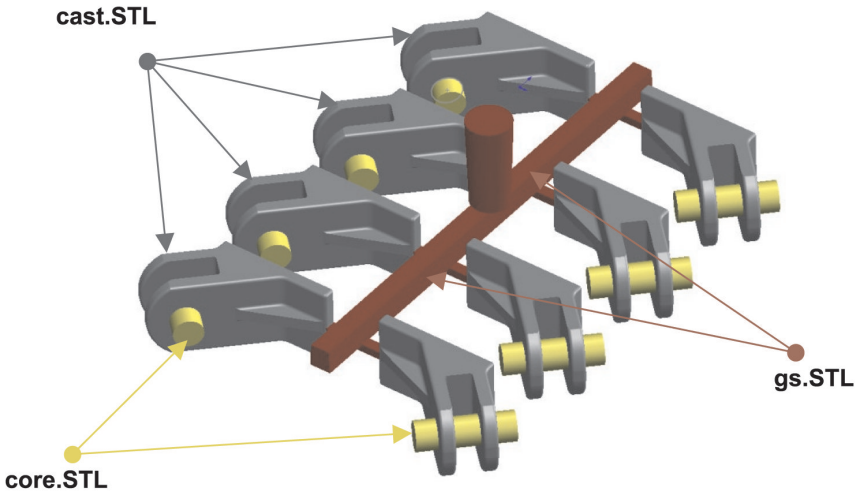

Fig. 7. 3D geometry stored in three STL files applied in the study

Table 2. Parameters of the virtual three-dimensional geometry applied in the study

Part name	File name	Triangle count
Casts	cast.STL	12908
Cores	core.STL	800
Gating system	gs.STL	330

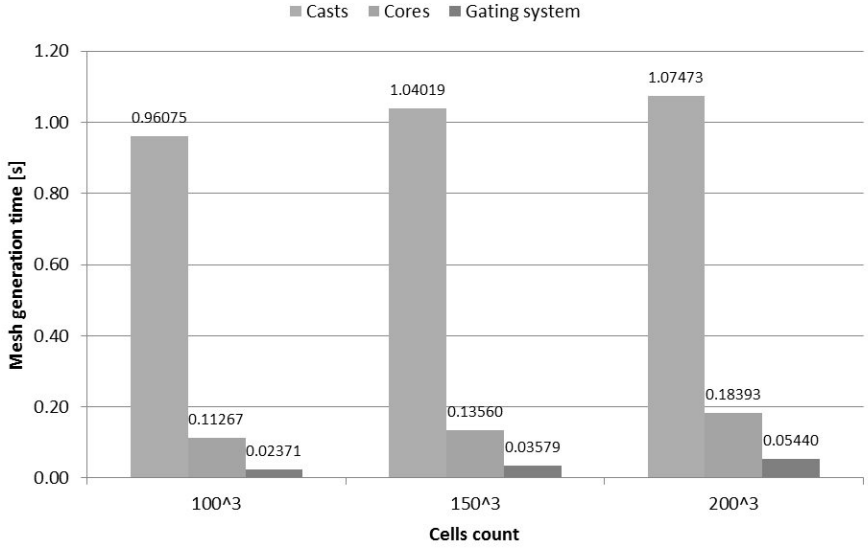

Fig. 8. Comparison of the mesh generation time for the three STL files applied in the study

The next step was aimed at the efficiency analysis of the generation process of the mesh consisting of three subdomain (casts.STL, cores.STL and gs.STL) determined by the 3D geometry (Fig. 9).

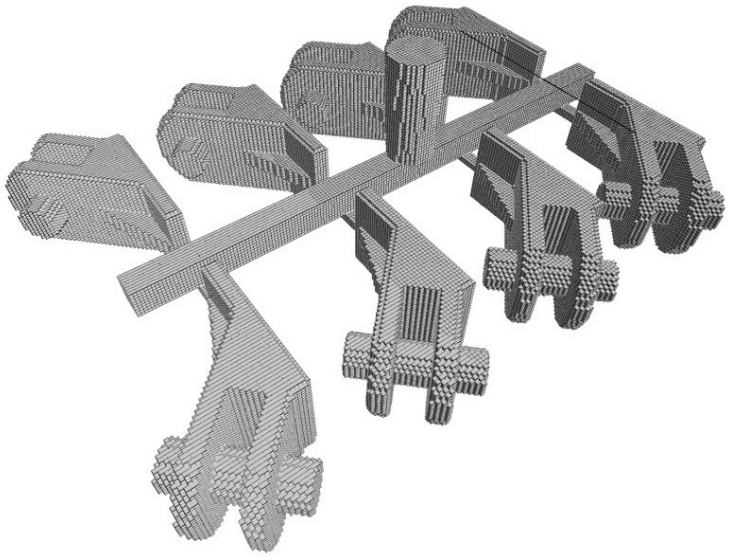

Fig. 9. *Multi-subdomain mesh generated based on for the three STL files applied in the study*

Mesh generation time as a function of the number of cells for this case is presented in Figure 10.

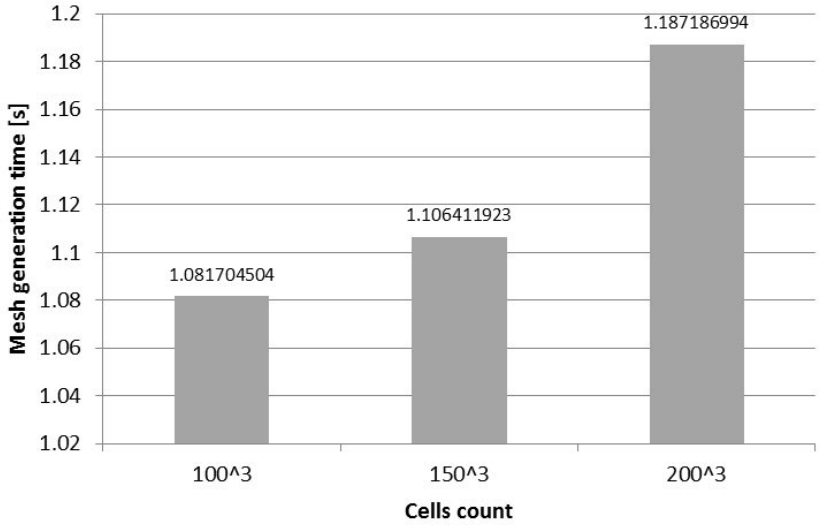

Fig. 10. *Comparison of the mesh generation time for the system consisting of three subdomains*

The last test consisted of a comparison of the mesh generation time for the locally refined mesh. In this case the coarse grid built of 8 millions of cells consisted of three subdomains (cast.STL, core.STL and gs.STL). Then, each cell of the casts area (cast.STL) was divided into three directions two (n = 2), four (n = 4) and six (n = 6) times, respectively. The time of the mesh generation as a function of n parameter is presented in Figure 11.

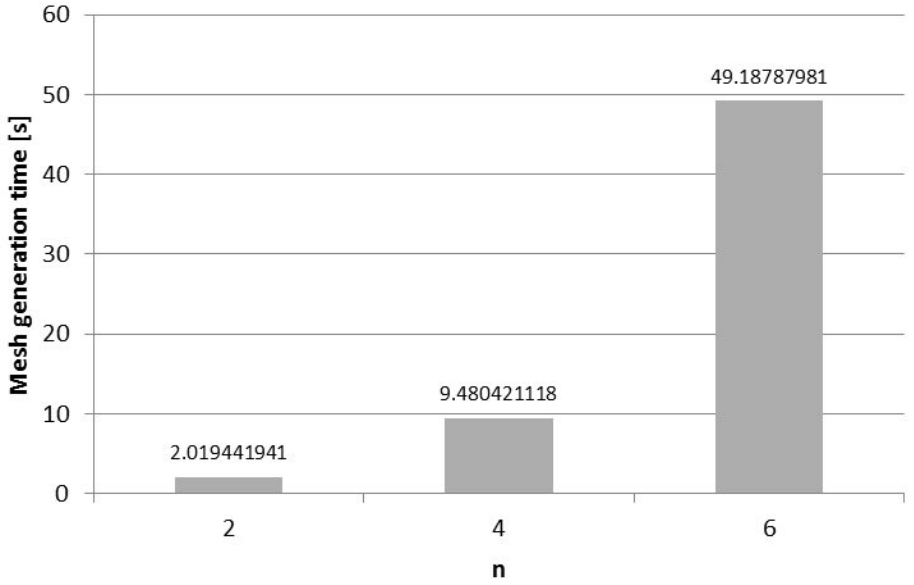

Fig. 11. *Mesh generation time as a function of grid local refinement level n*

On the basis of the performed analyses it can be stated that the mesh generation rate depends mainly on the structure of the three-dimensional geometry stored in the STL file (number of triangles) as well as on the level of the space discretization (number of cells). For the finest grid consisting of 8 million cells, generated on the basis of the three-dimensional virtual object built of 12908 triangles the algorithm work time was about 1.07 seconds. In the situation, when the analogous grid consists of three subdomains (14038 triangles) this time was approximately 10.5% longer. This difference in the case of two multi-subdomains grids built from 1 million and 8 million cells was less than 10% This indicates that the most time consuming stage the algorithm is the process of creating quadtree sets (which is directly dependent on the number of triangles). However, in all cases the method effectiveness is very high and comparable with commercial solutions available on the market.

The separate problem constitutes the generation process of locally refined grid. In such cases the method operation rate strongly depends on the number and the discretization density of the individual subdomains.

9. CONCLUSIONS

The presented method enables the effective generation of regular computational grids. By choosing the adequate space discretization density it is possible to select between the computation time and the accuracy of the three-dimensional geometry representation. The described solution is simple in implementation and can be easily extended with additional features. It enables the mesh generation of practically any complexity and does not require from the user an introduction of additional input data, apart from the STL files geometry and the individual cell size. These advantages allow for the direct application of the pre-sented solutions in problems related to the computer modeling of foundry processes.

Acknowledgements

The authors acknowledge the financial support from The Polish Ministry of Science and Higher Education through the Dean's Grant, AGH No. 15.11.170.423.

REFERENCES

[1] *Piwowarski G., Krajewski W.K., Lelito J.*: Optimization of casting technology of the pressure die cast AZ91D Mg-based alloy, MaFE Metallurgy and Foundry Engineering, 36 (2010) 2, 105–111

[2] *Żak P., Lelito J., Krajewski W.K., Suchy J.S., Gracz B., Szucki M.*: Model of dendrite growth in metallic alloys, MaFE Metallurgy and Foundry Engineering, 36 (2010) 2, 131–136

[3] *Szucki M., Żak P., Lelito J., Suchy J.S.*: Thermal lattice Boltzmann method accuracy analysis for az91 alloy and az91/sic composite, 7th International PhD Conference, conference proceedings (2010)

[4] *Szucki M., Suchy J.S., Żak P., Lelito J., Gracz B.*: Extended free surface flow model based on the lattice boltzmann approach, MaFE Metallurgy and Foundry Engineering, 36 (2010) 2, 113–121

[5] *Karabassi A., Papaioannou G., Theoharis T.*: A depth buffer based voxelization algorithm, Journal of Graphics Tools, 4 (4) (1999), 5–10

[6] *Fang S., Chen H.*: Hardware accelerated Voxelisation, Volume Graphics, (2000), 301–315

[7] *Thon S., Gesquière G., Raffin R.*: A Low Cost Antialiased Space Filled Voxelization Of Polygonal Objects, International Conference Graphicon, conference proceedings (2004)

[8] *Chen H., Filippova O., Hoch J., Molvig K., Shock R., Teixeira C., Zhang R.*: Grid refinement in lattice Boltzmann methods based on volumetric formulation, Physica A, 362 (1) (2006), 158–167

[9] *Thurey N., Rude U.*: Stable free surface flows with the lattice Boltzmann method on adaptively coarsened grids, Computing and Visualization in Science, 12 (2009) 5, 247–263

Received
February 2012