Parallel Tetrahedral Mesh Generation Based on A-priori Domain Decomposition

Evgeny Ivanov

Vom Fachbereich Mathematik
der Universität Kaiserslautern
zur Verleihung des akademischen Grades
Doktor der Naturwissenschaften
(Doctor rerum naturalium, Dr. rer. nat.)
vorgelegte Dissertation.

Erster Gutachter: Prof. Dr. A. Klar
Zweiter Gutachter: Prof. Dr. U. Rüde

D 386

Table of contents

Introduction

An important element of the numerical simulations of partial differential equations by finite-element or finite-difference methods on general regions is a grid which represents the physical domain in a discrete form.

The efficiency of a numerical study of a problem is estimated from the accuracy of the computed solution and from the cost and time of the computation.

The accuracy of the numerical solution in the physical domain depends on both the error of the solution at the grid points and the error of interpolation. Commonly, the error of the numerical computation at the grid point arises from several distinct sources. First, mathematical models do not represent physical phenomena with absolute accuracy. Second, an error arises at the stage of the numerical approximation of the mathematical model. Third, the error is influenced by the size and shape of the grid cells. Fourth, an error is contributed by the computation of the discrete physical quantities satisfying the equations of the numerical approximation. And fifth, an error in the solution is caused by the inaccuracy of the process of interpolation of the discrete solution. Of course, the accurate evaluation of the errors due to these sources remains a difficult task. It is apparent, however, that the quantitative and qualitative properties of the grid play a significant role in controlling the influence of the third and fifth sources of the error in the numerical analysis of physical problems.

Two fundamental classes of grids are popular for the numerical solution of boundary value problems: structured and unstructured.

Many field problems of interest involve very complex geometries that are not easily amenable to the framework of the pure structured grid concept. Structured grids may lack the required flexibility and robustness for handling domains with complicated boundaries, or the grid cells may become too skewed and twisted, thus prohibiting efficient numerical solution. An unstructured grid concept is considered as one of the appropriate solutions to the

problem of producing grids in regions with complex shapes.

However, the use of unstructured grids complicates the numerical algorithm because of the inherent data management problem, which demands a special program to number and order the nodes, edges faces, and cells of the grid, and extra memory is required to store information about the connections between the cells of the mesh. One further disadvantage of tetrahedral unstructured grids, that causes excessive computational work is associated with increased number of cells, cell faces, and edges in comparison with those for hexahedral meshes. For example, a tetrahedral mesh of N points has roughly $6N$ cells, $12N$ faces, and $7N$ edges, while a mesh of hexahedra has roughly N cells, $3N$ faces, and $3N$ edges. Furthermore, moving boundaries or moving internal surfaces in the physical domain are difficult to handle with unstructured grids. As a result, the numerical algorithms based on unstructured grid topology are the most costly in terms of operations per time step and memory per grid point.

The introduction of parallel computers is enabling ever-larger problems to be solved in such areas as Computational Mechanics (CM), Computational Fluid Dynamics (CFD) and Computational Electromagnetics (CEM). The savings in computation time, and in general cost, from these parallel machines for simulations is clearly advantageous. While many solvers have been ported to parallel machines, grid generators have left behind. Still the preprocessing process of mesh generation remains a sequential bottleneck in the simulation cycle.

Grids in excess of 10^7 elements have become common for production runs in CFD [102–106], CEM [100, 101], and CSM. The expectation is that in the near future grids in excess of $10^8 - 10^9$ elements will be required [107]. As mesh cell numbers become as large as this, the process of mesh generation on a serial computer becomes problematic in terms of computational time as well as memory requirements. Especially this is true for applications where remeshing is an integral part of simulations, e.g. problems with moving bodies or changing topologies, the time required for mesh regeneration can easily consume more than 50% of the total time required to solve the problem [107]. Since early 1990s attempts have been made at parallelizing this stage.

Therefore, the need for developing parallel mesh generation technique is

well justified and caused by the following major factors:

- **Lack of memory.** Computational meshes exceeding $2.5 \cdot 10^7$ tetrahedra can not be generated on a single CPU because of memory limitations. Many scientific applications suffer from a lack of memory size. The performance that is achieved by modern processors is often wasted by starvations due to memory bandwidth.
- **Long computational time.** Preprocessing, especially mesh generation, can consume most of the time required to solve the problem. Especially for applications where remeshing is an integral part of simulations grid generation procedure is the main bottleneck.

Objectives of this work are:

1. Development of algorithm for automatic parallel generation of unstructured three-dimensional meshes;
2. Analysis and comparison of algorithms for parallel generation of finite element tetrahedral meshes;
3. Investigation and analysis of methods and criteria of domain decomposition for obtaining good load balancing;
4. Implementation and testing automatic parallel grid generation based upon developed algorithm;
5. Analysis of efficiency of the developed algorithm on real-life problems from CSM.

Dissertation consists of introduction, five chapters, fifty three paragraphs, conclusion and bibliography, it contains 54 pictures and two tables. Size of dissertation is 123 pages. Bibliography has 145 entries.

The current importance of the work is explained in the *introduction*. Work objectives are given and short description of chapters content is introduced.

The *first* chapter gives a general introduction to the subjects of grids. They are classified into two fundamental forms of mesh: structured and unstructured. Advantages and disadvantages of unstructured ones are then described. The chapter outlines some basic approaches to unstructured grid generation

and emphasizes methods based upon Delaunay criterion. Comparative table of different realizations of Delaunay triangulation construction algorithms is given. Problems of the existence of a solution in two and three dimensions are discussed and examples of nontriangulable polyhedra are shown. Finally, overview of works related to unstructured grid generation is done.

The *second* chapter is devoted to parallelization and domain decomposition approaches. Here the reasons for parallelization of mesh generation stage are explained. Advantages and disadvantages of a priori and a posteriori decomposition methods are described. Various domain decomposition approaches such as prepartition along the same direction, recursive prepartitioning, overdecomposition are shown. Examples of different criteria for setting the cutting planes up are given: equal volume of subdomains, equal moment of inertia, equal number of surface elements etc. Then overview of works related to parallel mesh generation and some conclusions are given.

The *third* chapter gives an extensive description of developed parallel mesh generation algorithms. At first, problem and goals of the work are formulated. Then overview of algorithmic steps are given with following detailed explanation in corresponding sections: setting up the cutting planes and load balancing, forming of splitting contour, construction of interface and 2D constrained Delaunay triangulation, splitting along the path of edges, overall domain decomposition and parallel mesh construction. Since at each stage many different solution techniques may be applied, the reasons for choosing one, advantages and disadvantages are discussed.

In the *fourth* chapter the implementation of proposed algorithms is discussed. Adopted parallel computation model (message-passing) and programming model SPMD (Single Program, Multiple Data) are explained. Then programming packages for two-dimensional and three-dimensional triangulations are described. The organization of interprocessor communications based on MPI (Message Passing Interface) [131] along with scheme of parallel work organization of parallel mesh generator are illustrated. Special attention is paid to integration of parallel grid generator with parallel finite element solver. Several real-life examples of computation are given.

The *fifth* chapter discusses results of computations for real-life problems, where femoral, tibial knee prosthesis components and a bearing cap to fix a

crank shaft at a motorblock and others are considered. Decomposition and parallel mesh generation is demonstrated for different mesh sizes and number of processors. Special attention is paid to surface and volume mesh quality. Different aspects of computational efforts related to complexity, computational time, and total area of interfaces are explained. Finally, advantages of parallel grid generator are pointed out and prospective directions of further development are specified as a result of analysis of developed algorithm and of computational results for real-life problems.

Conclusion contains a summary of major results.

Chapter I

General concepts related to unstructured mesh generation

The *first* chapter gives a general introduction to the subjects of grids. They are classified into two fundamental forms of mesh: structured and unstructured. Advantages and disadvantages of unstructured ones are then described. The chapter outlines some basic approaches to unstructured grid generation and emphasizes methods based upon Delaunay criterion. Comparative table of different realizations of Delaunay triangulation construction algorithms is given. Problems of the existence of a solution in two and three dimensions are discussed and examples of nontriangulable polyhedra are shown. Finally, overview of works related to unstructured grid generation is done.

1.1. Grid and solution

An important element of the numerical simulations of partial differential equations by finite-element or finite-difference methods on general regions is a grid which represents the physical domain in a discrete form. In fact, the grid is a preprocessing tool or a foundation on which physical, continuous quantities are described by interpolation functions, which approximate the differential equations interpolating discrete values computed at nodal points. The grid technique also has the capacity, based on an appropriate distribution of the grid points, to enhance the computational efficiency of the numerical solutions of complex problems.

The efficiency of a numerical study of a problem is estimated from the computational costs at a prescribed accuracy.

The accuracy of the numerical solution in the physical domain depends on both the error of the solution at the grid points and the error of interpolation.

Commonly, the error of the numerical computation at the grid point arises from several distinct sources. First, mathematical models do not represent physical phenomena with absolute accuracy. Second, an error arises at the stage of the numerical approximation of the mathematical model. Third, the error is influenced by the size and shape of the grid cells. Fourth, round-off error contributed by the computation of the discrete physical quantities satisfying the equations of the numerical approximation. And fifth, an error in the solution is caused by the interpolation error of the discrete solution. Of course, the accurate evaluation of the errors due to these sources remains a difficult task. It is apparent, however, that the quantitative and qualitative properties of the grid play a significant role in controlling the influence of the third and fifth sources of the error in the numerical analysis of physical problems.

Another important characteristic of a numerical algorithm that influences its efficiency is the cost of solution of (non)linear system of equations. From this point of view, the process of generating a sophisticated grid may increase the computational costs for the numerical solution and encumber the computer tools with the requirement of additional memory. On the other hand, there may be a significant profit in accuracy which allows one to use a smaller number of grid points. Any estimation of the contributions of these opposing factors can help in choosing an appropriate grid. In any case, since grid generation is an important component of numerical modeling, research in this field is aimed at creating techniques which are not too costly but which give a significant improvement in the accuracy of the solution. The utilization of these techniques provides one with the real opportunities to enhance the efficiency of the numerical solution of complex problems. Thus grid generation helps to satisfy the constant demand for enhancement of the efficiency of the numerical analysis of practical problems.

The first efforts aimed at the development of grid techniques were undertaken in 1960s. Now, a significant number of advanced methods have been created: algebraic, elliptic, hyperbolic, parabolic, variational, Delaunay, advancing-front, etc. The development of these methods has reached a stage where calculations in fairly complicated domains and on surfaces that arise while analyzing multidimensional problems are possible. Because of its suc-

cessful development, the field of numerical grid generation has already formed a separate mathematical discipline with its own methodology, approaches and technology.

At the end of 1980s there started a new stage in the development of grid generation techniques. It is characterized by creation of comprehensive, multi-purpose, three-dimensional grid generation codes which are aimed at providing a uniform environment for the construction of grids in arbitrary multidimensional regions.

The grid must be generated for the region of interest to allow the routine computational solution of the equations, and this is still remains a challenging task. When solving three-dimensional (non)linear systems of partial differential equations in domains with complex geometry, the generation of the grid may be the most time-consuming part of the calculation. In fact, it may take more man-hours to generate a grid than it takes to solve discretized system of equations. This is especially true now that the development of codes for the numerical solution of partial differential equations has reached a very high efficiency, while the grid generation field still remains in a nearly teenager stage of its development. Consequently, the meshes still limit the efficiency of numerical methods for the solution of partial differential equations.

1.2. Grid classes

There are two fundamental classes of grid popular in the numerical solutions of boundary value problems: structured and unstructured. These classes differ in the way in which the mesh points are locally organized. In the most general sense, this means that if the local organization of the grid points and the form of the grid cells do not depend on their position but are defined by a general rule, the mesh is considered as structured. When the connection of the neighboring grid nodes varies from point to point, the mesh is called unstructured. As a result, in the structured case the connectivity is taken into account, while the connectivity of unstructured grids must be explicitly described by an appropriate data structure procedure. Additionally, structured grid can be regular and not regular (see Fig. 1.1). In case of not regular structured grid additional memory and efforts are required since sizes of cell edges should be specified due to not regular cell shape.

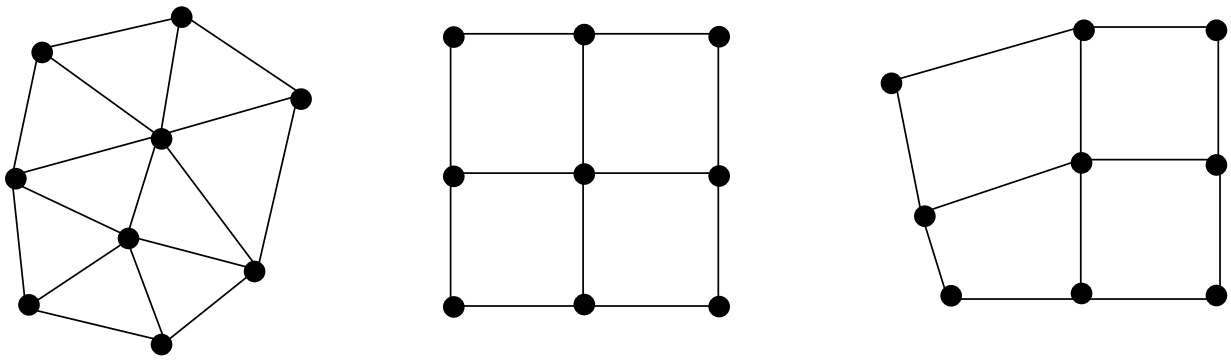


Figure 1.1. Grid types. Left - unstructured grid; middle - structured grid (regular); right - structured grid (not regular).

The two fundamental classes of mesh give rise to three additional subdivisions of grid types: block-structured, overlapping, and hybrid. These kinds of meshes possess to some extent the features of both structured and unstructured grids, thus occupying an intermediate position between the purely structured and unstructured grids.

In the commonly applied block strategy, the region is divided without holes or overlaps into a few contiguous subdomains, which may be considered as the cells of a coarse, generally unstructured grid. And then a separate structured grid is generated in each block. The union of these local grids constitutes a mesh referred to as a block-structured or multiblock grid. Grids of this kind can thus be considered as locally structured at the level of an individual block, but globally unstructured when viewed as a collection of blocks. Thus a common idea in the block-structured grid technique is the use of different structured grids, or coordinate systems, in different regions, allowing the most appropriate grid configuration to be used in each region.

Block-structured grids require partition of the domain into blocks that are restricted so as to abut each other. Overlapping grids are exempt from this restriction. With the overlapping concept the blocks are allowed to overlap, what significantly simplifies the problem of the selection of the blocks covering the physical region. In fact, each block may be a subdomain which is associated with only with a single geometry or physical feature. The global grid is obtained as an assembly of structured grids with which are generated separately in each block. These structured grids are overlap each other, with data communicated by interpolation in overlapping areas of the blocks (Fig.1.2).

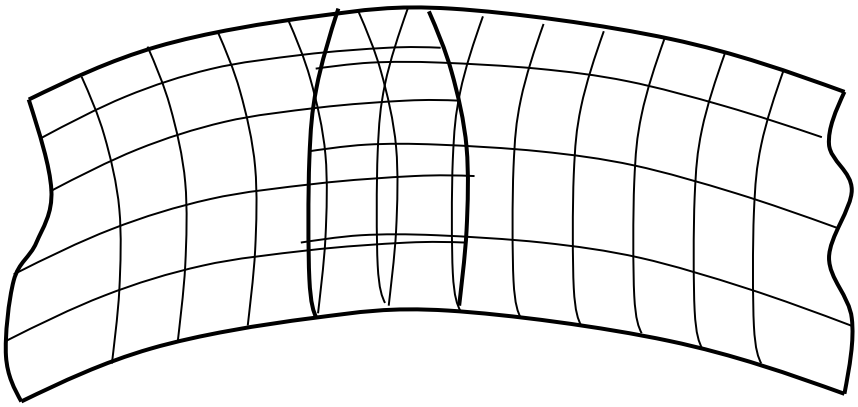


Figure 1.2. Fragment of an overlapped grid.

Hybrid numerical grids are meshes which are obtained by combining both structured and unstructured grids. These meshes are widely used for the numerical analysis of boundary value problems in regions with a complex geometry and with a solution of complicated structure. They are formed by joining structured and unstructured grids on different parts of the region or surface. Commonly, a structured grid is generated about each chosen boundary segment. These structured grids are required not to overlap. The reminder of the domain is filled with the cells of an unstructured grid (Fig.1.3). This construction is widely applied for the numerical solution of problems with boundary layers.

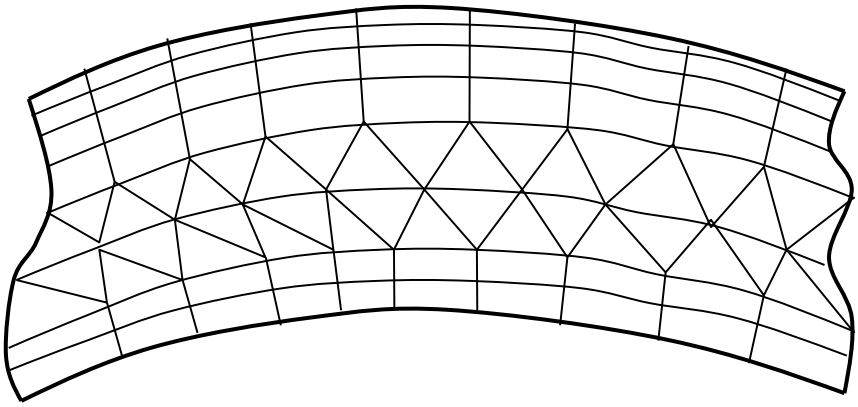


Figure 1.3. Fragment of a hybrid grid.

1.3. Unstructured computational grids

Many field problems of interest involve very complex geometries that are not easily amenable to the framework of the pure structured grid concept. Structured grids may lack the required flexibility and robustness for handling domains with complicated boundaries, or the grid cells may become too skewed and twisted, thus prohibiting efficient numerical solution. An unstructured grid concept is considered as one of the appropriate solutions to the problem of producing grids in regions with complex shapes.

Unstructured grids have irregularly distributed nodes and their cells are not obliged to have any one standard shape. Besides this, the connectivity of neighboring grid cells is not subject to any restrictions; in particular, the cells can overlap or enclose one another. Thus, unstructured grids provide the most flexible tool for the discrete description of a geometry.

These grids are suitable for the discretization of domains with a complicated shape, such as regions around aircraft surfaces or turbomachinery blade rows. They also allow one to apply a natural approach to local adaptation, by either insertion or removal of nodes. Cell refinement in an unstructured system can be accomplished locally by dividing the cells in the appropriate zones into a few smaller cells. Unstructured grids also allow excessive resolution to be removed be deleting grid cells locally over regions in which the solution does not vary appreciably. In practice, the overall time required to generate unstructured grid in complex geometries is much shorter than for structured or block structured grids.

However, the use of unstructured grids complicates the numerical algorithm because of the inherent data management problem, which demands a special program to number and order the nodes, edges faces, and cells of the grid, and extra memory is required to store information about the connections between the cells of the mesh. One further disadvantage of unstructured grids that causes excessive computational work is associated with increased number of cells, cell faces, and edges in comparison with those for hexahedral meshes. For example, a tetrahedral mesh of N points has roughly $6N$ cells, $12N$ faces, and $7N$ edges, while a mesh of hexahedra has roughly N cells, $3N$ faces, and $3N$ edges. Furthermore, moving boundaries or moving internal surfaces of physical domain are difficult to handle with unstructured grids. As a result,

the numerical algorithms based on an unstructured grid topology are the most costly in terms of operations per time step and memory per grid point.

Originally, unstructured grids were mainly used in computational solid mechanics (the theory of elasticity and plasticity), and in numerical algorithms based on finite-element methods. However, the field of application of unstructured grids has now expanded considerably and includes computational fluid dynamics.

1.4. Unstructured grid generation methods

Unstructured grids can be obtained with cells of arbitrary shape, but are generally composed of tetrahedra (triangles in two dimensions). There are three fundamental approaches to the generation of unstructured grids: octree method, Delaunay procedures, and advancing-front techniques.

Octree approach. In the octree approach the region is first covered by a regular Cartesian grid of cubic cells (squares in two dimensions). Then the cubes containing segments of the domain surface are recursively subdivided into eight cubes (four squares in two dimensions) until the desired resolution is reached. The cells intersecting the body surfaces are formed into irregular polygonal cells. The grid generated by this octree approach is not considered as the final one, but serves to simplify the geometry of the final grid, which is commonly composed of tetrahedral (or triangular) cells built from the polygonal cells and the remaining cubes (see Fig. 1.4).

The main drawback of the octree approach is the inability to match a prescribed boundary surface grid, so the grid on the surface is not constructed beforehand as desired but derived from the irregular volume cells that intersect the surface. Another drawback of this grid is its rapid variation in cell size near the boundary. In addition, since each surface cell is generated by the intersection of a hexahedron with the boundary there arise problems in controlling the variation of the surface and cell size and shape.

Delaunay approach. The Delaunay approach connects neighboring points (of some previously specified set of nodes in the region) to form tetrahedral cells in such a way that the circumsphere through the four vertices of a tetrahedral cell does not contain any other point. The points can be generated in two ways; they can be defined at the start by some technique or

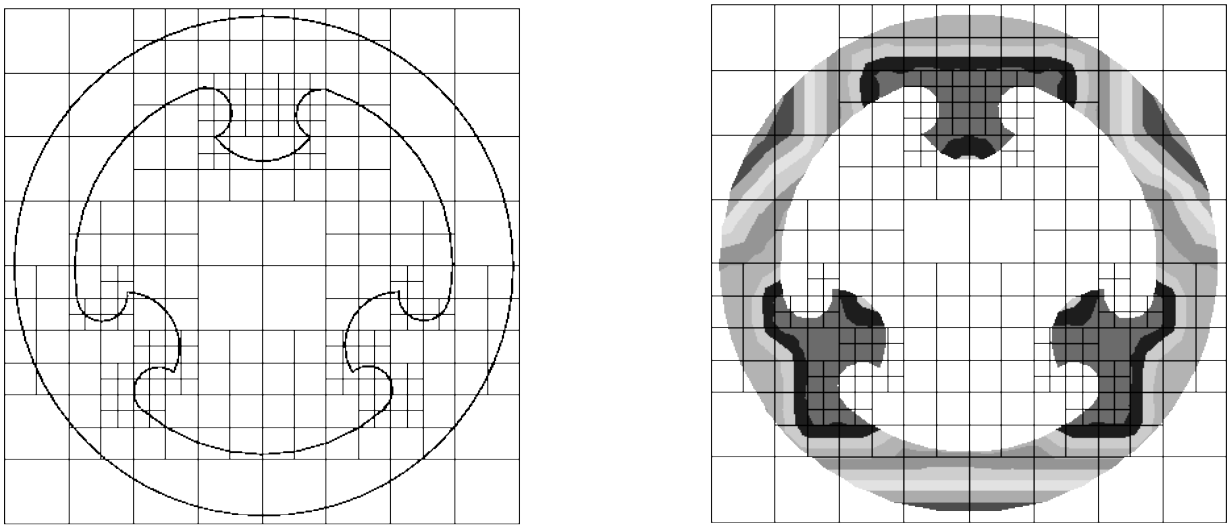


Figure 1.4. Octree approach in two dimensions.

they can be inserted within the tetrahedra as they are created, starting with very coarse elements connecting boundary points and continuing until the element size criteria satisfied. In the latter case a new Delaunay triangulation is constructed at every step using usually Watson's and Rebay's incremental algorithm. The major drawback of the Delaunay approach is that it requires the insertion of additional boundary nodes, since the boundary cells may not become the boundary segments of the Delaunay volume cells. Either the Delaunay criterion must be mitigated near the boundaries or boundary points must be added as necessary to avert breakthrough of the boundary.

Advancing front techniques. In these techniques the grid is generated by building cells progressively one at a time and marching from the boundary into the volume by successively connecting new points to points on the front until all previously unmeshed space is filled with grid cells. Some provision must be made to keep the marching front from intersecting.

To find a suitable vertices for the new cells is very difficult task in this approach, since significant searches must be made to adjust the new cells to the existing elements. Commonly, the marching directions for the advancing front must take into account the surface normals and also the adjacent surface points. A particular difficulty of this method occurs in the closing stage of the procedure, when the front folds over itself and the final vertices of the empty space are replaced by tetrahedra. Serious attention must also be paid to the marching step size, depending on the size of the front faces as well as the

shape of the unfilled domain that is left.

A major drawback remaining for unstructured techniques is the increased computational cost of the numerical solution of partial differential equations in comparison with structured grids.

Methods for unstructured grids were reviewed by Thacker (1980) [1], Ho-Le (1988) [2], Shephard et al. (1988) [3], Baker (1995, 1997) [4,5], Field (1995) [6], Carey (1997) [7], George and Borouchaki (1998) [8], Krugljakova et al. (1998) [9]. An exhaustive survey of both structured and unstructured techniques has been given by Thompson and Weatherill (1993) [10].

1.5. Methods based on the Delaunay criterion

Much attention has been paid in the development of methods for unstructured discretizations to triangulations which are based upon the very simple geometrical constraint that the hypersphere of each n-dimensional simplex defined by $n + 1$ points is void of any other points of the triangulation. For example, in three dimensions the four vertices of a tetrahedron define a circumsphere which contains no other nodes of the tetrahedral mesh. This restriction is referred to as the Delaunay or incircle criterion, or the empty-circumcircle property. Triangulations obeying the Delaunay criterion are called Delaunay triangulations. They are very popular in practical applications owing to the following optimality properties valid in two dimensions:

1. Delaunay triangles are nearly equilateral;
2. the maximum angle is minimized;
3. the minimum angle is maximized.

These properties give us some grounds to expect that the grid cells of a Delaunay triangulation are not too deformed. In Fig. 1.5 triangles which do not satisfy Delaunay criterion are shown on the left. Triangles which satisfy this criterion are in the middle. Example of Delaunay triangulation is illustrated on the right part of the figure.

The Delaunay criterion does not give any indications as to how the grid points should be defined and connected. One more drawback of the Delaunay criterion is that it may not be possible to realize it over the whole region with

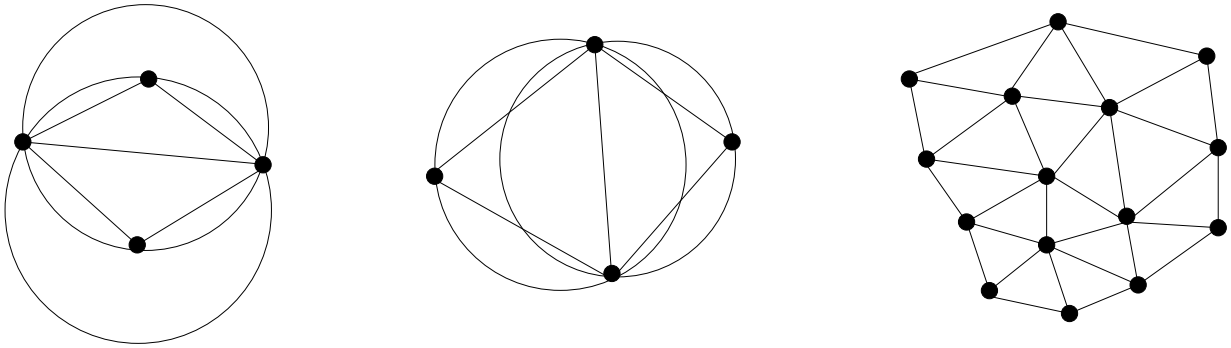


Figure 1.5. Delaunay criterion. Left - triangles which do not satisfy Delaunay criterion; middle - triangles which satisfy Delaunay criterion; right - Delaunay triangulation.

a prespecified boundary triangulation. This disadvantage gives rise to two grid generation approaches of constrained triangulation which preserve the boundary connectivity and take into account the Delaunay criterion. In the first approach of constrained Delaunay triangulations the Delaunay property is overriden at points close to the boundaries and consequently the previously generated boundary grid remains intact. Alternatively, or in combination with this technique, points can be added in the form of a skeleton to ensure that breakthroughs of the boundary do not occur. Another approach, which observes Delaunay criterion over the whole domain, is to postprocess the mesh by recovering the boundary simplexes which are missed during the generation of the Delaunay triangulation and by removing the simplexes lying outside the triangulated domain.

There are a number of algorithms to generate unstructured grids based on Delaunay criterion in constrained or unconstrained forms.

Some methods for Delaunay triangulations are formulated for a preassigned distribution of points which are specified by means of some appropriate technique, in particular, by a structured grid method. These points are connected to obtain a triangulation satisfying certain specific geometrical properties which, to some extent, are equivalent to the Delaunay criterion.

Many Delaunay triangulations use an incremental Bower-Watson algorithm which can be readily applied to any number of dimensions. It starts with an initial triangulation of just a few points. The algorithm proceeds at each step by adding points one at a time into current triangulation and locally reconstructing the triangulation.

Table 1.1. Comparative table of different realization of Delaunay triangulation construction algorithms

Algorithm	Complexity in worst case	Complexity in practice	Computational time for 10000 points	Realization efforts
Iterative algorithms				
Simple iterative algorithm	$O(N^2)$	$O(N^{3/2})$	$5,80$	*****
Iterative algorithm «Remove and build»	$O(N^2)$	$O(N^{3/2})$	$8,42$	**
Algorithm with indexing of searching by R-tree	$O(N^2)$	$O(N \log N)$	$9,23$	***
Algorithm with indexing of searching by k-D-tree	$O(N^2)$	$O(N \log N)$	$7,61$	***
Algorithm with indexing of searching by quad-tree	$O(N^2)$	$O(N \log N)$	$7,14$	***
Algorithm of static cashing	$O(N^2)$	$O(N^{9/8})$	$1,68$	*****
Algorithm of dynamic cashing	$O(N^2)$	$O(N)$	$1,49$	*****
Algorithm with stripe points dividing	$O(N^2)$	$O(N)$	$3,60$	*****
Algorithm with square points dividing	$O(N^2)$	$O(N)$	$2,61$	*****
Layer densening algorithm	$O(N^2)$	$O(N)$	$1,93$	****
Algorithm with points sorting along fractal curve	$O(N^2)$	$O(N)$	$5,01$	****
Algorithm with Z-code point sorting	$O(N^2)$	$O(N)$	$5,31$	*****
Merging algorithms				
Algorithm «Divide & Conquer»	$O(N \log N)$	$O(N \log N)$	$3,14$	***
Recursive algorithm with cutting along diameter	$O(N \log N)$	$O(N \log N)$	$4,57$	**
Algorithm of convex stripe merging	$O(N^2)$	$O(N)$	$2,79$	***
Algorithm of non-convex stripe merging	$O(N^2)$	$O(N)$	$2,54$	***
Direct construction algorithms				
Single-step algorithm	$O(N^2)$	$O(N^2)$	-	**
Single-step algorithm with binary searching tree	$O(N^2)$	$O(N \log N)$	-	**
Single-step cellular algorithm	$O(N^2)$	$O(N)$	-	**

Algorithm	Complexity in worst case	Complexity in practice	Computational time for 10000 points	Realization efforts
Double-line algorithms				
Double-line algorithm «Divide & Conquer»	$O(N \log N)$	$O(N \log N)$	$2,79$	****
Double-line algorithm with cutting along diameter	$O(N \log N)$	$O(N \log N)$	$4,13$	***
Double-line algorithm of convex stripe merging	$O(N^2)$	$O(N)$	2,56	****
Double-line algorithm of non-convex stripe merging	$O(N^2)$	$O(N)$	2,24	****
Modified hierarchical algorithm	$O(N^2)$	$O(N)$	15,42	*****
Algorithm of linear sweeping	$O(N^2)$	$O(N)$	4,36	*****
Radial algorithm	$O(N^2)$	$O(N)$	4,18	*****
Algorithm of recursive splitting	$O(N \log N)$	$O(N \log N)$	–	*
Strip algorithm	$O(N^2)$	$O(N)$	2,60	*****

The process allows one to provide both solution-adaptive refinement and mesh quality improvement in the framework of the Delaunay criterion. The distinctive characteristic of this method is that point positions and connections are computed simultaneously.

One more type of algorithm is based on a sequential correction of a given triangulation, converting it into a Delaunay triangulation.

The most recent review of different algorithm for construction of Delaunay triangulation was done by Skvortsov [11]. The Table 1.1 of considered by him algorithms is given below. For each algorithm complexity in practice and in worst case, computational time spent on 10000 points and authors estimation of realization efforts (more stars - easier implementation) are given.

In general, according to author's of paper experience, algorithm of dynamic cashing made a good showing. Approximately the same efficiency showed the algorithm of layer densening. It is important to mention that both of them are easy to program on any data structure. Among other good algorithms it is necessary to point out the algorithm of non-convex stripe merging and stripe algorithm but they are not trivial to program.

1.6. From Dirichlet to Delaunay

Dirichlet [12], at the end of the 19th century, has shown, that for a given set of points in two dimensions, it is possible to partition the plane into convex cells using a proximity criterion. At the turn of this century, Voronoi [13], focused on quadratic forms related to proximity problems. From this are derived a quite interesting application of the «parallélloédres primitifs», that is, the elementary polyhedra. As a consequence, Dirichlet's results can be extended in the three dimensional space. The concept of a Voronoi diagram, a set of cells corresponding to the proximity notion for a set of points, is then introduced.

In the 30's, Delaunay established that it is possible to deduce a triangulation from these diagrams by duality. His name being associated with this kind of triangulations, the Delaunay triangulation is introduced. Several issues related to this type of triangulation are then established including its uniqueness. This property is expressed using a key feature, referred as the empty sphere criterion.

In the 70's, i.e. approximately 40 years later, Lawson [15], noticed that, in two dimensions, the Delaunay triangulation and diagonal swapping are closely related. He shows that a local pattern of two non-Delaunay adjacent elements can be replaced by a Delaunay configuration by simply swapping the common edge. As a main cosequence of this result, a Delaunay triangulation can be derived from any arbitrary configuration using local modifications (edge swapping) only. Green and Sibson [14] pointed out that Delaunay triangulation enjoys a series of interesting properties. For instance, a «maximum» criterion is achieved. That is the Delaunay triangulation is the one, among all the possible triangulation, that maximizes the minimum of the angles formed by the edges between the elements.

One has still wait until the 80's to find some effective algorithms for constructing the Delaunay triangulations in terms of computational efficiency, particularly in dimensions higher than two; Hermeline [16] as well as Bower [17], Watson [18], Avis et al. [19] being the main references related to this aspect.

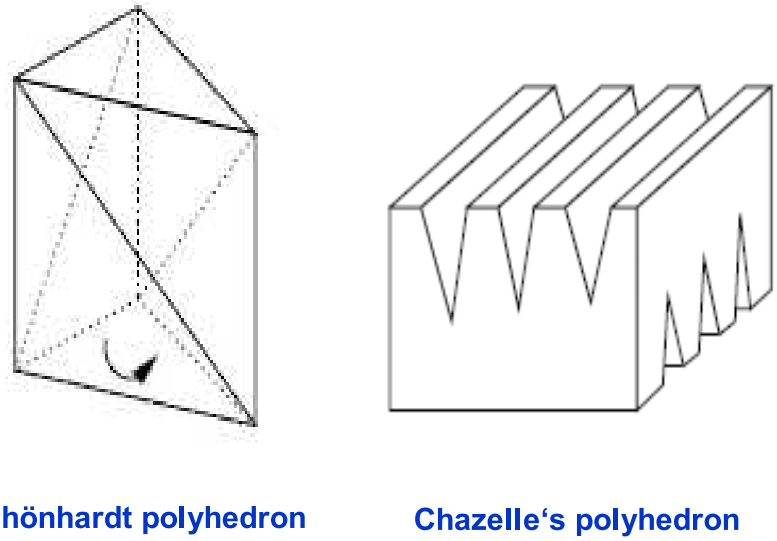


Figure 1.6. Two polyhedra which can not be tetrahedralized. Left - the Schönhardt's polyhedron, which is formed by rotating one end of a triangular prism; right - the Chazelle's polyhedron, which can be built out of a cube by cutting deep wedges.

1.7. Existence of 2D and 3D triangulations

While the two dimensional problem is successfully solved, the corresponding three-dimensional problem is still not fully treated. Numerous problems are still open and, specifically, the existence of a solution is not clearly established. Nevertheless it does not mean that more or less heuristic methods have not been developed, which aim at constructing a suitable solution. In practice, several authors have shown that it is possible to achieve a constrained triangulation in most of the cases.

Some polyhedra do not have triangulation at all [20–23]. In Fig. 1.6 two examples are given: the Schönhardt polyhedron and Chazelle's polyhedron. The Schönhardt polyhedron is a rather simple polyhedron for which it is not possible to find a triangulation (consisting of strictly positive volume polyhedra) without introducing internal point(s). Such points are called Steiner points.

In this example, this results from the fact that the triangulation without any internal points contains a tetrahedron whose volume is exactly zero. On the other hand, a point located anywhere in the interior of polyhedron results in a valid triangulation.

This example, while rather obvious, points out a fundamental difference between the two dimensional and the three-dimensional problem. In two dimensions, a solution without (Steiner) points always exist. The example also leads us to expect some difficulties in detecting this type of problem as well as in determining the adequate Steiner points.

Thus, an open problem is the determination of the minimum number of Steiner points required to make the constrained triangulation possible. Another open question is that of finding appropriate locations for these points, in a polynomial complexity. Ruppert and Seidel [24] showed that it is NP-complete to decide whether a simple polyhedron can be tetrahedralized or not. Nevertheless, any polyhedron is tetrahedralizable as long as additional points can be inserted. Constrained Delaunay triangulation algorithm insert additional vertices. A key question, when such additional points are necessary to ensure the existence, is to decide what is the optimal (minimal) number of additional points. Another concern, mainly for mesh refinement algorithms, is to avoid very short edges, which endanger very small tetrahedra, hence the number of additional points can be undesirably large. Various approaches [25–29] based on different point insertion schemes for constrained Delaunay triangulation have been proposed.

1.8. Overview of works regarding unstructured grid generation

The most recent and comprehensive review of structured and unstructured mesh generation techniques was done by Liseikin [30]. Unstructured grid methods were originally developed in solid mechanics. The paper by Field (1995) [6] reviews some early techniques for unstructured mesh generation that rely on solid modeling.

Though unstructured technology deals mainly with tetrahedral (triangular in two dimensions) elements, some approaches rely on hexahedrons (or quadrilaterals) for the decomposition of arbitrary domains.

Properties of n-dimensional triangulations were reviewed by Lawson (1986) [31]. The relations between the numbers of faces were proved in the monograph by Henle [32] and in the papers by Steinitz (1922) [33], Klee (1964)

[34], and Lee (1976) [35].

The Delaunay triangulation and Voronoi diagram were originally formulated in the papers of Delaunay (1934,1947) [36, 37] and Voronoi (1908) [13], respectively. Algorithms for computing of Voronoi diagrams have been developed by Green and Sibson (1978) [14], Brostow, Dussault, and Fox (1978) [38], Finey (1979) [39], Bower (1981) [17], Watson (1981) [18], Tanemura, Ogawa, and Ogita (1983) [40], Sloan and Houlsby (1984) [41], Fortune (1985) [42], and Zhou (1990) et al. [43]. Results of studies of geometrical aspects of Delaunay triangulation and their dual Voronoi diagrams were presented in the monographs by Edelsbrunner (1987) [44], Du and Hwang (1992) [45], Okabe, Boots, and Sugihara (1992) [46], and Preparata and Shamos (1985) [47]. Proofs of the properties of planar Delaunay triangulations were given by Guibas and Stolfi (1985) [48] and by Baker (1987,1989) [49, 50].

A technique for creating the Delaunay triangulation of an a priori given set of points was proposed by Tanemura, Ogawa, and Ogita (1983) [40]. The incremental two-dimensional triangulation which starts with an initial triangulation was developed by Bower (1981) [17] and Watson (1981) [18]. Watson has also shown the visibility of the edges of the cavity associated with the inserted point. Having demonstrated that the Delaunay criterion is equivalent to the equiangular property, Sibson (1978) [51] devised and later Lee and Schachter (1980) [52] investigated a diagonal-swapping algorithm for generating a Delaunay triangulation by using the equiangular property.

A novel approach, based on the aspect ratio and cell area of the current triangles, to the generation of points as the Delaunay triangulation proceeds was developed by Holmes and Snyder (1988) [53]. In their approach a new point is introduced in the existing triangulation at the Voronoi vertex corresponding to the worst triangle. Ruppert (1992) [54] and Chew (1993) [55] have shown that in the planar case the procedure leads to a Delaunay triangulation with a minimum angle bound of 30 degrees. An alternative procedure of inserting the new point on a Voronoi segment was proposed by Rebay (1993) [56]. A modification of the Rebay technique was made by Baker (1994) [57]. Haman, Chen, Hong (1994) [58] inserted point into a starting Delaunay grid in accordance with the boundary curvature and distance from the boundary, while Anderson (1994) [59] added nodes while taking into account cell aspect ratio

and proximity to boundary surfaces.

Approaches to the generation of boundary-conforming triangulations based upon the Delaunay criterion have been proposed by Lee (1976) [35], Le and Lin [60], Baker [50], Chew [61], Cline and Renka (1990) [62], George, Hecht, and Saltel (1990) [63], Weatherill (1990) [64], George and Hermeline (1992) [65], Field and Nehl (1992) [66], Hazlewood (1993) [67], and Weatherill and Hassan (1994) [68].

Further development of unstructured grid techniques based on the Delaunay criterion and aimed at the solution of three-dimensional problems has been performed by Cavendish, Field, and Frey (1985) [69], Shenton and Cendes (1985) [70], Perronet (1988) [71], Baker (1987,1989) [49, 50], Jamson, Baker, and Weatherill (1986) [72], Weatherill [73]. The application of the Delaunay triangulation for the purpose of surface interpolation was discussed by DeFloriani (1987) [74].

The octree approach originated from the pioneering work of Yerry and Shephard (1985) [75]. The octree data structure has been adapted by Lohner (1988) [77] to produce efficient search procedures for the generation of unstructured grids by the moving front technique. Octree-generated cells were used by Shephard et al. (1988) [78] and Yerry and Shephard (1990) [76] to cover the domain and the surrounding space and then to derive a tetrahedral grid by cutting the cubes. The generation of hexahedral unstructured grids was developed by Schneiders and Bunten (1995) [79].

The moving front techniques has been successfully developed in three dimensions by Peraire et al. (1987) [80], Lohner [81] and Formaggia (1991) [82]. Some methods using Delaunay connectivity in the frontal approach have been created by Merriam (1991) [83], Mavriplis (1991,1993) [84, 85], Rebay (1993) [56], Muller, Roe, and Deconinck (1993) [86], Marcum and Weatherill (1995) [87].

Advancing-front grids with layers of prismatic and tetrahedral cells were formulated by Lohner (1993) [88]. A more sophisticated procedure, basically using bands of prismatic cells and a spring analogy to stop the advancement of approaching layers, was described by Pirzadeh (1992) [89]. The application of adaptive prismatic meshes to the numerical solutions of viscous flow was demonstrated by Parthasarathy and Kallinder (1995) [90].

Some procedures for surface triangulations have been developed by Peraire (1988) [91], Lohner and Parikh (1988) [92], and Weatherill et al. (1993) [93].

A survey of adaptive mesh refinement techniques was published by Powell, Roe, and Quirk (1992) [94]. The combination of the Delaunay triangulation with adaptation was performed by Holmes and Lamson (1986) [95], Mavriplis (1990) [96] and Muller (1994) [97]. The implementation of solution adaptation into the advancing-front method with directional refinement and regeneration of the original mesh was studied by Peraire (1987) [80]. Approaches based on the use of sources to specify the local point spacing have been developed by Pirzadeh (1993,1994) [98, 99], and Weatherill et al. (1993) [93].

Chapter II

Parallelization and domain decomposition approaches

The *second* chapter is devoted to parallelization and domain decomposition approaches. Here the reasons for parallelization of mesh generation stage are explained. Advantages and disadvantages of a priori and a posteriori decomposition methods are described. Various domain decomposition approaches such as prepartition along the same direction, recursive prepartitioning, overdecomposition are shown. Examples of different criteria for setting the cutting planes up are given: equal volume of subdomains, equal moment of inertia, equal number of surface elements and so on. Then overview of works related to parallel mesh generation and some conclusions are given.

2.1. Need for parallelization of grid generation procedure

The widespread availability of parallel machines with large memory, solvers that can harness the power of these machines, and desire to model in ever increasing detail geometrical and physical features has lead to a steady increase in the number of points used in PDE solvers.

While many solvers have been ported to parallel machines, grid generators have left behind.

Grids in excess of 10^7 elements have become common for production runs in Computational Fluid Dynamics (CFD) [102–106], Computational Electromagnetics (CEM) [100, 101] as well as in Computational Mechanics [141]. In high frequency CEM, typical simulations employ meshes of five million triangles in two dimensions and twenty million tetrahedra in three dimensions. In CFD, mesh of at least ten million tetrahedra can be required for a high Reynolds number viscous turbulent flow simulation over a complete air-

craft.The expectations is that in the near future grids in excess of $10^8 - 10^9$ elements will be required [107].

For applications where remeshing is an integral part of simulations, e.g. problems with moving bodies [108–114] or changing topologies [115, 116], the time required for mesh regeneration can easily consume more than 50% of the total time required to solve the problem [107].

As mesh size becomes as large as this, the process of mesh generation on a serial computer becomes problematic and sometimes impossible both in terms of time and memory requirements. As problem sizes grow, mesh generation on a single processor becomes a computational bottleneck. Parallel computers afford the potential to remove the bottleneck. Since early 1990s attempts have been made to parallelize this stage.

Therefore, the need for developing parallel mesh generation technique is well justified and caused by the following major factors:

- **Lack of memory.** Computational meshes exceeding $2,5 \cdot 10^7$ tetrahedra can not be generated on a single CPU because of memory limitations. Many scientific applications suffer from a lack of memory size. The performance that is achieved by modern processors is often wasted by starvations due to memory bandwidth.

- **Long computational time.** Pre-simulation process of mesh generation can consume most of the time required to solve the problem. Especially for applications where remeshing is an integral part of simulations grid generation procedure is the main bottleneck.

2.2. Parallel computing

Parallel computing is a solution to handle large size problems (i.e., with a large number of unknowns). When coupled with a domains decomposition solution method, this approach needs to construct the mesh of several sub-domains whose union covers the entire initial domain. These sub-meshes must enjoy a series of properties and must make communication possible between sub-meshes. From the algorithmical point of view, regarding the meshing point of view, a parallel computation relies on a partitioning of the domain consisting of several meshes (so as to furthermore distribute the computation on the various

processors, where on processor is in charge of one sub-mesh). Constructing this partitioning, as well as constructing each of the sub-meshes, can be achieved using many different approaches. Actually, several kinds of approaches can be found that will be considered. In general they are classified into an a priori and a posteriori methods.

2.3. A posteriori partitioning

Provided a (fine) mesh of the domain under consideration, an a posteriori partitioning method decomposes this mesh into sub-meshes so as to extract the sub-domains. Numerous methods exists to address this problem. For more details, we refer, for instance, Simon [117] or Farhat and Lesonne [118].

The main drawback of this approach is that the necessary memory requirement to complete the partitioning is approximately the sum of the size needed to store the initial mesh and the size of at least one of sub-meshes. Furthermore, various classical difficulties related to the partitioning methods are observed (related to the sub-meshes, load balancing, the interface smoothness, the interface sizes, etc.).

It is necessary to mention that the a posteriori approach is most likely the worst way to perform the parallelism at the mesh level. Nevertheless, this method leads to good results for reasonably sized meshes. Conversely, if the size of the problem is very large (for instance, of the order of ten million elements), the creation of the initial mesh may not even be possible. For instance, it is possible to achieve a mesh with about 20 million elements but, while it is possible to create a mesh a little bit larger, this mesh usually can not be generated for memory size reason. Hence, ***the a priori approach makes sense, even though it requires more efforts in order to make it reasonably efficient***.

2.4. A priori partitioning

Following this approach, we will attempt to avoid the difficulties and weakness related to the a posteriori method (large memory space requirements, no parallelism at the mesh generation step, etc.) by first constructing a partitioning. This step may start either from a coarse mesh (i.e., with a small number of

elements) and then by constructing the mesh of the different members of partitions concurrently, thus taking advantage of the parallelism from the mesh generation step. Another approach starts only from the surface mesh of the domain.

The major difficulty, expected with this method, is to find an appropriate method ensuring a good load balancing between the sub-domains.

Indeed, the load balancing can be addressed by using the information provided by either a coarse mesh or the surface mesh. For the first approach, we can consider a mesh with relatively small number of internal points as the coarse mesh. For the second approach, the coarse mesh will be constructed using the surface mesh as sole input data.

Similarly, the sub-domain interface, irrespective of the method, must be constructed using either the coarse mesh or the surface mesh.

2.4.1. Types and criteria of domain decomposition

There are several major types of domain decomposition approaches which use different criteria for splitting and defining the cutting planes.

- **Prepartitions along the same direction.** The object is partitioned along several partitioning planes which are parallel one to another. A partitioning of an object into N subdomains would require $N-1$ parallel tasks. The partitioning can be done in parallel.

- **Recursive prepartitioning.** An object is cut into two. Then a new cutting plane is determined for each part and the parts are cut into two and so on. Note, that first task is sequential, second task involves two parallel tasks, and the $k-th$ step involves 2^{k-1} tasks.

- **Overdecomposition.** The number of subdomains is much larger than the number of processors. Then, in the case of load imbalance, the master process gives the task (subdomain) to idle processor.

These techniques are illustrated in Fig. 2.1 [119].
Several criteria can be used to control load balancing:

- volume of the subdomain enclosed by a triangulation;

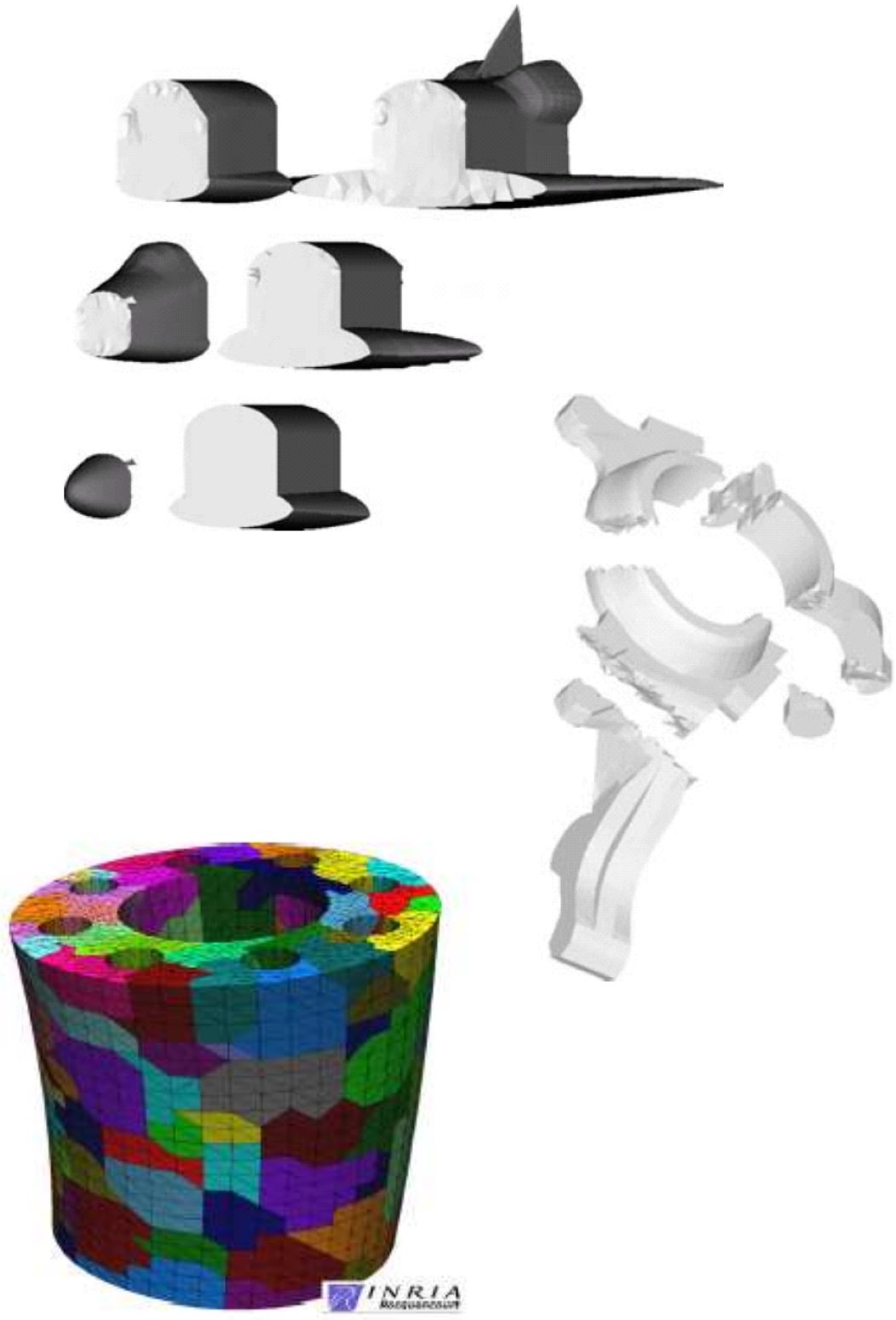


Figure 2.1. Decomposition methods: prepartitioning along the same direction of a shuttle «Columbia», recursive prepartitioning of a mechanical object, overdecomposition of a pipe with holes.

- number of surface nodes in the subdomain;
- number of surface faces in the subdomain;
- moment of inertia of the subdomain.

2.5. Overview of works regarding parallel mesh generation

Parallel mesh generation is a relatively new research area between the boundaries of two scientific computing disciplines: computational geometry and parallel computing.

Starting in two dimensions, Verhoeven et al. [120] demonstrated the ability to produce parallel unstructured Delaunay meshes across a network of workstations. Topping et al. [121], Lämer et al. [122], Löhner et al. [123], amongst others, have parallelized the advancing front algorithm. Moving to three dimensions, the task becomes more complicated. Chew et al. [124], Chrisochoides et al. [125], Okunsanya et al. [126] have parallelized the Delaunay algorithm. Löhner [127] has demonstrated the extension of the advancing front algorithm to produce tetrahedral elements on parallel platforms. Said et al. [128] have shown a parallel mesh generation using initial coarse meshing and decomposition.

The exhaustive survey of parallel mesh generation methods has been given by Chrisochoides [129]. The parallel mesh generation methods are classified in terms of two basic attributes: the sequential technique used for meshing the individual subproblems and the degree of coupling between the subproblems. These methods are based on three widely used techniques: Delaunay, Advancing Front, and Edge Subdivision.

It takes about ten to fifteen years to develop the algorithmic and software infrastructure for sequential industrial strength mesh generation libraries. Moreover, improvements in terms of quality, speed, and functionality are open ended and permanent which makes the task of delivering state-of-the-art parallel mesh generation codes even more difficult.

In area with immediate high benefits to parallel mesh generation is domain decomposition. The domain decomposition problem is still open for 3D geometries and its solution will help to deliver stable and scalable methods

that rely on off-the-shelf mesh generation codes for Delaunay and Advancing Front Techniques. The edge subdivision methods are independent of the domain decomposition.

A longer term goal is the development of both theoretical and software frameworks to implement new mesh generation methods which can: take advantage of multicore architectures with more than two hardware contexts for the next generation of high-end workstations and scale without any substantial implementation costs for clusters of high-end workstations.

Finally, a long term investment to parallel mesh generation is to attract the attention of mathematicians with open problems in mesh generation and broader impact in mathematics.

Chapter III

Decomposition and parallel mesh generation algorithm

The *third* chapter gives an extensive description of developed parallel mesh generation algorithm. At first, problem and goals of the work are formulated. Then overview of algorithm steps are given with following detailed explanation in corresponding sections: setting up the cutting planes and load balancing, forming of splitting contour, construction of interface and 2D constrained Delaunay triangulation, splitting along the path of edges, overall domain decomposition and parallel mesh construction. Since at each stage many different solution techniques may be applied, the reasons for choosing one, advantages and disadvantages are discussed.

3.1. Problem formulation and goals of the work

The goal of the work is to create a parallel grid generator for high-quality tetrahedral grids with good properties (e.g. Delaunay property) for solving PDEs and suitable for FEM calculations. It should be fully automatic, adaptive (via coupling with the solver) and, of course, be able to generate large meshes. The input data is a CAD surface description of an object. The algorithm should preserve original surface mesh and use boundary description only, since creation of initial coarse mesh may not be possible for very complex objects such as fibrous microstructures or porous media (Fig 3.1) which we intend to work with in the future.

The major goal of the algorithm is not primarily to reduce computational time. It is to overcome the memory restrictions required to generate very large meshes. Simulations involving very large meshes, of the order 10^7 nodes, are typically performed on large parallel computer platforms.

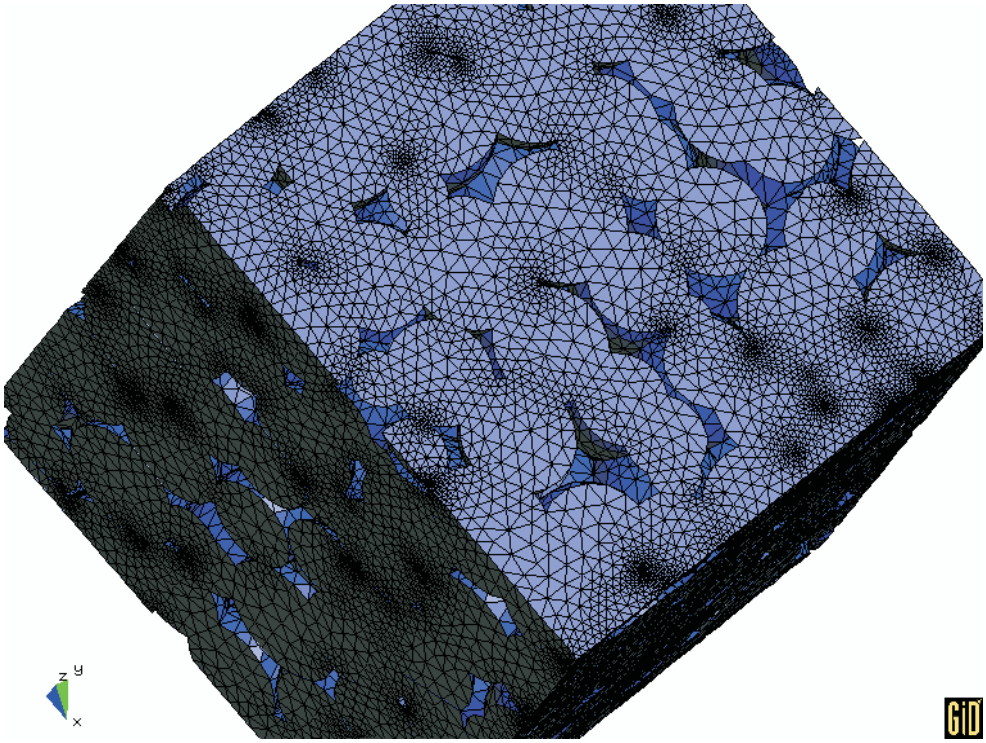


Figure 3.1. Computer generated stochastic representative volume element (RVE) of sinter material for the computation of effective elasticity coefficients.

In Fig. 3.2 the main steps of implementation are shown.

3.2. Algorithm steps description

The approach presented here based upon geometrical decomposition of computational domain. The algorithm consists of the following major steps:

1. **Decomposition of an object into open non-overlapping subdomains.**

   (a) computing the center of mass and inertia tensor computation to determine the cutting planes

Figure 3.2. Scheme of major implementation steps of the parallel grid generator.

(b) extracting of all intersecting edges and construction a cross-section contour line.

2. **Construction of closed and compatible surface mesh for each subdomain.** This step includes: projection of contour nodes on a cutting plane, construction of 2D constrained Delaunay triangulation on the plane inside the contour and mapping back the contour nodes of the triangulation to original surface positions.

3. **Independent parallel volume meshing (without communication) within each subdomain based on and compatible with its surface mesh description.**

In Fig. 3.3 on a simple geometry, cylinder, major steps of the algorithm are illustrated. For simplicity we will stick to this example. In the Chapter 5 devoted to results more complex geometries with greater number of subdomains will be shown.

The main goal of the algorithm is to perform simultaneous construction of three dimensional computational mesh inside each subdomain. This is computationally the most expensive process.

Decomposition criterion is based upon moment of inertia and principal inertia axis to achieve better load balancing and to minimize area of interfaces. So the algorithm is sensitive to both the object shape and the grid resolution.

The advantage of this algorithm is that it allows us to use well tested and fine-tuned sequential 2D and 3D triangulators, which are capable of producing high-quality Delaunay meshes with different conditions and constraints. They are widely available (Triangle, mesh2d, NetGen, TetGen, GRUMMP and other). So the parallel grid generator can be easily implemented or modified.

Developed algorithm is clearly advantageous in terms of computational time and memory use compare to an a posteriori partitioning method used by mesh partitioning libraries such as (PAR)METIS.

It achieves 100 % code re-use and eliminates communication and synchronization since volume meshing is performed independently.

Another important property is that the algorithm preserves original surface mesh and produces almost plane interfaces, i.e. interfaces with small perturbations, between the subdomains.

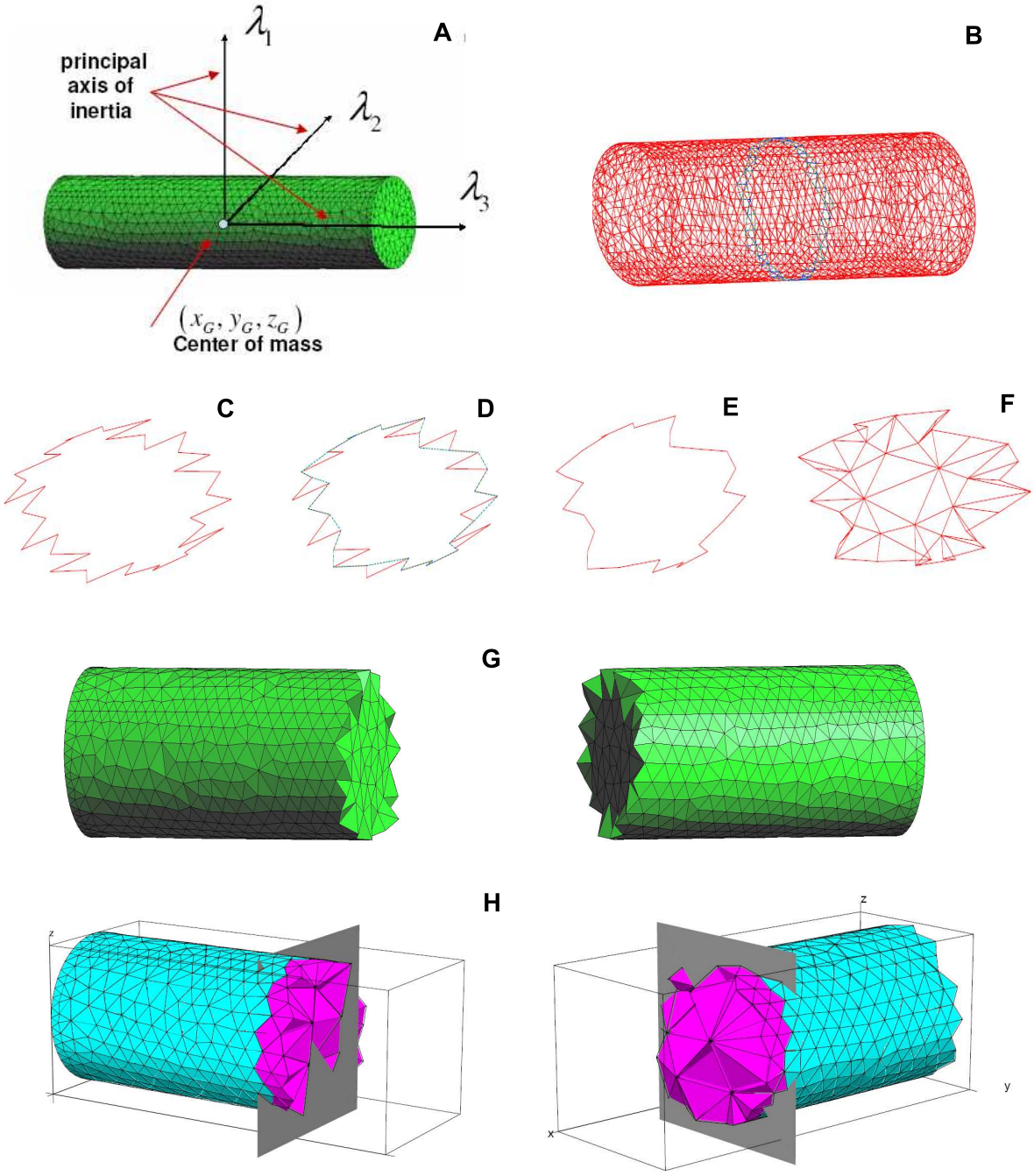


Figure 3.3. Major steps of the algorithm. A – center of mass and inertia matrix calculation. B – setting up the cutting plane. C – closed loop of intersected edges. D – smoothing the contour. E – smoothed contour of edges. F – interface triangulation and mapping the contour nodes back. G – construction of closed and compatible surface mesh. H – parallel independent construction of volume mesh inside (shown in pink).

In spite of all these advantages the method consists of a sequence of steps, where almost all of them are simple, as opposite of sophisticated algorithms, which are used in other mesh generating methods.

The main disadvantage of this approach is the fact that the associated «optimal» partitioning problem is NP-complete for general regions. So it can be used for a certain class of geometric domains only.

The flow chart of parallel grid generator is shown in Fig. 3.4. Details of the algorithm are given in the following subsections.

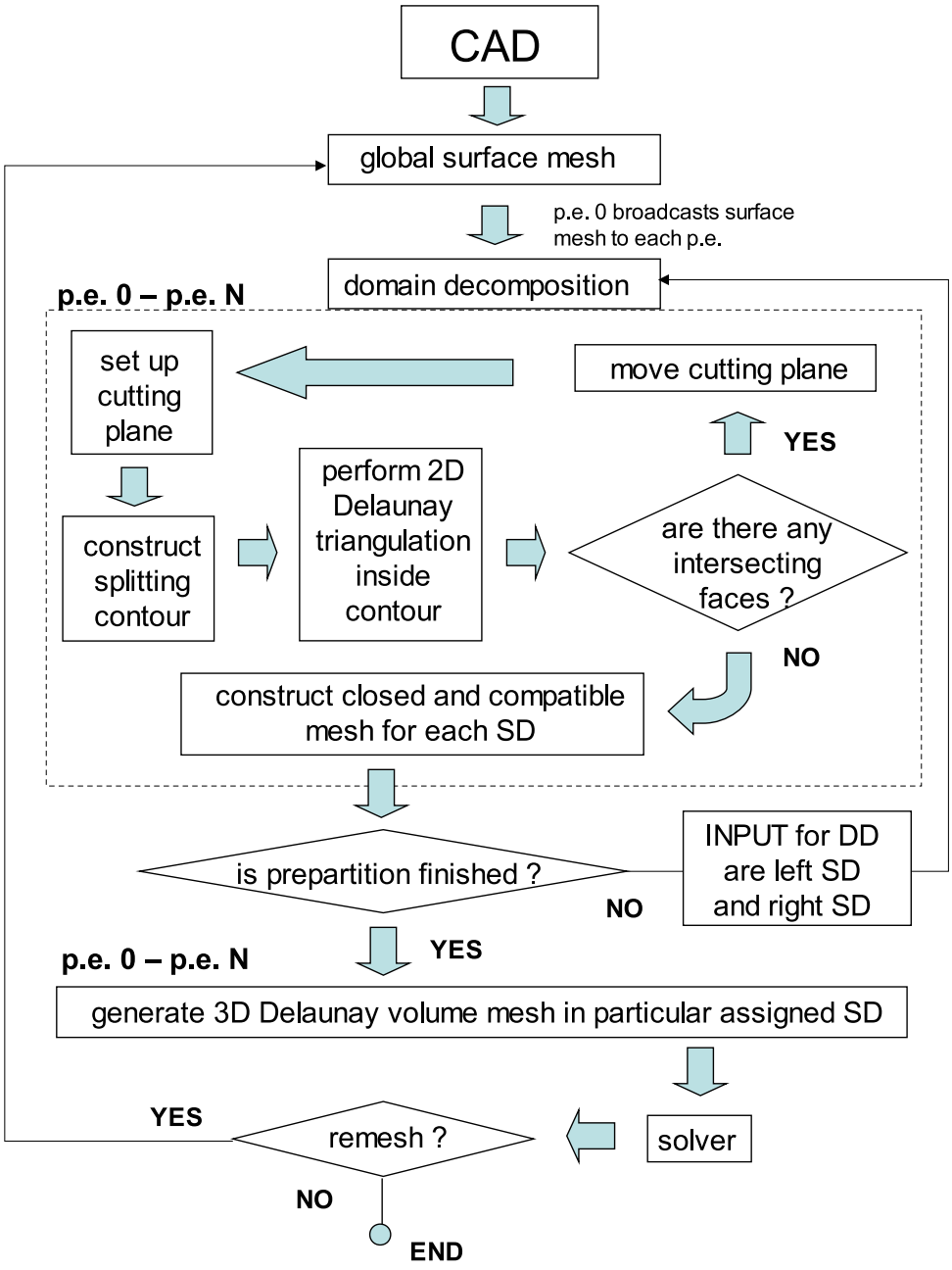


Figure 3.4. Flow chart of parallel grid generator.

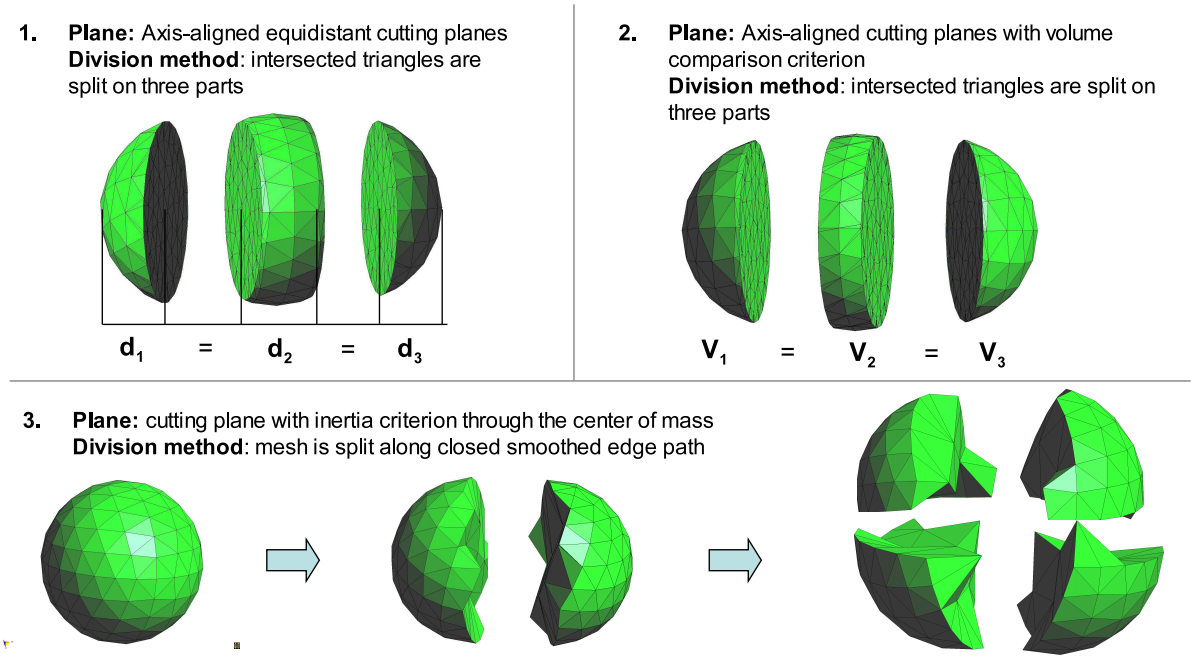


Figure 3.5. Evolution of the parallel grid generator: 1 – equidistant axis-aligned cutting; 2 – axis-aligned cutting with volume comparison; 3 – recursive cutting with moment of inertia equality.

3.3. Setting up the cutting planes and load balancing

3.3.1. Evolution of the load balanced splitting algorithm

Going through developing stages, the parallel grid generator had different partitioning techniques and splitting criteria (see Fig. 3.5).

In the first versions of parallel grid generator axis-aligned equidistant planes were used. It could result in unbalanced partitioning depending on the shape of the object because the scheme it is not sensitive to the object shape and limited by the same angle of the planes. So it can not give a good result for any arbitrary shape due to irrational and unbalanced cutting.

Next improvement of the algorithm was using of the axis-aligned cutting with volume comparison criterion which produces balanced partitioning, but it also could result in a large interface area, which is not optimal for further interface triangulation and crucial for a parallel solver, because of communication overhead between subdomains.

Currently, recursive method of inertia bisection is used, which is based on surface triangulation. This method minimizes area of interfaces and is sensitive

to both the object shape and the grid resolution.

Much attention has been paid to the construction of the splitting contour, which is a key stage in procedure of qualitative decomposition of computational domain. More detailed description of inertial bisection method used and of splitting contour construction technique will be given in sections 3.3.4 and 3.4 correspondingly.

3.3.2. Orientation of boundary faces

Due to different operations performed on mesh, such as cutting of triangles, insertion of new nodes, adding of interface mesh to the original one, boundary elements could be not properly oriented. The Delaunay grid generator we are going to use is sensitive to the orientation of the boundary faces, in order to determine domain to be meshed. Also for different estimations like the volume enclosed by surface triangulation of the subdomain the boundary faces of all subdomains have to be correctly oriented.

A technique based on a very simple concept can be applied. Two adjacent triangles are considered to have the same orientation if the shared edge exists in order ij on the first triangle, and the same edge ordered ji on the other triangle. In Fig. 3.6 (A) two correctly oriented triangles are shown.

The original surface faces which remain intact have correct orientation. Thus, algorithm starts from one of this faces and changes vertices order in triangles oriented differently.

3.3.3. Comparison of volumes enclosed by surface triangulation

Gaussian integral theorem relates the divergence of a vector field within a volume to the flux of a vector field through a closed surface by the following:

$$\iiint_V \nabla F \cdot dv = \iint_S F \cdot da, \tag{3.1}$$

where surface S encloses the volume V. The ∇ vector is defined by

$$\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right). \tag{3.2}$$

When F is set to

$$F(x, y, z) = (x, 0, 0), \tag{3.3}$$

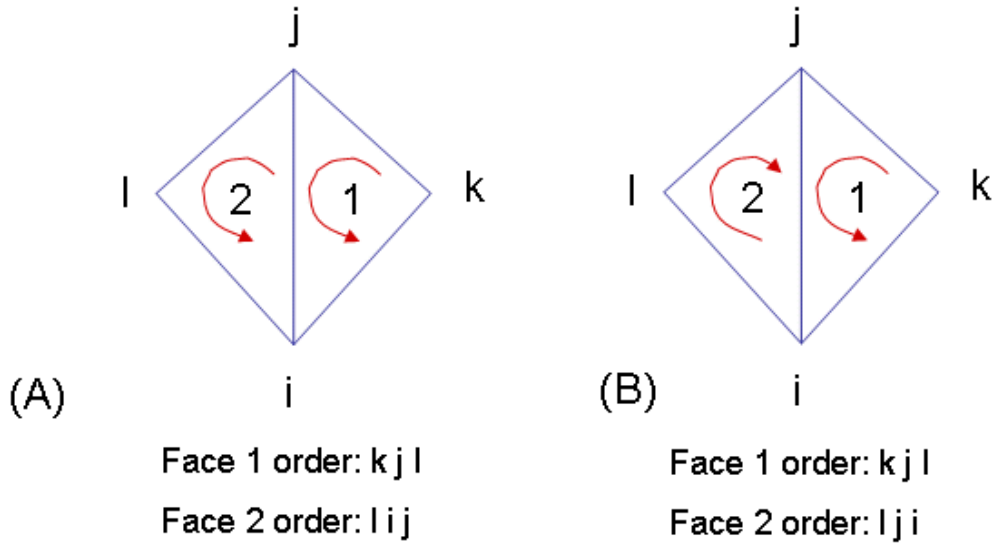


Figure 3.6. Two adjacent faces oriented correctly in (A). Notice the order of the shared edge in every face.

$\nabla \cdot F = 1$ and equation (3.1) becomes volume equation

$$\iiint_V 1dv = \iint_S F \cdot da \tag{3.4}$$

Equation (3.4) tells us that the flux of the vector field F through the closed surface S is equal to the volume enclosed by S. When S is defined by a set of triangles the enclosed volume is equal to the sum of the flux of F through every triangle. A straightforward parameterization of the triangle $\{v^{(1)}, v^{(2)}, v^{(3)}\}$ is given by:

$$e^{(1)} = v^{(2)} - v^{(1)}; \quad e^{(2)} = v^{(3)} - v^{(1)}; \quad s(u,v) = v^{(1)} + ue^{(1)} + ve^{(2)}, \tag{3.5}$$

where the evaluation of $s(u,v)$ yields any point on the triangle for $u \in [0,1]$ and $v \in [0, 1-u]$. Under this parameterization da from equation (3.4) can be written as:

$$da = (e^{(1)} \times e^{(2)})\, dvdu \tag{3.6}$$

The flux of F through the triangle $\{v^{(1)}, v^{(2)}, v^{(3)}\}$ now follows:

$$\Phi = \int_0^1 \int_0^1 F\left(s(u,v)\right) \cdot (e^{(1)} \times e^{(2)})\, dvdu \tag{3.7}$$

$$\Phi = \int_0^1 \int_0^1 \left(v_x^{(1)} + u e_x^{(1)} + v e_x^{(2)}\right) \left(e_y^{(1)} e_z^{(2)} - e_z^{(1)} e_y^{(2)}\right) \, dvdu \tag{3.8}$$

The analytical solution to equation (3.8) can be written as:

$$\begin{aligned} \Phi \;=\; & \frac{1}{6}\left[\left(v_y^{(2)} - v_y^{(1)}\right)\left(v_z^{(3)} - v_z^{(1)}\right) - \left(v_z^{(2)} - v_z^{(1)}\right)\left(v_y^{(3)} - v_y^{(1)}\right)\right] \times \\ & \times \left(v_x^{(1)} + v_x^{(2)} + v_x^{(3)}\right) \end{aligned} \tag{3.9}$$

The right hand side of equation (3.9) yields the flux of F through the triangle $\{v^{(1)}, v^{(2)}, v^{(3)}\}$. Given a closed and clockwise wound triangle set $\{v^{(i_1)}, v^{(i_2)}, v^{(i_3)}\}$ for $i = \{0, 1, 2, ..., n\}$ the enclosed volume is computed by evaluating the following sum:

$$\begin{aligned} \mathcal{V} \;=\; & \frac{1}{6}\sum_i \left[\left(v_y^{(i_2)} - v_y^{(i_1)}\right)\left(v_z^{(i_3)} - v_z^{(i_1)}\right) - \left(v_z^{(i_2)} - v_z^{(i_1)}\right)\left(v_y^{(i_3)} - v_y^{(i_1)}\right)\right] \times \\ & \times \left(v_x^{(i_1)} + v_x^{(i_2)} + v_x^{(i_3)}\right) \end{aligned} \tag{3.10}$$

3.3.4. Inertia bisection method

At the moment the center of gravity along with moment of inertia criterion is used. Each object is cut perpendicular to its smallest principal inertia axis. It means that for each part with the set of nodes V, the inertia matrix I is computed by:

$$I = \begin{bmatrix} \sum_{v\in V}(v_y - c_y)^2 + (v_z - c_z)^2 & -\sum_{v\in V}(v_x - c_x)(v_y - c_y) & -\sum_{v\in V}(v_x - c_x)(v_z - c_z \\ -\sum_{v\in V}(v_y - c_y)(v_x - c_x) & \sum_{v\in V}(v_z - c_z)^2 + (v_x - c_x)^2 & -\sum_{v\in V}(v_y - c_y)(v_z - c_z) \\ -\sum_{v\in V}(v_z - c_z)(v_x - c_x) & -\sum_{v\in V}(v_z - c_z)(v_y - c_y) & \sum_{v\in V}(v_x - c_x)^2 + (v_y - c_y)^2 \end{bmatrix},$$

where $(x_c, y_c, z_c) = (c_x, c_y, c_z)$ is the coordinate of the center of gravity C which is calculated by assigning unit mass to each node of the mesh. Thus grid resolution is also taken into account. Then (one of) the eigenvector(s) with the smallest eigenvalue is selected (see Fig. 3.7). This procedure defines planes perpendicular to the smallest principal inertia axis. The actual cutting plane is chosen to go through the center of gravity. This partitioning technique is

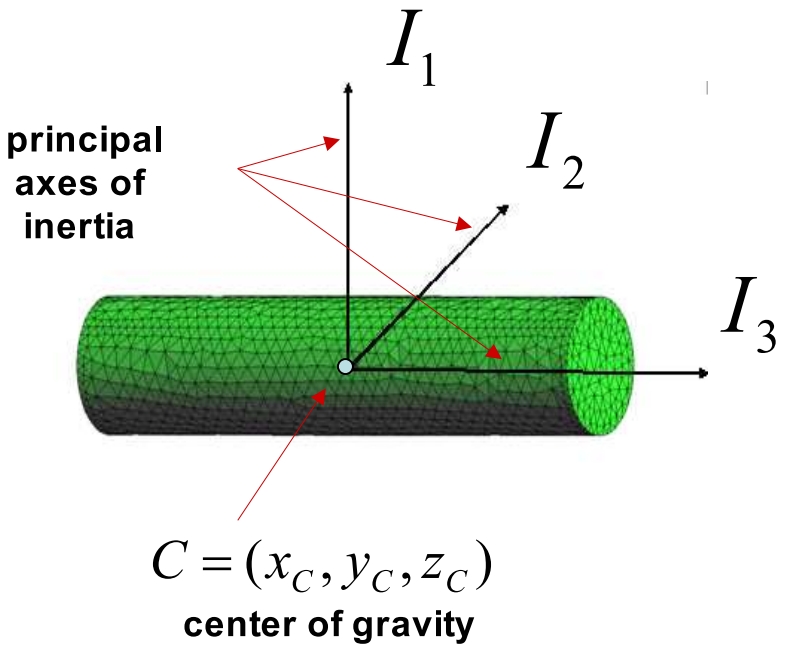


Figure 3.7. Principal axis of inertia corresponding to eigenvalues of inertia tensor. Center of gravity of the object defined by surface triangulation.

sensitive to the object shape and grid resolution and can minimize the interface area.

Nevertheless, it turns out to be hard to find a reasonable criterion for predicting a good load balancing in advance. Even if the number of tetrahedra is approximately the same for each subdomain, the CPU time spent for the volume meshing of each part can be quite different [119].

3.4. Forming the splitting contour

The stage of forming the splitting contour is a key step in the whole process of getting volume mesh. This is due to the fact that possibility of splitting and quality of decomposition directly depend on quality of the splitting contour. It influences adjacent surface elements and, therefore, volume elements too.

Going through different developing stages the parallel grid generator had various techniques for constructing the splitting contour. First is based on breaking of intersected triangles up into three smaller triangles and insertion of new nodes at the points of intersection. It results in triangles of bad quality and with small area. In order to remove this kind of triangles special mesh

optimization technique had been used. Second approach is based on idea of building the splitting contour out of smoothed chain of surface edges, keeping original surface mesh preserved.

3.4.1. Straight contour by breaking up intersected triangles

In this approach the intersections of the cutting planes with the surface triangles are calculated. Then the intersection points are sorted in a such way, that they form a closed and continuous cross-section contour. The intersected triangles are split into three other smaller triangles (Fig. 3.8). One could

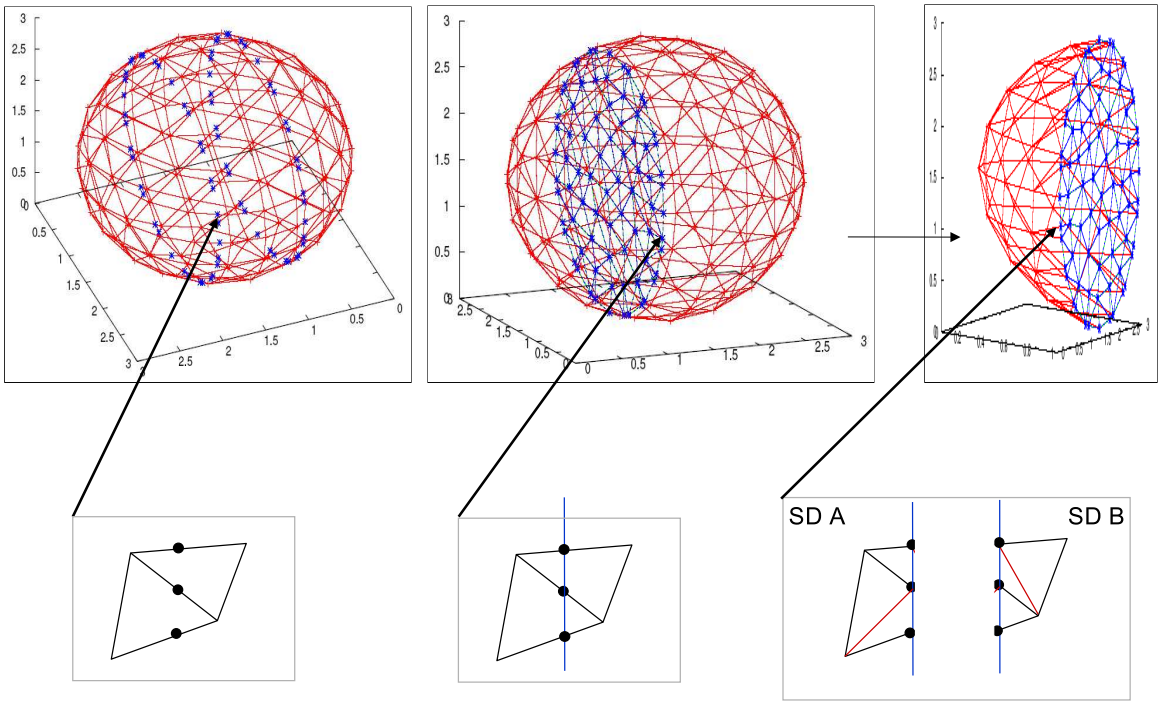


Figure 3.8. Construction of splitting contour by breaking up intersecting triangles.

notice that in case of arbitrary plane location this method could result in bad quality triangles, i.e. triangles with very acute angles or with very small area. This kind of mesh usually is not suitable for further volume mesh construction. That is why certain optimization technique had been used for improving of surface mesh quality.

3.4.2. Surface mesh optimization

The cutting of a surface mesh, as it could be noticed in Fig. 3.8 results in bad triangles with very acute angles, which are highly undesirable for further volume mesh construction and for finite element methods. One of the ways to get rid of them is to apply some mesh optimization technique. Method described in [130] has been used for improving of quality characteristics of surface mesh. Given a set of data points scattered in three dimensions and an

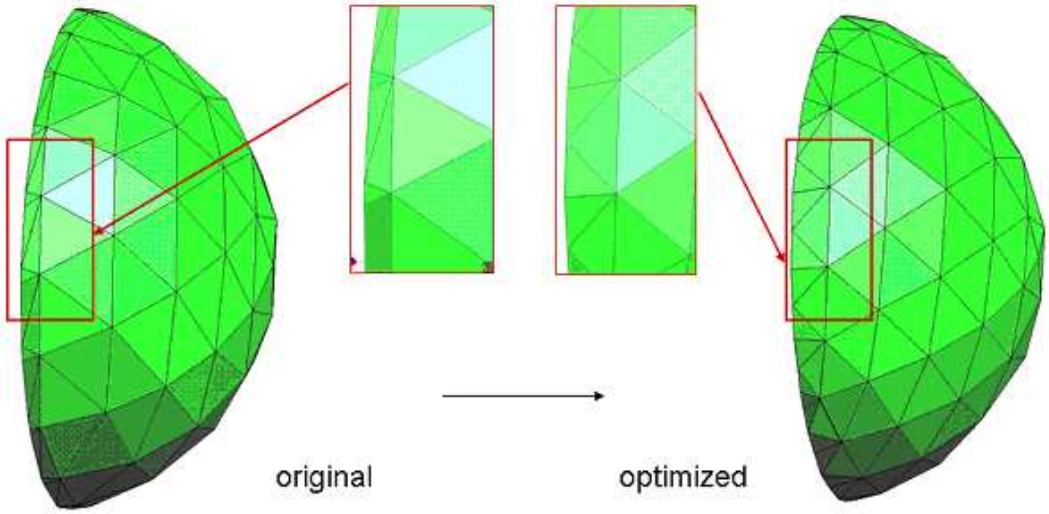


Figure 3.9. Surface mesh optimization.

initial triangular mesh $\mathcal{M}_0$, the task is to produce a mesh $\mathcal{M}$, of the same topological type as $\mathcal{M}_0$, that fits the data well and has a small number of vertices.

Here it is important to distinguish connectivity of the vertices and their geometric positions. Formally a mesh $\mathcal{M}$ is a pair (K, V), where K is a simplicial complex representing the connectivity of the vertices, edges and faces, thus determining the topological type of the mesh and $V = \{v^{(1)}, \ldots, v^{(m)}\}$, $v^{(i)} \in \mathbb{R}^3$ is a set of vertex positions defining the shape of the mesh in $\mathbb{R}^3$ (its geometric realization).

We find a simplicial complex K and a set of vertex positions V defining a mesh $\mathcal{M} = (K, V)$ that minimizes the energy function:

$$E(K, V) = E_{dist}(K, V) + E_{rep}(K) + E_{spring}(K, V).$$

The distance energy $E_{dist}(K, V)$ is equal to the sum of squared distances from a given set of points to the mesh. It measures the closeness of fit. The

representation energy $E_{rep}(K)$ is proportional to the number of vertices m of K. It penalizes meshes with a large number of vertices. The spring energy $E_{spring}(K, V)$ places on each edge a spring of rest length 0 and spring constant k. It guarantees existence of a minimum. More detailed information can be found in [130].

This optimization method is used in computer graphics applications. In our specific case, we are optimizing the mesh for further FEM calculations. So in present work we are not changing the topology of the mesh and the number of its nodes, but the mesh nodes are moved. As it can be seen in Fig. 3.9 this optimization technique improves the quality of the mesh a lot.

3.4.3. Improved construction of contour out of surface edges

Once the cutting plane is defined, we can construct a cross-section contour where 2D constrained Delaunay triangulation will be performed.

Here we present an improved technique of forming the splitting path of edges. The construction of the contour consists of the following steps:

1. **Extract all intersected edges of the surface triangulation.**
2. **Remove all edges with "hanging"nodes.**
3. **Sort the edges into a closed loop.**
4. **Remove multiple paths if any exists.**
5. **Smooth the contour.**

Procedure of forming the contour is shown in Fig. 3.10. For more complex geometry configurations, especially involving corners and concavities, multiple paths can occur. Several examples of such incidents are illustrated in Fig.3.11. Special care is taken in order to remove all of these incidents. The smoothing step requires more detailed explanation. We consider all triangles attached to the path of edges and for those triangles, which have two edges in the path, we replace them with third edge. The smoothing phase is required for better projection on the cutting plane and construction of 2D triangulation of interface. Next section is devoted to that problem.

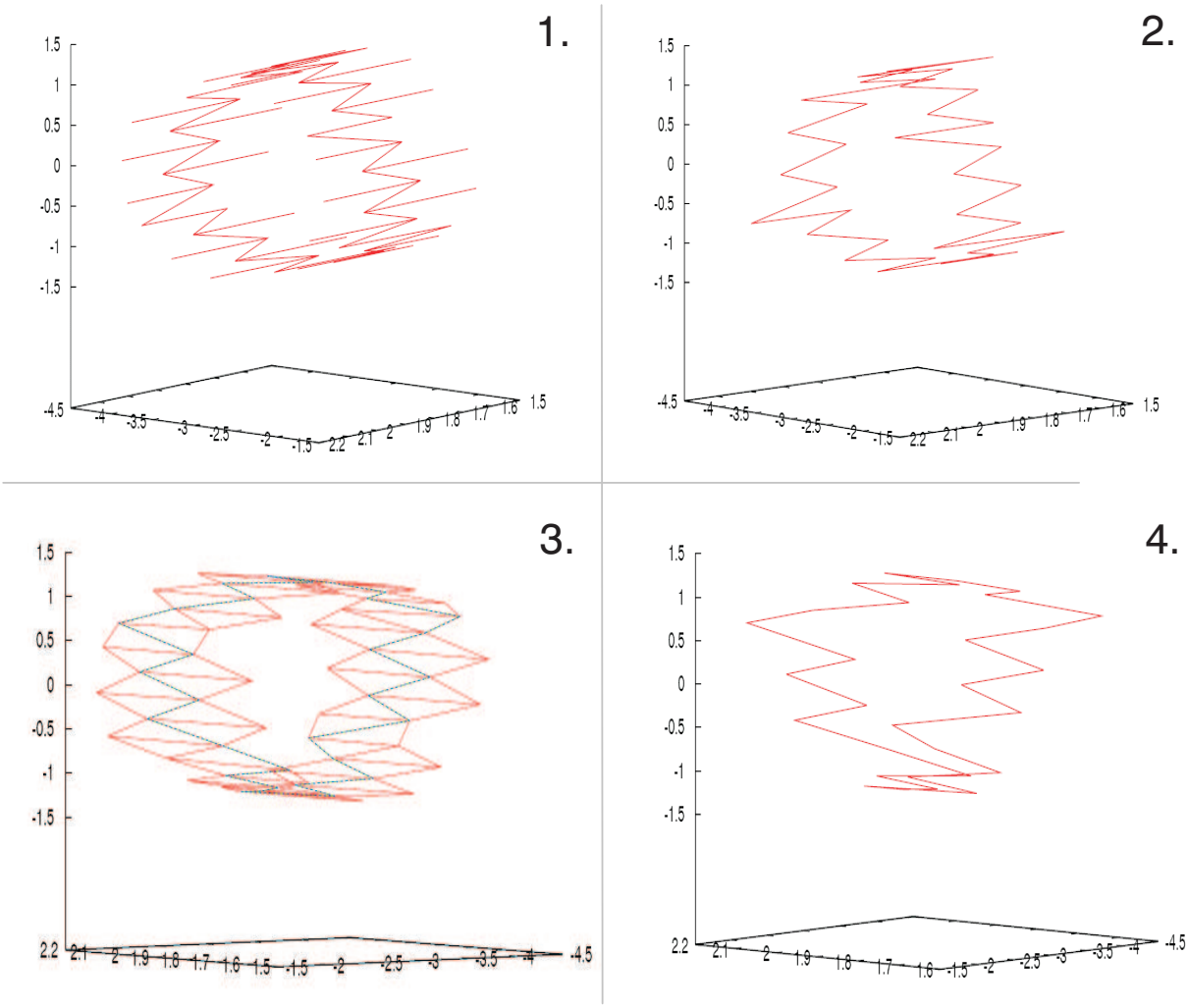


Figure 3.10. Steps of the contour construction. a – extract all intersected edges; b – remove edges with "hanging" nodes; c – replace two edges in the loop with third one for triangles which have three nodes in the path; d – smoothed contour of edges.

3.5. Construction of interface and 2D constrained Delaunay triangulation

Two steps are required before the 2D triangulation of the interface can be done:

1. Projection of the contour nodes on the cutting plane.

2. Rotation into x-y plane for 2D triangulation.

The program *Triangle* [132] is used for triangulation of the interface.

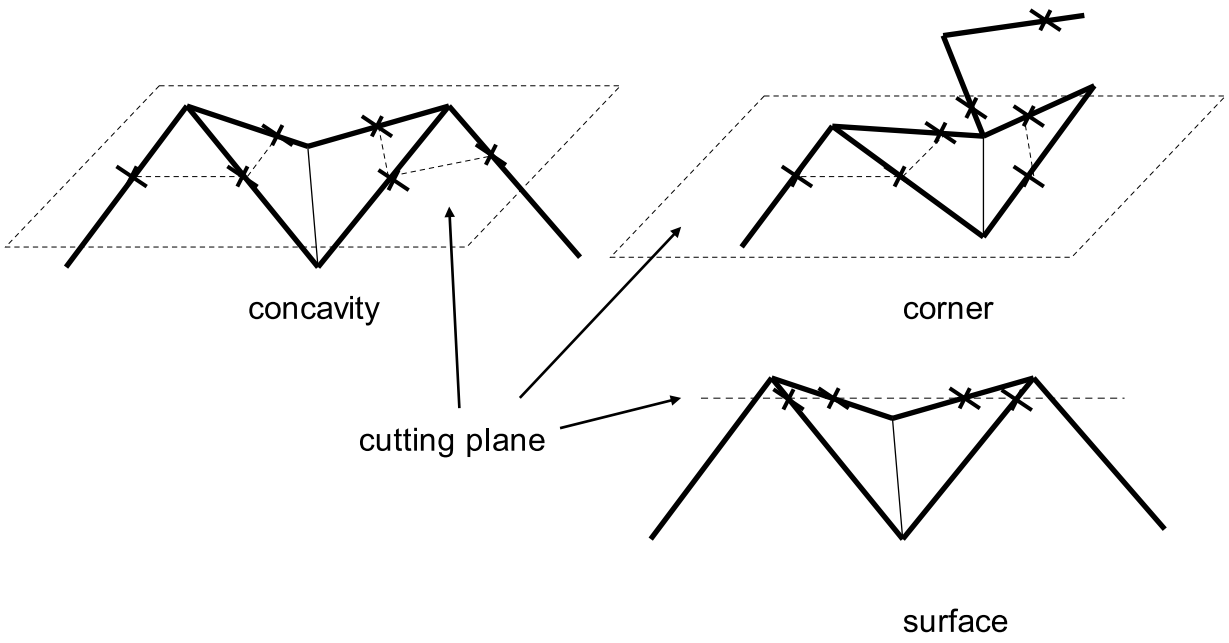


Figure 3.11. Incidents of multiple paths in case of concavity and corner.

In three dimensions a coordinate rotation can be described by a $3\text{x}3$ matrix T, which rotates a coordinate x, y, z by an angle θ around a unit vector ν:

$$T(\nu,\theta) = \begin{bmatrix} \cos\theta + x^2(1-\cos\theta) & (1-\cos\theta)yx + z\sin\theta & (1-\cos\theta)zx + y\sin\theta \\ (1-\cos\theta)yx + z\sin\theta & \cos\theta + y^2(1-\cos\theta) & (1-\cos\theta)zy + x\sin\theta \\ (1-\cos\theta)zx + y\sin\theta & (1-\cos\theta)zy + x\sin\theta & \cos\theta + z^2(1-\cos\theta) \end{bmatrix}.$$

After triangulation of the interface with certain constraints on minimal angle and maximum triangle area, the coordinates are reversed back and the contour nodes are mapped back on their original surface positions (Fig. 3.12). This approach to construction of interface preserves original surface mesh and creates quality 2D mesh on the interface with small deterioration of triangles with projected vertices.

It has been found that it can result in the intersection of triangular faces of the interface grid and the original domain surface grid. This happens when a straight line between two nodes, that form an edge of two-dimensional boundary, can not be drawn in three dimensions without intersection with another part of the mesh, that does not form part of the interface. This is immediately detected and the cutting plane is moved on a certain step perpendicular to the minimal principal inertia axis till there are no any intersections. This

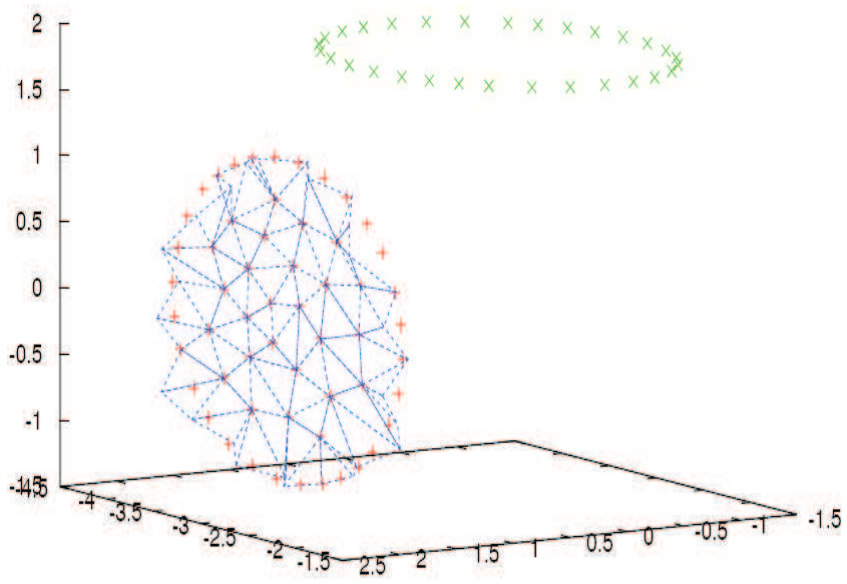


Figure 3.12. Construction of interface. Red – nodes projected on the cutting plane; green – rotation of coordinates into $X - Y$ plane; blue – triangulation of interface and mapping the contour nodes back on original positions.

procedure computationally is not expensive and requires only several reconstructions of two-dimensional interface mesh.

Described here method enables us to get closed and compatible mesh for each subdomain.

3.6. Splitting along path of edges

Produced surface meshes have to meet two major requirements: they should be waterproof, i.e. have no gaps, and consistent. These are necessary conditions for further generation of tetrahedral mesh inside.

So we have an original surface mesh and triangulated interface. Task is to split the original mesh and obtain two subdomains with waterproof and consistent meshes. The general splitting rule for triangles is following: a triangle belongs to a certain part if it has at least two vertices there (Fig. 3.13).

There are some exceptions, where triangulation should be split in a different way. It concerns cases, where decomposition according to general rule results in holes in one subdomain and extra triangles in another (see Fig.3.14).

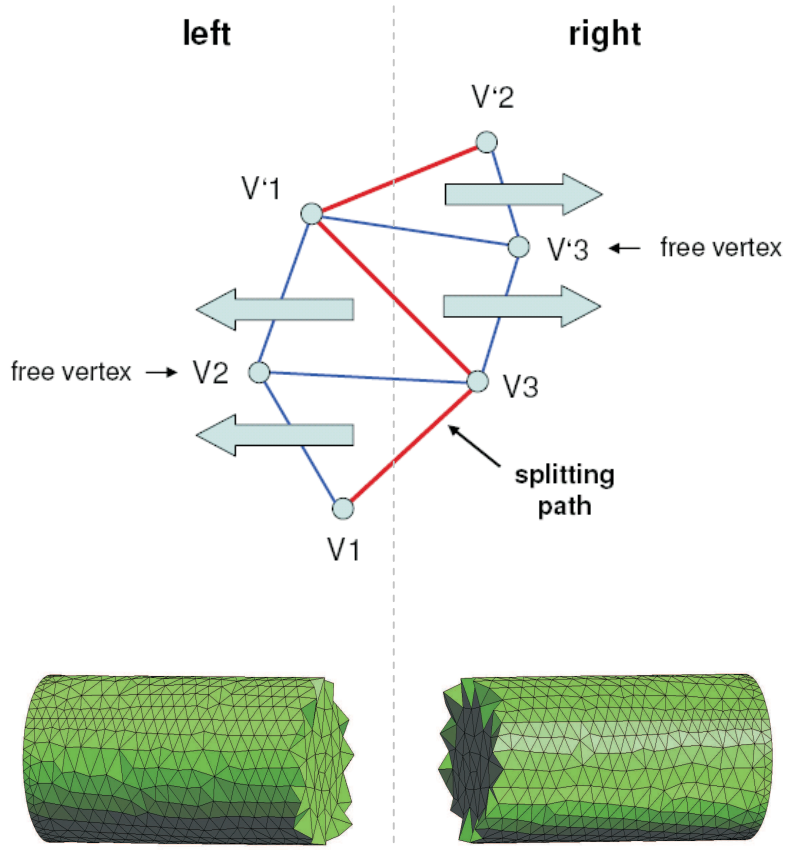


Figure 3.13. Splitting along path of edges. Red – splitting path of edges, blue – the mesh. (V_1, V_2, V_3) and (V_1', V_2', V_3') vertices of two triangles. Triangle (V_1, V_2, V_3) belongs to left subdomain and triangle (V_1', V_2', V_3') to right subdomain.

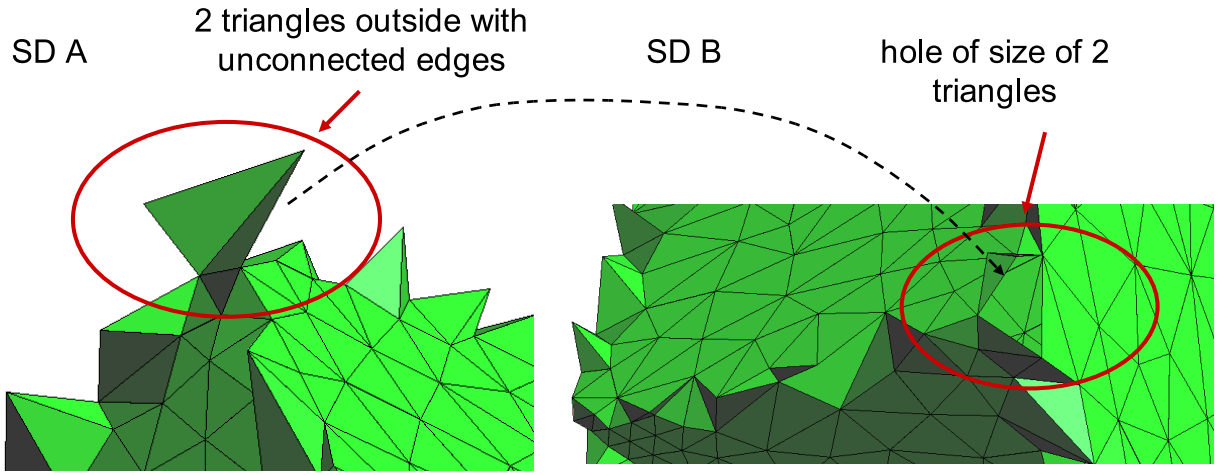


Figure 3.14. Hole in the mesh and moving of triangles according to a special rule.

3.7. Overall domain decomposition

So far decomposition of an object into two parts was described. For obtaining the decomposition of more than two parts, the above described algorithm is applied recursively to each part. New center of mass, eigenvectors and inertia axis are recomputed for each subdomain determining new positions of the cutting planes. So the overall domain decomposition of an object can be represented by the binary tree (see Fig. 3.15), where decomposition depth defines number of resulting subdomains. When the decomposition reaches prescribed number of subdomains (CPUs) it stops and one subdomain is assigned to each CPU for further volume mesh construction. By using numeration (shown in Fig. 3.15) of subdomains, it can be seen that on decomposition level N it is necessary to consider subdomains with numbers from 2^N to $2^{(N+1)} - 1$ and then assign to CPU number $peNum$ subdomain $2^{(N+1)} - 1 - peNum$.

```
Program output:
Decomposition Depth = 3 levels (8 subdomains)
 indexes level 0 | name of source 1 | product names 2 3
 indexes level 1 | name of source 2 | product names 4 5
 indexes level 1 | name of source 3 | product names 6 7
 indexes level 2 | name of source 4 | product names 8 9
```

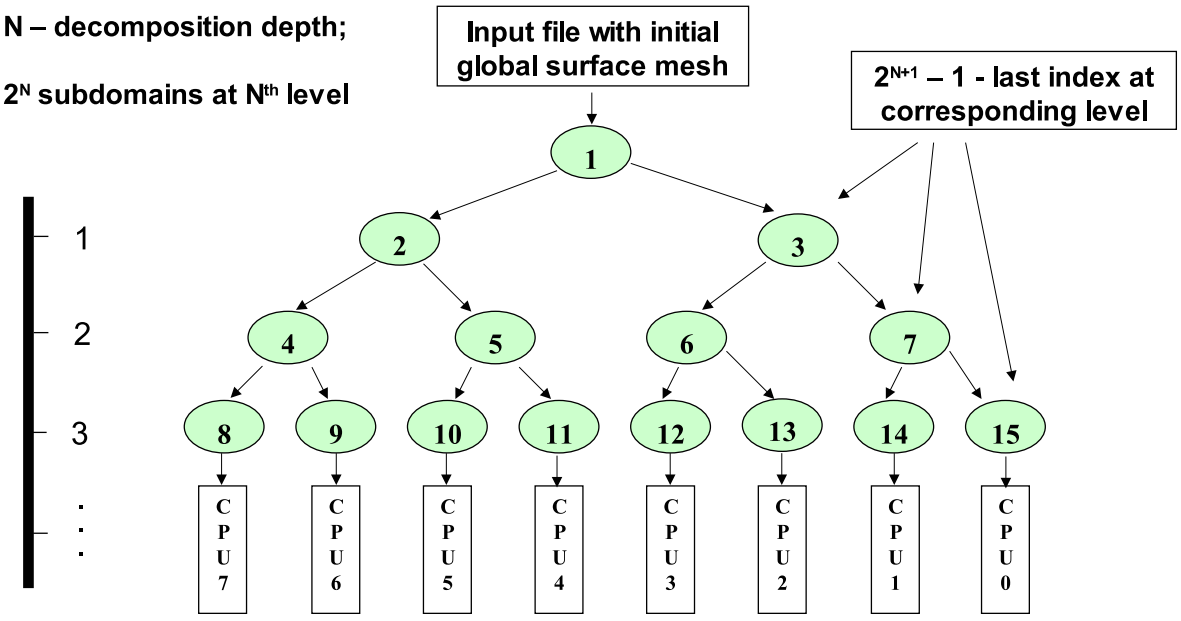


Figure 3.15. Architecture of overall domain decomposition. Decomposition binary tree.

```
indexes level 2 | name of source 5 | product names 10 11
indexes level 2 | name of source 6 | product names 12 13
indexes level 2 | name of source 7 | product names 14 15
```

3.8. Parallel volume mesh generation

When balanced partitioning is done and a closed and compatible surface mesh is constructed for each subdomain, then the volume meshes are constructed in parallel. *TetGen* - a quality tetrahedral mesh generator and three-dimensional Delaunay triangulator [133] can be used for volume Delaunay tetrahedralization with certain quality bound (radius-edge ratio), a maximum volume bound, a maximum area bound on a facet, a maximum edge length on a segment. Fig. 3.16 shows an example of partitioning of cylinder and final tetrahedralization inside of the subdomains on 4 CPUs, where *TetGen* was used. Time speed up of volume mesh generation time in case of cylinder is shown in Fig. 3.17. You can notice that it has superlinear scaling. Explanation of this fact and other related issues will be discussed in the chapter devoted to

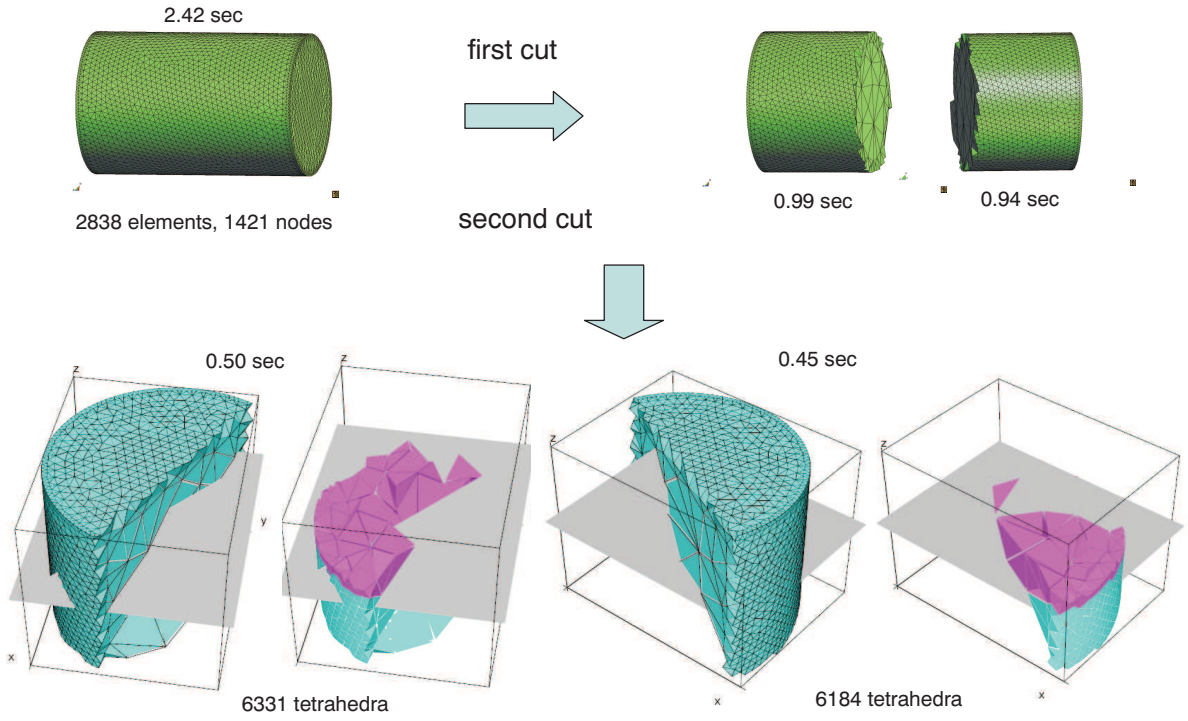


Figure 3.16. Volume mesh generation in each subdomain. Cutting of volume mesh (shown in pink)

results and discussions.

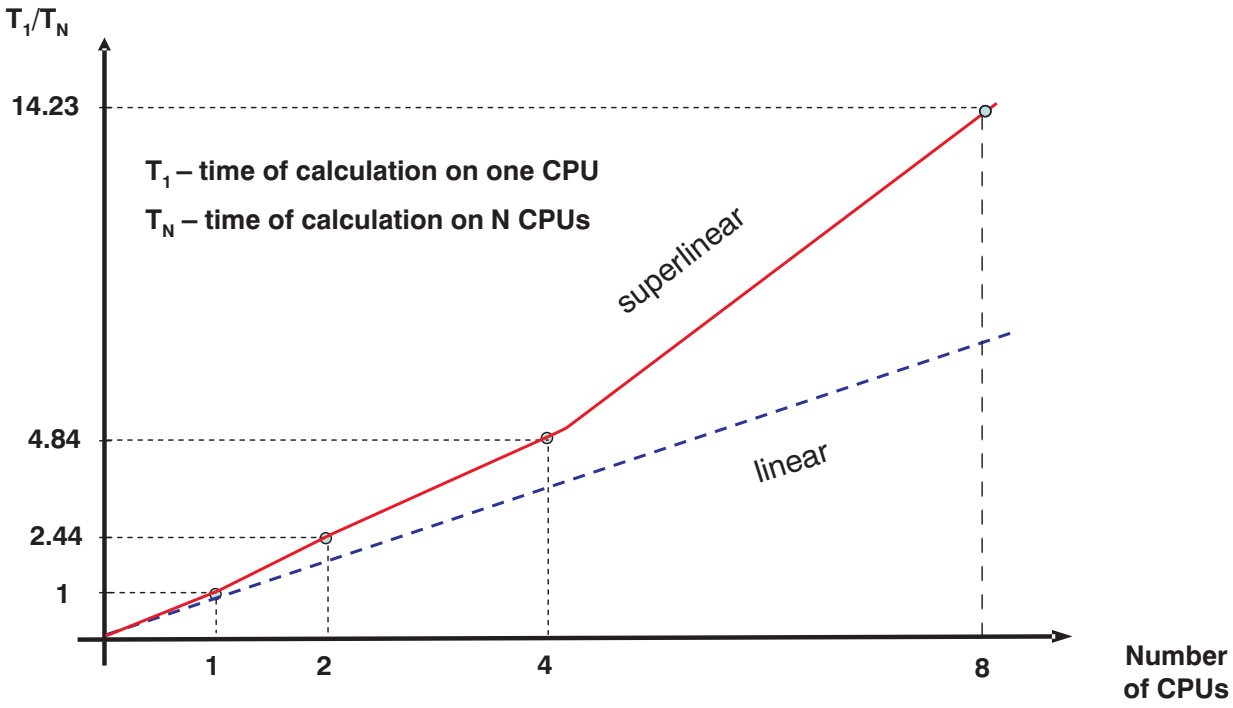


Figure 3.17. Speed-up of volume mesh generation time for test case of cylinder.

Chapter IV

Implementation of parallel grid generator

In the *fourth* chapter the program realization of proposed algorithms is discussed. Adopted parallel computation model (message-passing) and programming model SPMD (Single Program, Multiple Data) are explained. Then programming packages for two-dimensional and three-dimensional triangulations are described. The organization of interprocessor communications based on MPI (Message Passing Interface) along with scheme of parallel work organization of parallel mesh generator are illustrated. Special attention is paid to integration of parallel grid generator with parallel finite element solver. Several real-life examples of computation are given.

4.1. Parallel program realization

4.1.1. Architecture of computational system and programming model of parallel computations

Special feature of MIMD (Multiple Instruction, Multiple Data) is availability of distributed (individual) memory only. That is why data is distributed between nodes. Access to the data on this kind of computational systems is carried out by inter processor communications (program level) which realized by network calls. To ensure the effective parallel mesh generation on computational systems with MIMD architecture employed algorithms should have minimum of interprocessor exchanges and minimum of data transfered.

Efficiency of using of computational system by parallel algorithm can be estimated as:

$$E_P = \frac{S_P}{P} \cdot 100\%,$$

where S_P - acceleration of parallel algorithm realization, computed by:

$$S_P = \frac{T_1}{T_P}.$$

Here T_1 and T_P computational time on one and P processors. In this work it is assumed that number of computational nodes is equal to number of computational processes.

Message-passing model is employed as a model of parallel computations. Message Passing Interface (MPI) [131] library is a standard of message-passing model realization. Utilization of the library as a software for inter processor communications guarantees good portability of programs (on a source code level) onto most of computers.

If each process is executing the same program in this systems, then they realize SPMD programming model. In this model all processes executing, in general case, different branches of the same program (parameterized with regard to identifier, process or processor) but working with different data.

Algorithms presented in this work have been implemented as an application software in C programming language with use of MPI library. It provides opportunity to carry out computations on different computational systems without any modification in communication part.

Application software realizes SPMD parallel programming model and contain code of managing process (MP) as well as computational processes (CP). For the computational processes numeration MPI line topology is employed (processors get numbers from 0 to $P-1$). Number of processes is equal to number of computational nodes. Process with number 0 is carrying out managing functions additionally to functions of CP.

4.1.2. Computational model of parallel grid generator

The main reason for parallelization of grid generation procedure is to get rid of the most expensive computations associated with construction of volume mesh. That is why all decomposition operations are taking place before construction of the volume mesh. Since computational cost of operations with surface mesh is one dimension lower. Based on this fact we can state that the cost of operations with the surface mesh is incomparably lower than for volume mesh. Therefore, decomposition of computational domain is performed

on each processor. Afterwards, one subdomain is assigned to it, where independent volume mesh construction takes place. In Fig. 4.1 scheme of parallel work organization of parallel mesh generation algorithm is shown.

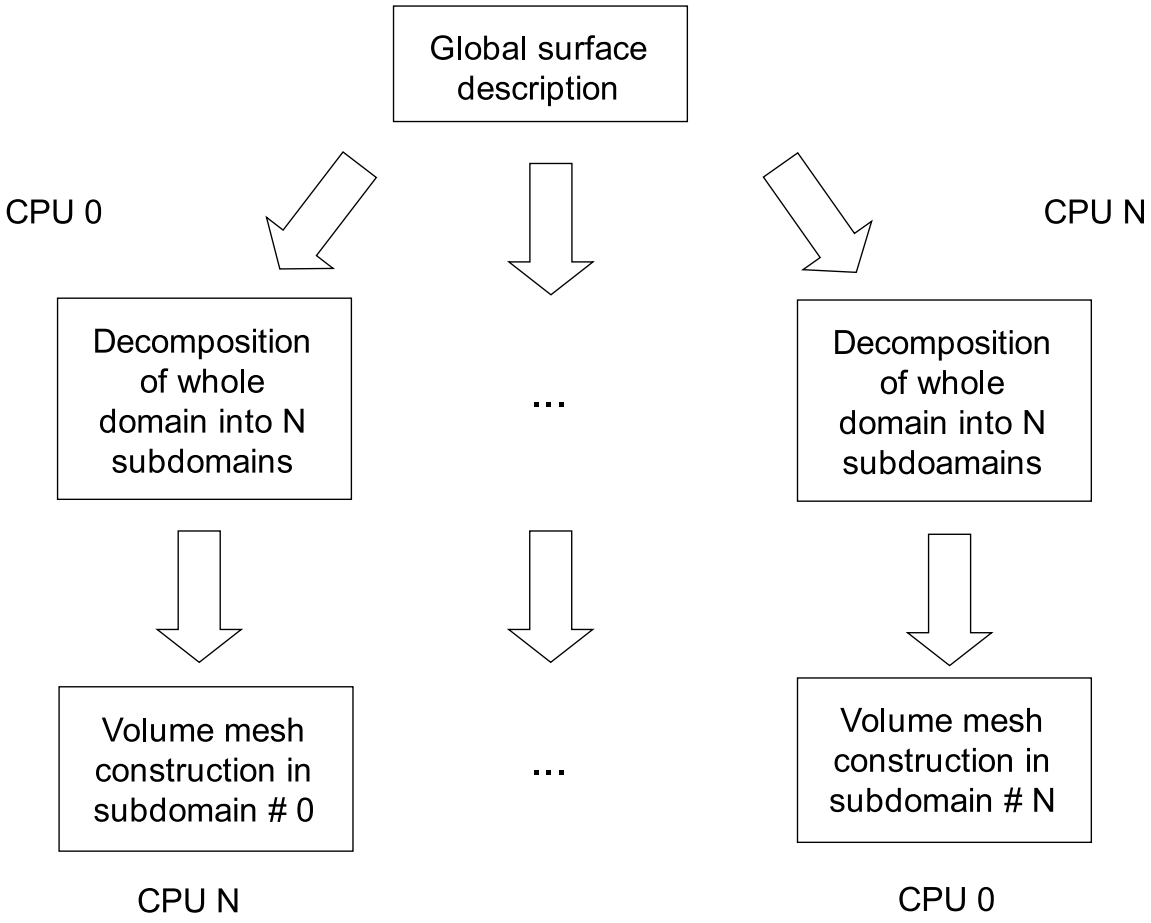


Figure 4.1. Parallel model of grid generator work organization.

4.2. Programming packages for 2D and 3D triangulations

4.2.1. Two-dimensional triangulation. Triangle package.

For the parallel grid generation algorithm program *Triangle* [132] has been used. Its function here is construction of 2D constrained Delaunay triangulation on interface. So it helps us to get closed and compatible mesh for each of subdomain for further independent construction of volume mesh inside.

Triangle is a C program for two-dimensional mesh generation and construction of Delaunay triangulations, constrained Delaunay triangulations, and Voronoi diagrams. Triangle is fast, memory-efficient, and robust. It com-

putes Delaunay triangulations and constrained Delaunay triangulations exactly. Guaranteed-quality meshes (having no small angles) are generated using Ruppert's Delaunay refinement algorithm. Features include user-specified constraints on angles and triangle areas, user-specified holes and concavities, and the economical use of exact arithmetic to improve robustness. Triangle is freely available from URL «http://www.cs.cmu.edu/ quake/triangle.html» and from Netlib «http://www.netlib.org/voronoi/triangle.zip».

4.2.2. Three-dimensional triangulation. TetGen package.

For the volume mesh construction *TetGen* programming package has been used. It enables us to generate tetrahedral mesh based on given surface triangulation.

The TetGen program generates tetrahedral meshes from three-dimensional domains. The goal is to generate suitable tetrahedral meshes for numerical simulation using finite element and finite volume methods. Besides, as a tetrahedral mesh generator, it can be used as a meshing component in many scientific and engineering applications.

For a three-dimensional domain, defined by its boundary (such as a surface mesh), TetGen generates the boundary constrained (Delaunay) tetrahedralization, conforming (Delaunay) tetrahedralization, quality (Delaunay) mesh. The latter is nicely graded and the tetrahedra have circumradius-to-shortest-edge ratio bounded. For a three-dimensional point set, the Delaunay tetrahedralization and convex hull are generated.

The code, written in C++, may be compiled into an executable program or a library for integrating into other applications. All major operating systems, e.g. Unix/Linux, MacOS, Windows, etc, are supported.

The algorithms used in *TetGen* are of Delaunay type. *TetGen* and more detailed information is available at «http://tetgen.berlios.de/index.html».

4.3. Solution of linear algebra problems. LAPACK package.

LAPACK is written in Fortran77 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of

equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision.

The original goal of the LAPACK project was to make the widely used EISPACK and LINPACK libraries run efficiently on shared-memory vector and parallel processors. On these machines, LINPACK and EISPACK are inefficient because their memory access patterns disregard the multi-layered memory hierarchies of the machines, thereby spending too much time moving data instead of doing useful floating-point operations. LAPACK addresses this problem by reorganizing the algorithms to use block matrix operations, such as matrix multiplication, in the innermost loops. These block operations can be optimized for each architecture to account for the memory hierarchy, and so provide a transportable way to achieve high efficiency on diverse modern machines. The term «transportable» is used instead of «portable» because, for fastest possible performance, LAPACK requires that highly optimized block matrix operations be already implemented on each machine.

LAPACK routines are written so that as much as possible of the computation is performed by calls to the Basic Linear Algebra Subprograms (BLAS). While LINPACK and EISPACK are based on the vector operation kernels of the Level 1 BLAS, LAPACK was designed at the outset to exploit the Level 3 BLAS – a set of specifications for Fortran subprograms that do various types of matrix multiplication and the solution of triangular systems with multiple right-hand sides. Because of the coarse granularity of the Level 3 BLAS operations, their use promotes high efficiency on many high-performance computers, particularly if specially coded implementations are provided by the manufacturer.

Highly efficient machine-specific implementations of the BLAS are available for many modern high-performance computers. For details of known vendor- or ISV-provided BLAS, consult the BLAS FAQ. Alternatively, the user can download ATLAS to automatically generate an optimized BLAS library for the architecture. A Fortran77 reference implementation of the BLAS is avail-

able from netlib. However, its use is discouraged as it will not perform as well as a specially tuned implementation.

For instance, LAPACK is used in inertial bisection method. It was necessary to find eigenvalues and eigenvectors of inertia tensor, then to choose eigenvector which corresponds to the smallest eigenvalue. For this purpose DSYEV routine has been used:

```
NAME
       DSYEV - compute all eigenvalues and, optionally,
       eigenvectors of a symmetric matrix A

SYNOPSIS

SUBROUTINE DSYEV(
      JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, INFO ) FO )

      TER     JOBZ, UPLO UPLO

       R       INFO, LDA, LWORK, N K, N

       PRECISION A( LDA, * ), W( * ), WORK( * )  * )

PURPOSE
       DSYEV computes all eigenvalues and, optionally,
       eigenvectors  of  a  real symmetric matrix A.


ARGUMENTS
   JOBZ    (input) CHARACTER*1
           = 'N':  Compute eigenvalues only;
           = 'V':  Compute eigenvalues and eigenvectors.

   UPLO    (input) CHARACTER*1
           = 'U':  Upper triangle of A is stored;
           = 'L':  Lower triangle of A is stored.
```

```
N       (input) INTEGER
        The order of the matrix A.  N >= 0.

A       (input/output) DOUBLE PRECISION array,
        dimension (LDA, N)
        On  entry, the symmetric matrix A.
        If UPLO = 'U', the leading N-by-N upper triangular
        part of A  contains  the  upper  triangular
        part  of  the  matrix A.  If UPLO = 'L', the leading
        N-by-N lower triangular part of A contains the lower
        triangular  part  of  the matrix  A.   On exit,
        if JOBZ = 'V', then if INFO = 0,
        A contains the orthonormal eigenvectors of the
        matrix A.
        If  JOBZ  =  'N',
        then on exit the lower triangle (if UPLO='L') or the
        upper triangle (if UPLO='U') of A, including the
        diagonal, is destroyed.

LDA     (input) INTEGER
        The leading dimension of the array A.
        LDA >= max(1,N).

W       (output) DOUBLE PRECISION array, dimension (N)
        If INFO = 0, the eigenvalues in ascending order.

WORK    (workspace/output) DOUBLE PRECISION array,
        dimension (LWORK)
        On exit, if INFO = 0, WORK(1) returns the
        optimal LWORK.

LWORK   (input) INTEGER
        The length of the array WORK.
```

```
          LWORK >= max(1,3*N-1).
          For  optimal  efficiency, LWORK >= (NB+2)*N,
          where NB is the blocksize for DSYTRD
          returned by ILAENV.

          If LWORK = -1, then a workspace query  is  assumed;
          the  routine only  calculates the optimal size
          of the WORK array, returns this value as the
          first entry of the WORK array, and no error
          message related to LWORK
          is issued by XERBLA.

   INFO   (output) INTEGER
          = 0:  successful exit
          < 0:  if INFO = -i, the i-th argument had an
                illegal value
          > 0:  if INFO = i, the algorithm failed to converge;
                i off-diagonal elements of an intermediate
                  tridiagonal form did not converge to zero.
```

The output of results of calculation of minimal principal inertia axis as a part of parallel grid generator output is shown below.

```
 ************************************************************
  Center of mass coordinates
  X = 0.207480 | Y = 44.594921 | Z = 2.069526
************************************************************

 INERTIA MATRIX
 ************************************************************
 20114508.000000         41528.484375    6864.812988
 41528.484375    56072332.000000         -124917.062500
 6864.812988     -124917.062500          71306504.000000
 ************************************************************

 The eigenvalues in ascending order
```

```
************************************************************
20114459.078229      56071355.865093      71307529.056678
************************************************************

Orthonormal eigenvectors of inertia matrix
************************************************************
0.999999        -0.001155       -0.000137
************************************************************
0.001156        0.999966        0.008198
************************************************************
0.000127        -0.008199       0.999966
************************************************************
```

More detailed information can be found in LAPACK user's guide at « *http* : *//www.netlib.org/lapack/* ».

4.4. Integration of parallel grid generator with parallel FEM solver.

4.4.1. DDFEM - parallel solver for 3D linear elasticity steady-state problems

DDFEM [141] is a linear elasticity solver developed at Fraunhofer ITWM.

Many engineering and scientific applications require the solution of the linear elasticity equations in arbitrary domains. The corresponding boundary value problem (BVP) in its displacement formulation is most often discretized via the finite element method (FEM).

Nowadays the iterative solution of the resulting large scale system of linear equations has no alternative: The problem possesses quasi-optimal complexity, providing effective preconditioners have been used. Additionally, the domain decomposition (DD) techniques and the rapid development of the computer technologies make it possible to solve this problem in parallel, achieving very high performance.

The demand for efficient parallel solvers from the practice is growing, and the numerical theory provides the necessary scientific basis to develop such

solvers. On the other side, many general purpose commercial parallel solvers have a performance, which is rather not satisfactory. Therefore, the need of effective solvers is obvious and DDFEM [141] is an example of such a solver.

The main issue was to develop a reliable, portable and high-performance application for parallel computers. Tetrahedral finite elements are used, because they are appropriate for automatic mesh generation, providing at the same time a good approximation for complicated geometries. DDFEM design follows the object oriented approach, it has been implemented in C++. The solver relies on two external libraries:

- METIS [139] for the mesh partitioning in subdomains,
- PETSc [140] for the (parallel) iterative solution of the assembled linear system.

The iterative linear solver, employed in DDFEM, is the conjugate gradients method with (block-)Jacobi preconditioning, which are provided by PETSc.

For complicated geometries, where no exact solution can be found, so results have been compared with those from other solvers, e.g. with ABAQUS, PERMAS [142]. The comparison shows, that the developed iterative solver produces the same results, and, as a rule, in a shorter time.

DDFEM has been used at ITWM as both:

- a standalone elasticity solver for complex large-sized problems,
- FE-solver, iteratively referenced to perform structural analysis in a shape-optimization loop.

A description with ensuing tetrahedral generation would be a direct mesh description as a set of tetrahedral FEs, together with the corresponding boundary conditions for the problem. Following the latter approach, a special input language for DDFEM has been designed.

The input language provides an interface between the optimization tool and the solver in the process of the shape optimization. An example of resolving such a (real) geometry - part of a casting machine - can be seen on Fig. 4.2.

Let us take up the problem, considered in chapter 5 as an example of real-life problem. The real boundary conditions for the bearing cap is following:

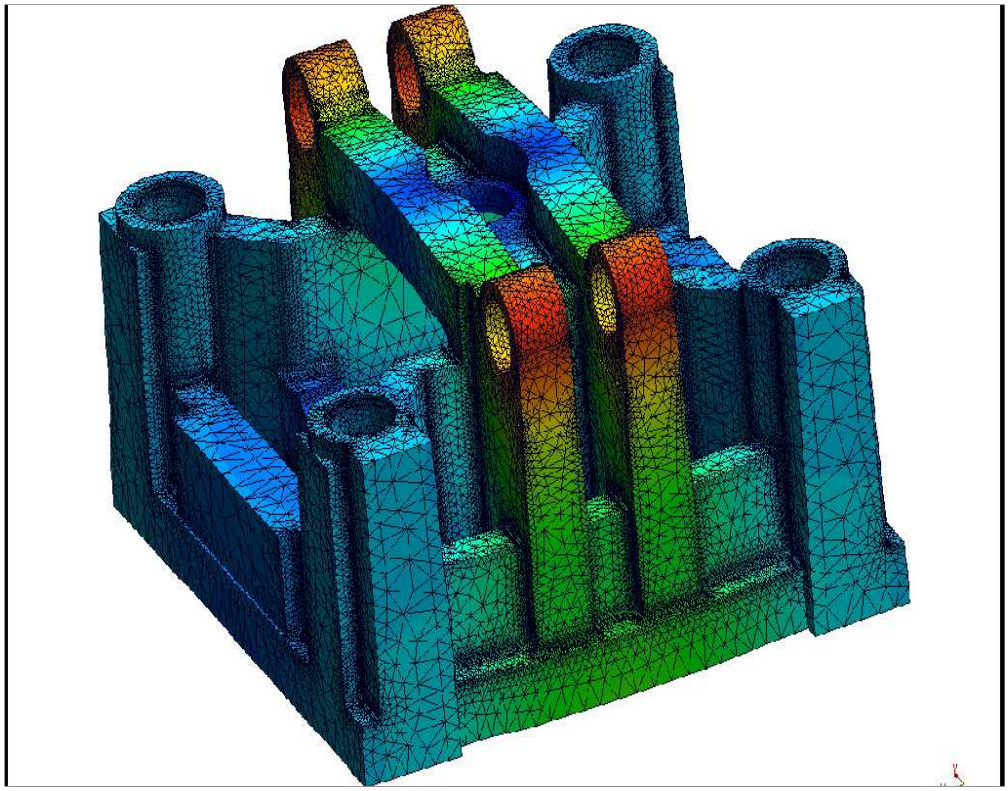


Figure 4.2. Closing plate of a casting machine: 384453 nodes, 2003917 FEs

pressure on top of the bore $P = 130\ \mathrm{N/mm^2}$. It simulates prestress of the screws. At the opposite side object is fixed (displacements are not allowed), and along semi circle surface distributed pressure $P = 100\ \mathrm{N/mm^2}$. Cast iron is chosen as material.

Calculations were performed by DDFEM for described above boundary conditions. In Fig. 4.3 original not deformed bearing cap with stresses is shown. Under applied forces the bearing cap is deforming and in Fig. 4.4 stress on deformed bearing cap is illustrated (deformation factor is 436.812).

The deformation magnitude on original not deformed mesh can be seen in Fig. 4.5. Deformation of mesh with nodes displacements is shown in Fig. 4.6. Deformation factor 436.812 is used for better visual perception of the shape change of the bearing cap.

4.4.2. DDFEM and parallel mesh generator

At present time DDFEM reads sequentially generated volume mesh and then distribute it between CPUs by using METIS mesh partitioning library. Grid

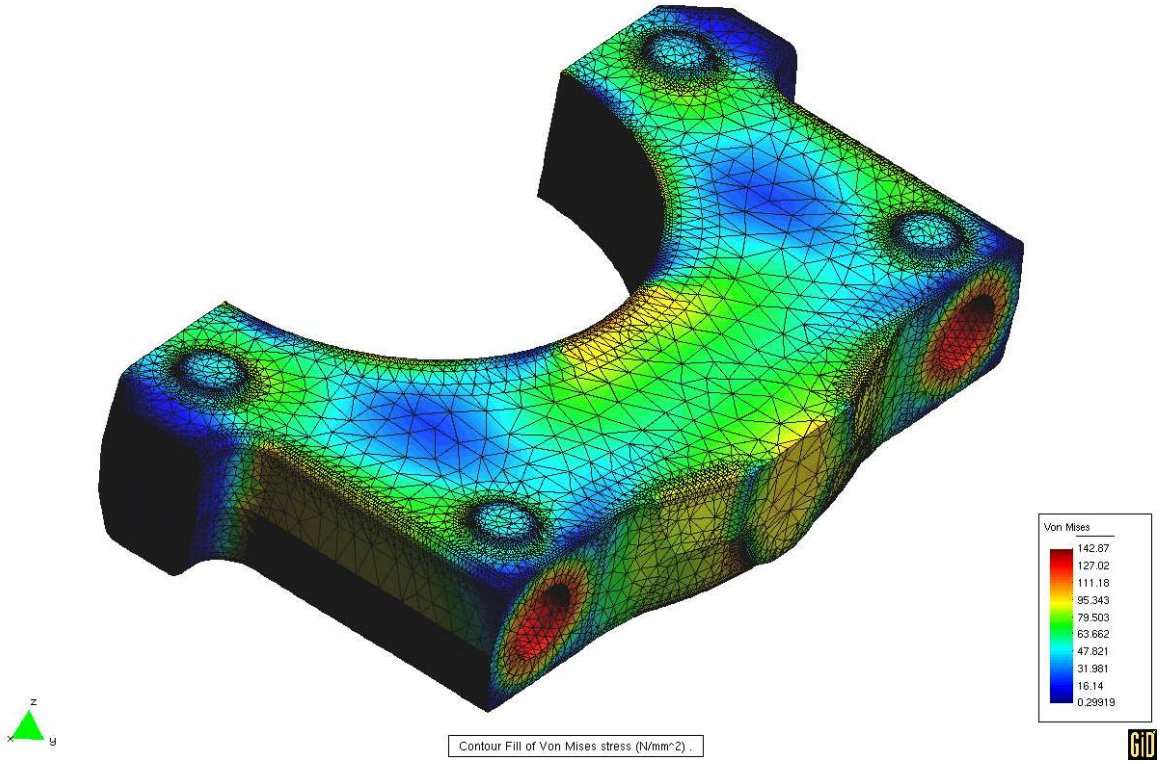


Figure 4.3. Stress on original mesh of bearing cap.

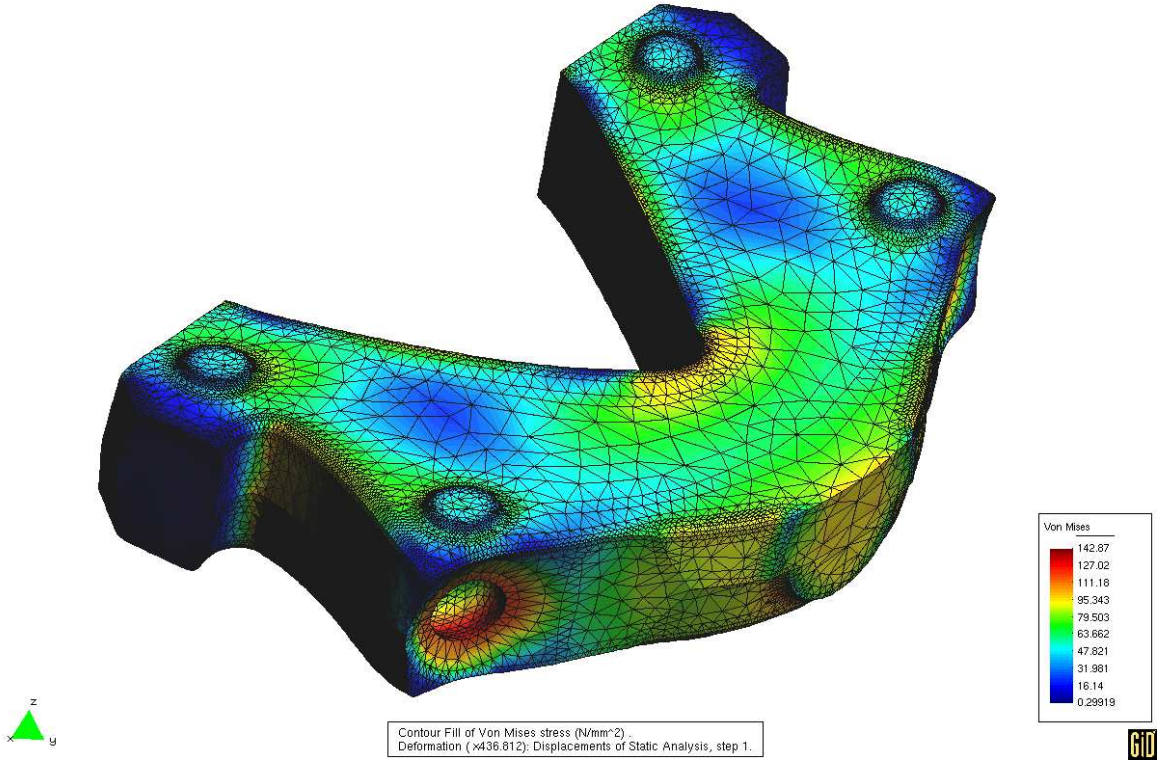


Figure 4.4. Stress on deformed mesh of bearing cap.

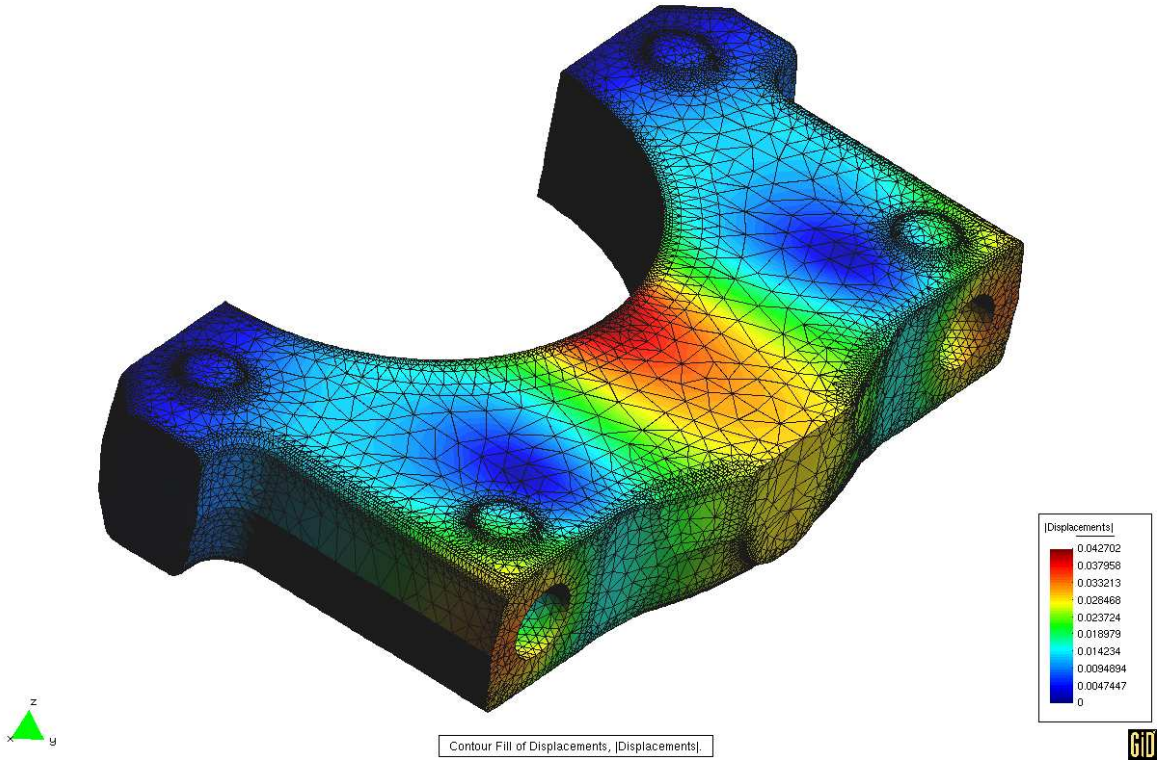


Figure 4.5. Nodal displacements on original mesh of bearing cap.

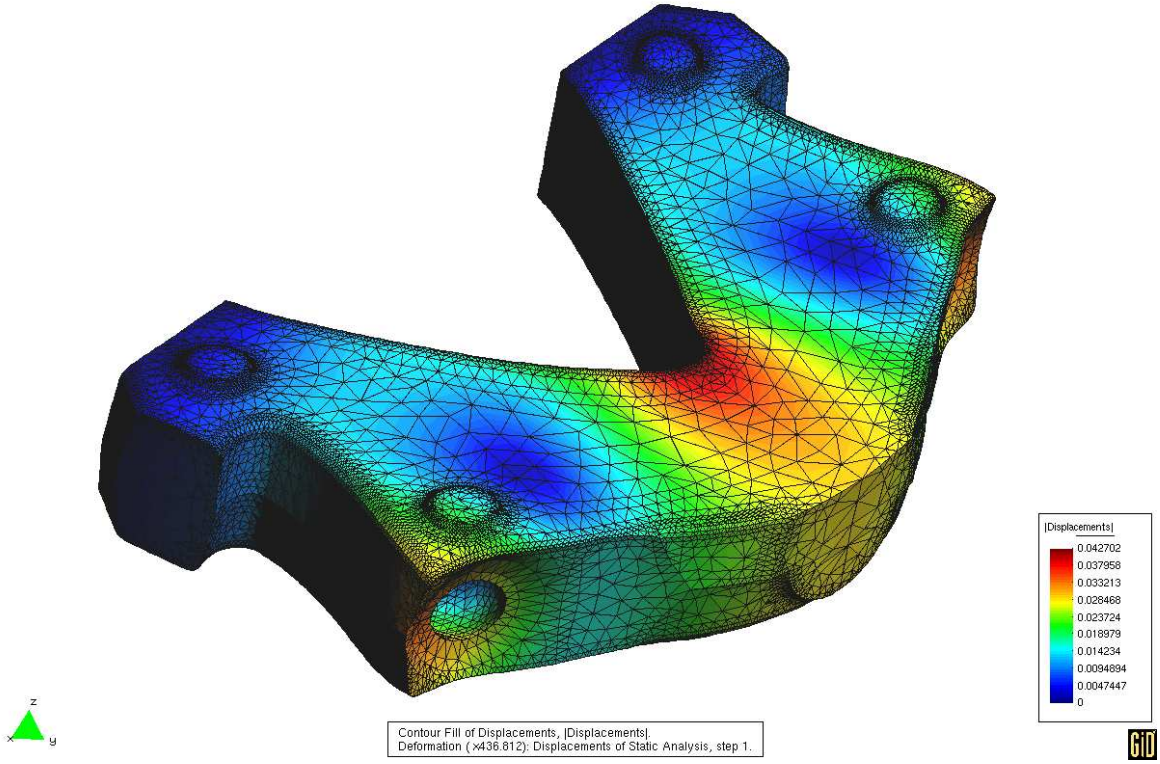


Figure 4.6. Nodal displacements on deformed mesh of bearing cap.

generation stage and distribution of the mesh remains a bottleneck in the whole process of getting solution, since these computations are done sequentially. When large meshes are involved it requires a lot of time and memory.

Therefore integration of parallel grid generator with DDFEM enables us to perform parallel construction of mesh and provides each CPU with required subdomain. Since all steps on the way to the final solution can be done in parallel, the integration of parallel grid generator with DDFEM allows us to achieve throughout parallelization, what increases computational speed dramatically (Fig. 4.7).

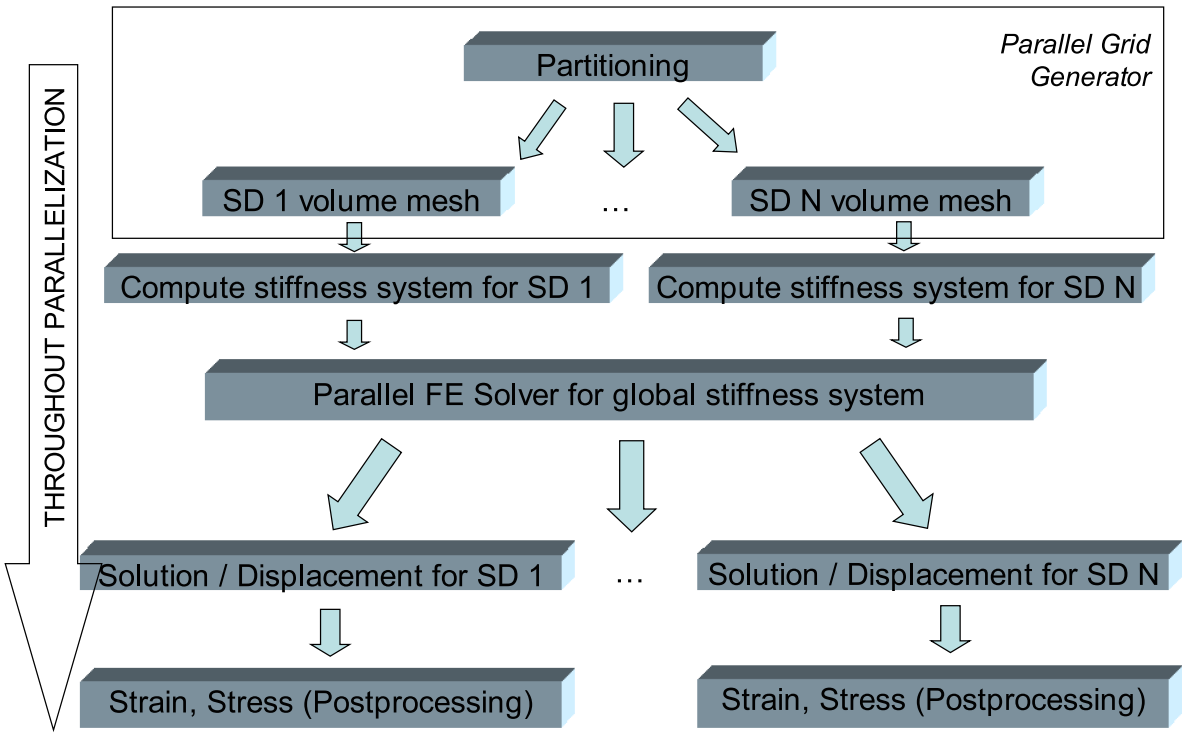


Figure 4.7. Integration of the parallel grid generator with DDFEM. Achieving of throughout parallelization at all stages of solution procedure.

Chapter V

Numerical tests and their analysis

The *fifth* chapter discusses results of computations for real-life problems, where femoral, tibial knee prosthesis components and a bearing cap to fix a crank shaft at a motorblock and others are considered. Decomposition and parallel mesh generation is demonstrated for different mesh sizes and number of processors. Special attention is paid to surface and volume mesh quality. Different aspects of computational efforts related to complexity, computational time, and total area of interfaces are explained. Finally, advantages of parallel grid generator are pointed out and prospective directions of further development are specified as a result of analysis of developed algorithm and of computational results for real-life problems.

Figure 5.1. Multigen - full knee prosthesis consisting of several parts.

5.1. Construction of computational meshes for knee prosthesis components

In Fig. 5.1 a full knee prosthesis consisting of several components is shown.

As first two examples of volume mesh construction in complex domains by using described algorithm components of knee prosthesis produced by Lima Group [144] are taken. In Fig.5.2 and 5.3 an a priori decomposition of original surface meshes in these two cases are shown (16 and 8 correspondingly). It can be seen that the algorithm is able to handle relatively complex geometries.

5.2. Construction of computational mesh for bearing cap

Next example is a bearing cap [145] to fix a crank shaft at a motor block (Fig. 5.4). This example will be further used for more detailed analysis of algorithm efficiency and quality of produced meshes.

A surface mesh representing the shape of the bearing cap is shown in Fig. 5.5.

For demonstration reasons decomposition of the computational domain is done up to 128 subdomains (seven levels of recursion) even need for such a big number of subdomains is not justified. Then volume mesh is constructed in each subdomain assigned specific CPU number. The procedure of domain decomposition and final mesh generation is shown in Fig. 5.6.

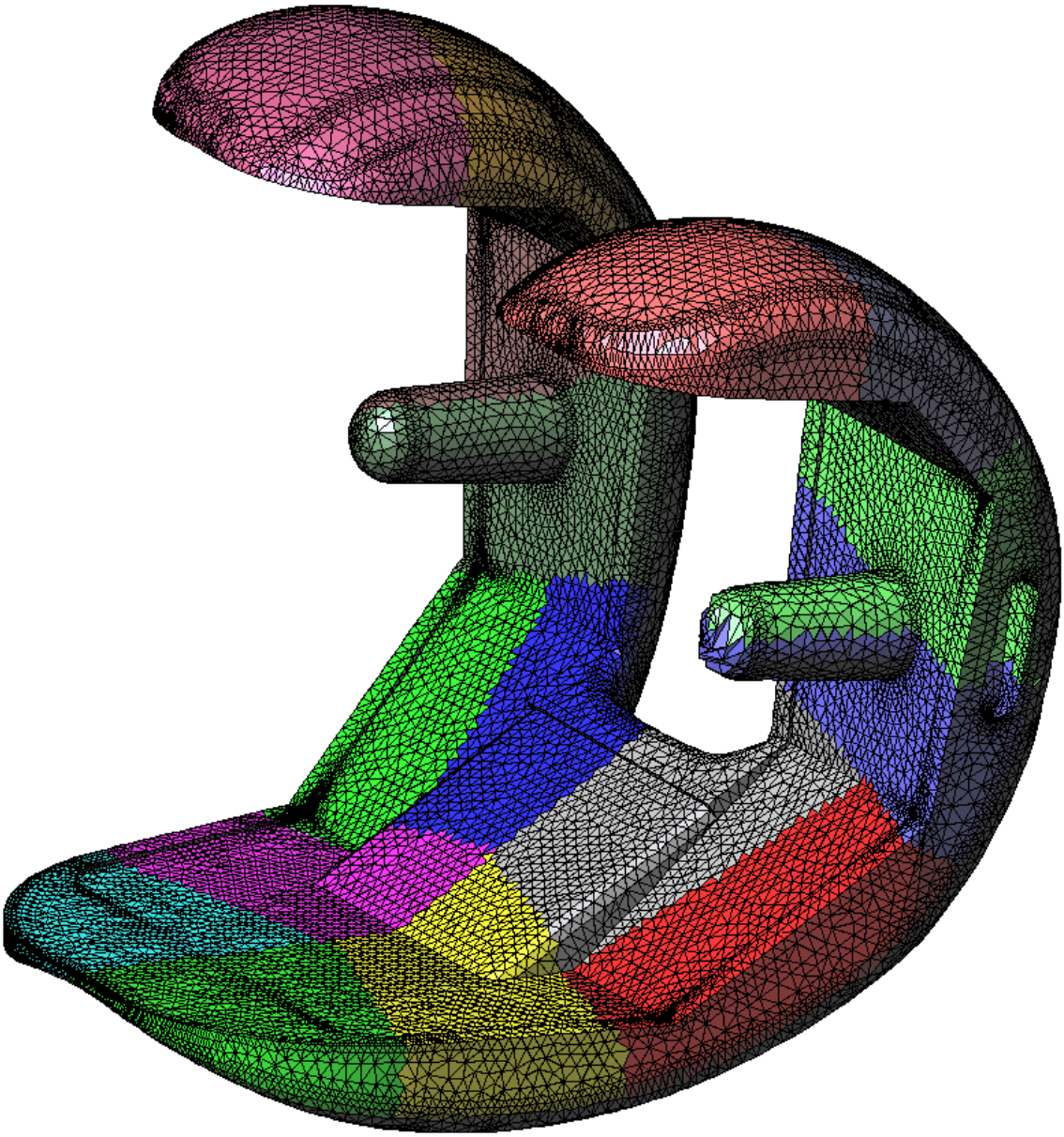


Figure 5.2. An a-priori domain decomposition of knee prosthesis femoral component on 16 subdomains.

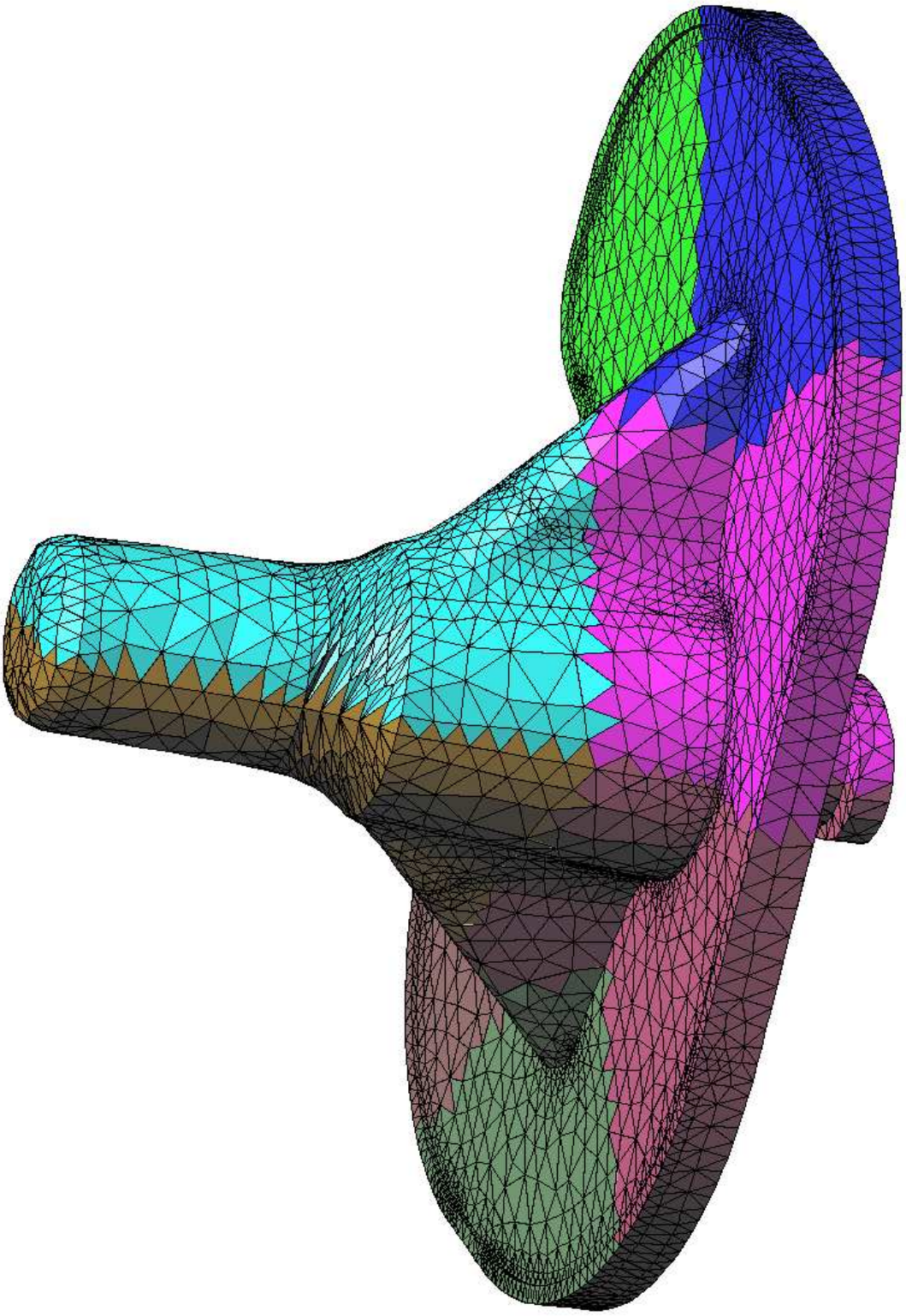


Figure 5.3. An a-priori domain decomposition of knee prosthesis tibial component on 8 subdomains.

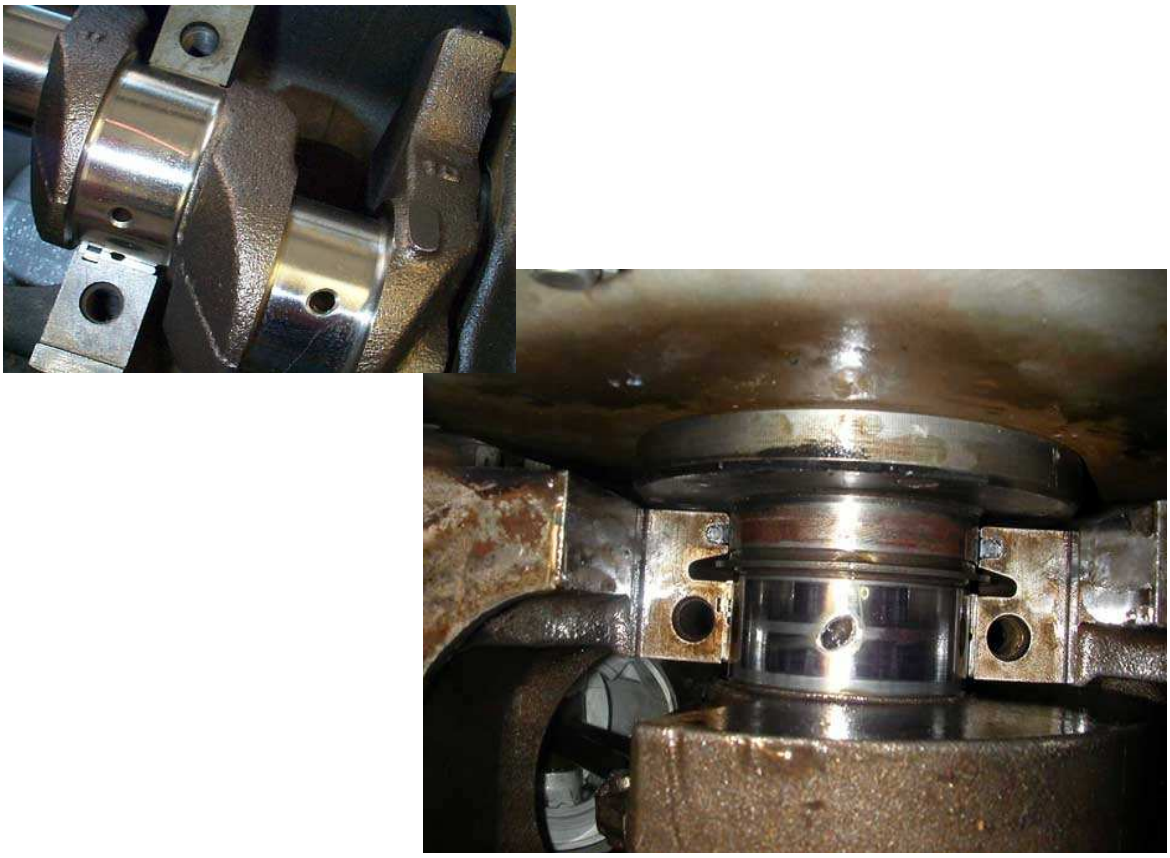


Figure 5.4. Bearing cap in engine.

Figure 5.5. Surface mesh representing the shape of the bearing cap.

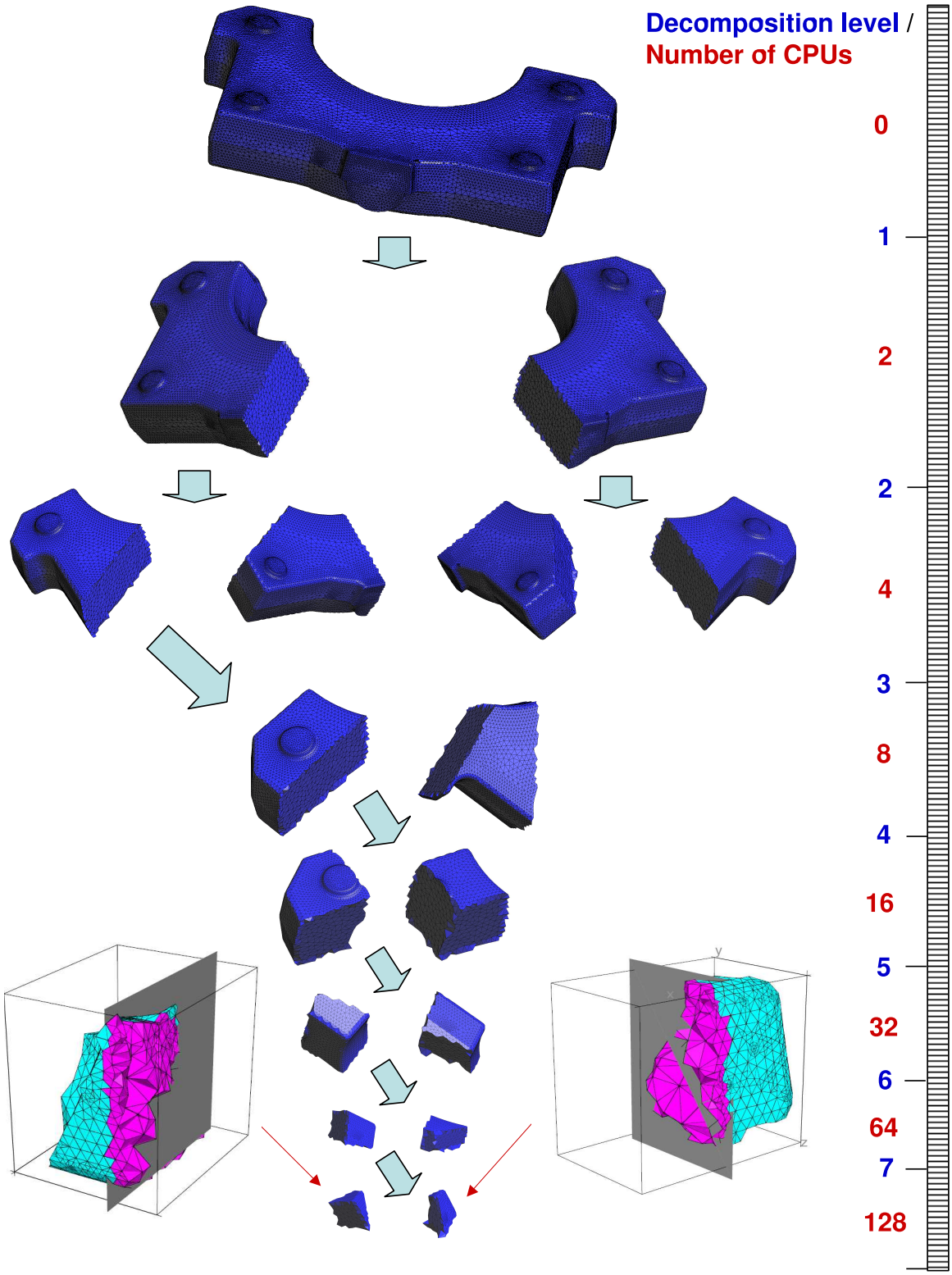


Figure 5.6. An a-priori domain decomposition of bearing cap component into 128 subdomains.

The simulation was run on the Fraunhofer ITWM cluster with high - bandwidth /low-latency myrinet network with 64 nodes. Each node has Dual Xeon CPU 2.4 GHz, 4GB RAM. Three meshes of different sizes were constructed. In Table 5.1 generation time on a different number of CPUs is given.

Table 5.1. Computational time for construction of volume meshes of different sizes

Elements/CPUs	1 CPU	2 CPU	4 CPU	8 CPU	16 CPU	32 CPU
$4{\cdot}10^5$ elements	20.91 s	10.26 s	5.78 s	3.30 s	2.54 s	1.25 s
$4{\cdot}10^6$ elements	169.56 s	83.07 s	48.43 s	26.75 s	15.02 s	9.06 s
$4{\cdot}10^7$ elements	failed	961.45 s	558.197 s	356.39 s	181.08 s	91.13 s

In case of 400 thousands and 4 millions elements the computational time was reduced dozens of times owing to developed parallel grid generator. If generation of these two meshes is still possible on a single CPU due to reasonable size, then sequential generation of the third mesh with more than 40 millions elements fails. Developed parallel grid generator constructs this mesh on 32 CPUs in approximately 1.5 minutes. Diagram of computational time in exponential scaling is shown in Fig. 5.7.

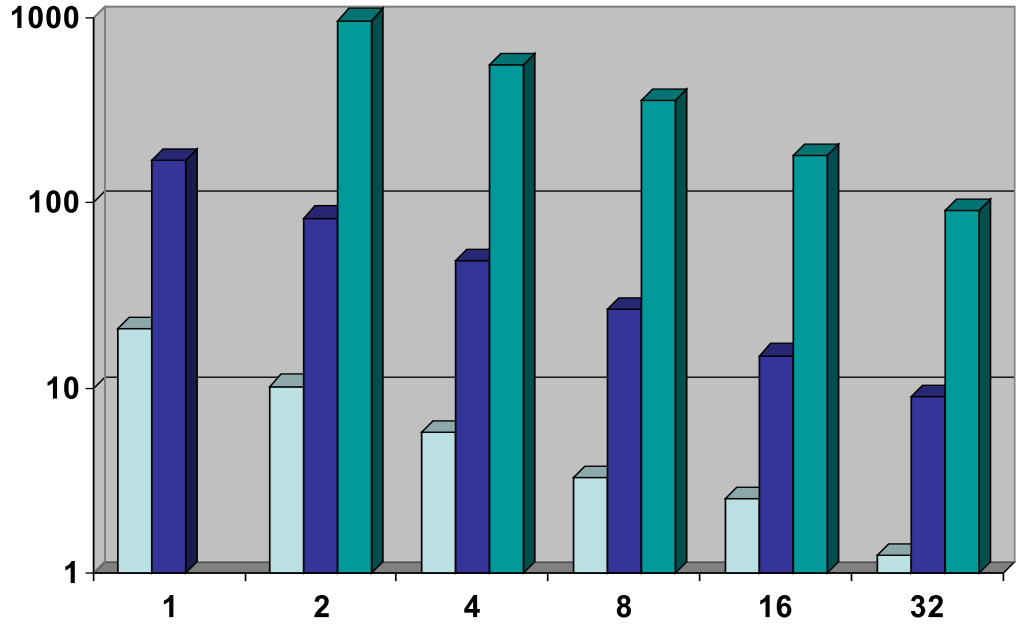


Figure 5.7. Computational time for meshes of different size. Logarithmic scaling. First column corresponds to 400 thousands, second to 4 millions and third to 40 millions elements.

The speed-up of volume mesh generation time for the cases of 400 thousands, 4 millions and 40 millions elements are illustrated in Fig. 5.8, Fig. 5.9 and Fig. 5.10 correspondingly. Fig. 5.11 gives the comparison between these three plots.

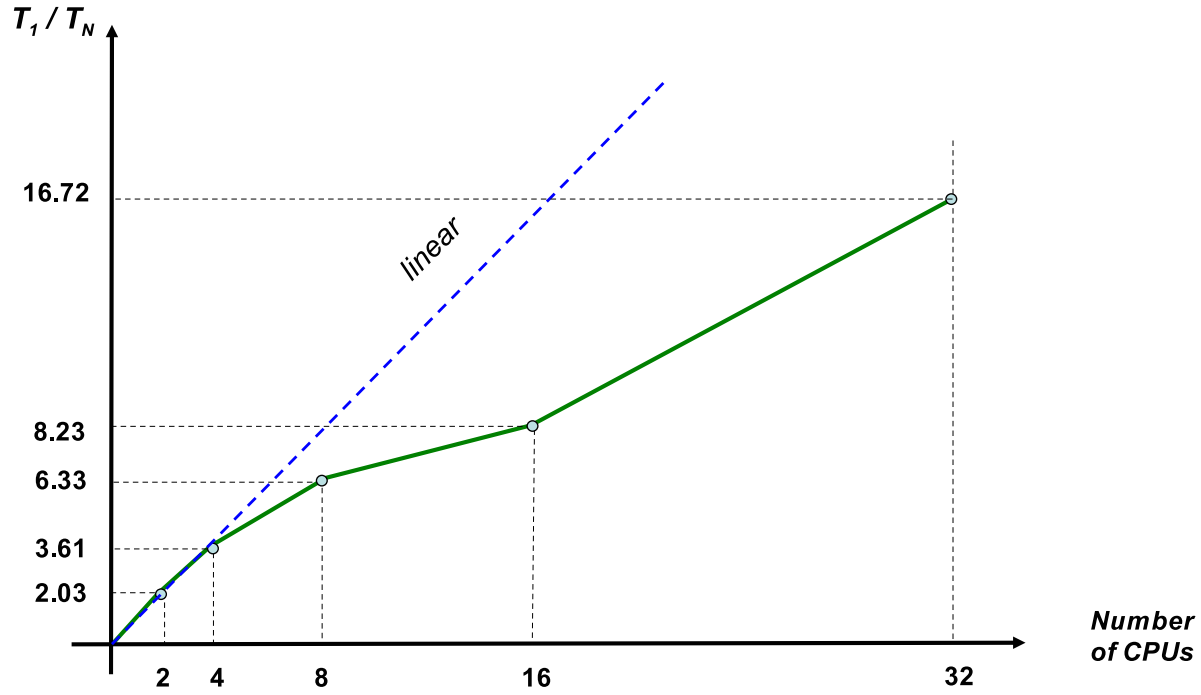


Figure 5.8. Speed-up of volume mesh generation time for the bearing cap geometry with 400 thousands tetrahedral elements.

As it is seen in Fig. 5.11 efficiency of parallel grid generator work on 32 CPUs is higher for larger mesh. This is quite reasonable since you provide each CPU with enough work in case of larger mesh and, clearly parallel realization is not so efficient if mesh is small. In Fig. 5.12 example of generation of relatively small mesh of 300 thousands elements on large number of CPUs (up to 128) is illustrated. One can notice that the efficiency is lower than in all considered before cases. The highest efficiency which the parallel grid generator reaches is 70% in case of 40 millions elements.

Developed parallel grid generator helps to remove a computational bottleneck related with sequential construction of volume mesh. It can be used with parallel solvers providing prepared subdomain for each CPU as well as with other sequential solvers to reduce mesh generation time. In latter case subdomains should be joined back together into one global volume mesh (see

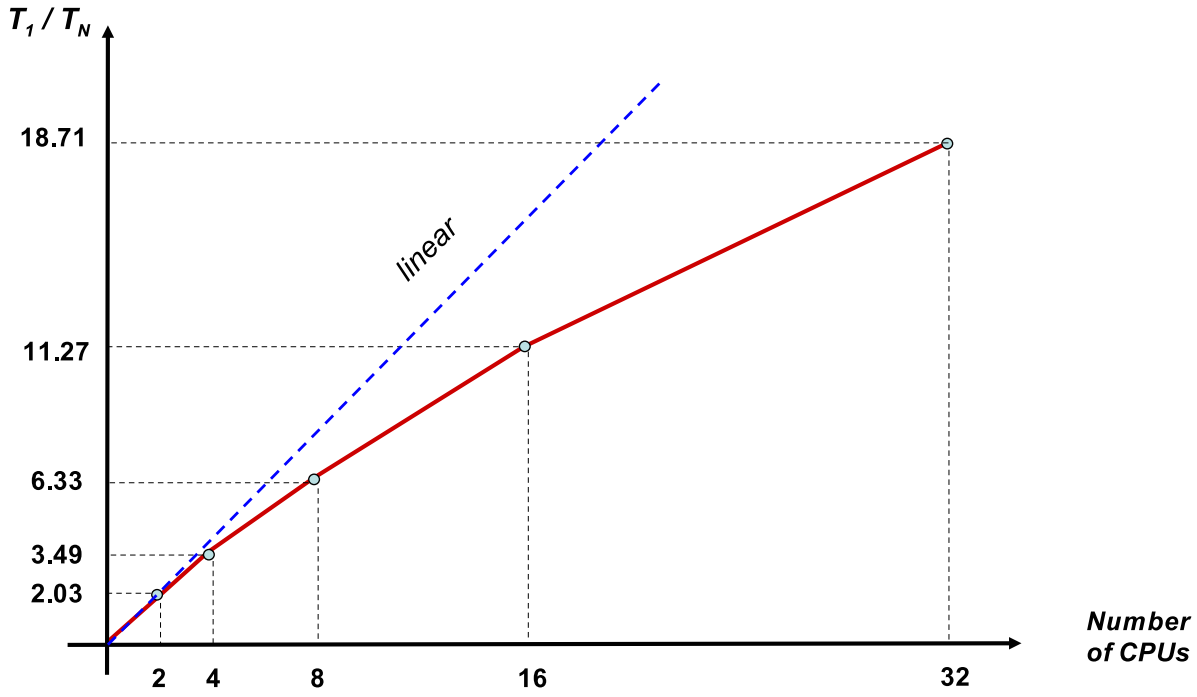


Figure 5.9. Speed-up of volume mesh generation time for the bearing cap geometry with 4 millions tetrahedral elements.

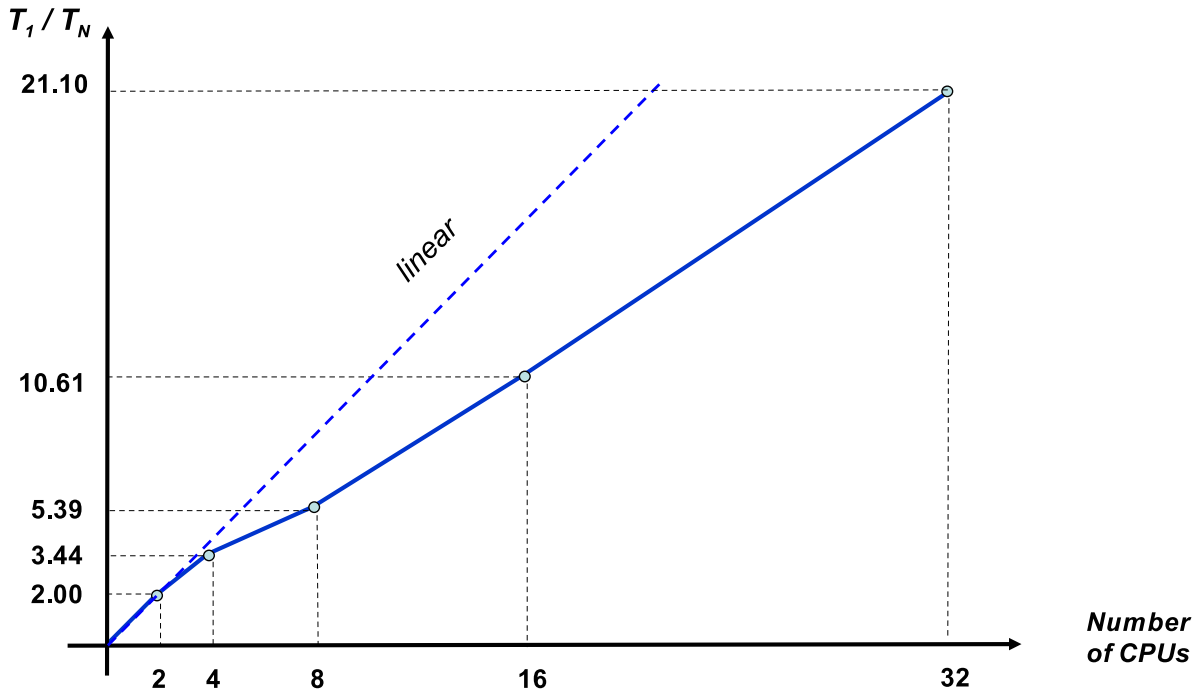


Figure 5.10. Speed-up of volume mesh generation time for the bearing cap geometry with 40 millions tetrahedral elements.

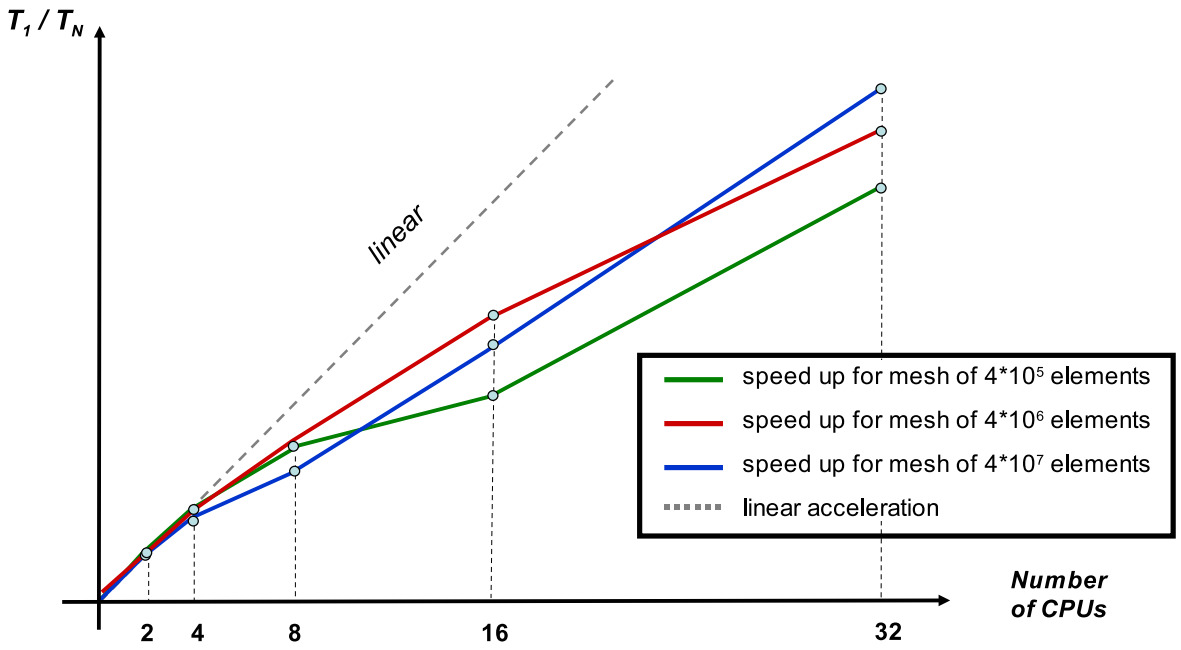


Figure 5.11. Speed-up of volume mesh generation time for the bearing cap geometry. 400 thousands, 4 millions and 40 millions tetrahedral elements.

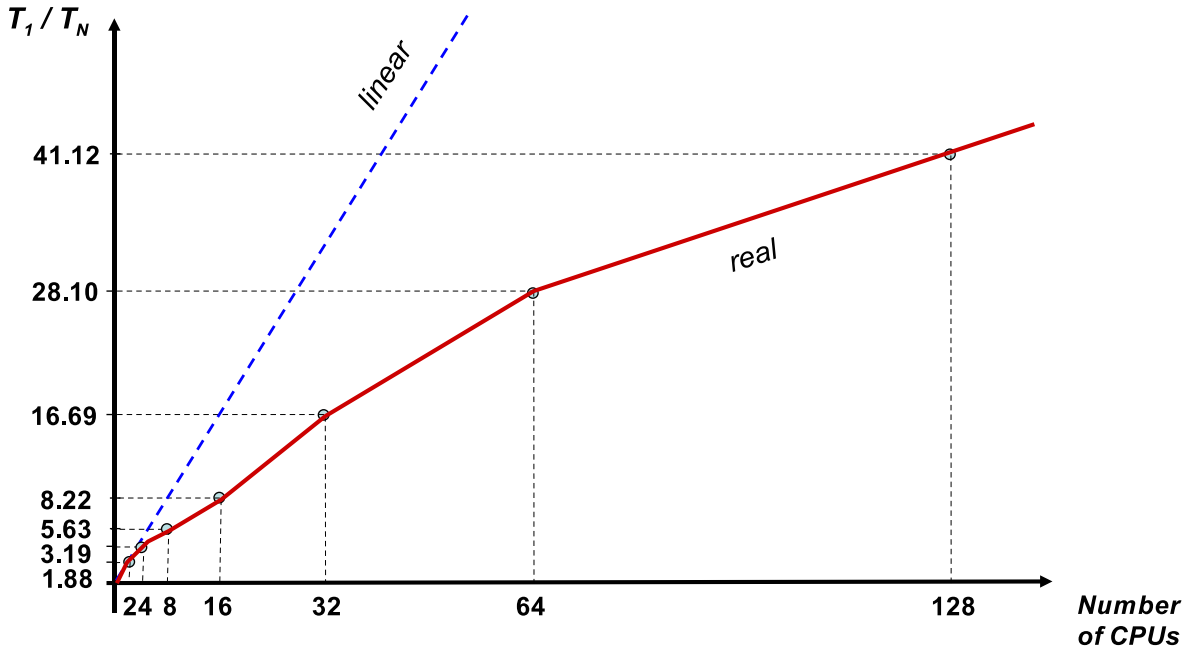


Figure 5.12. Speed-up of volume mesh generation time for the bearing cap geometry in case of relatively small mesh (300 thousands elements) and big CPU number (up to 128).

Fig.5.13).

Currently, construction of very large meshes (several hundreds of millions

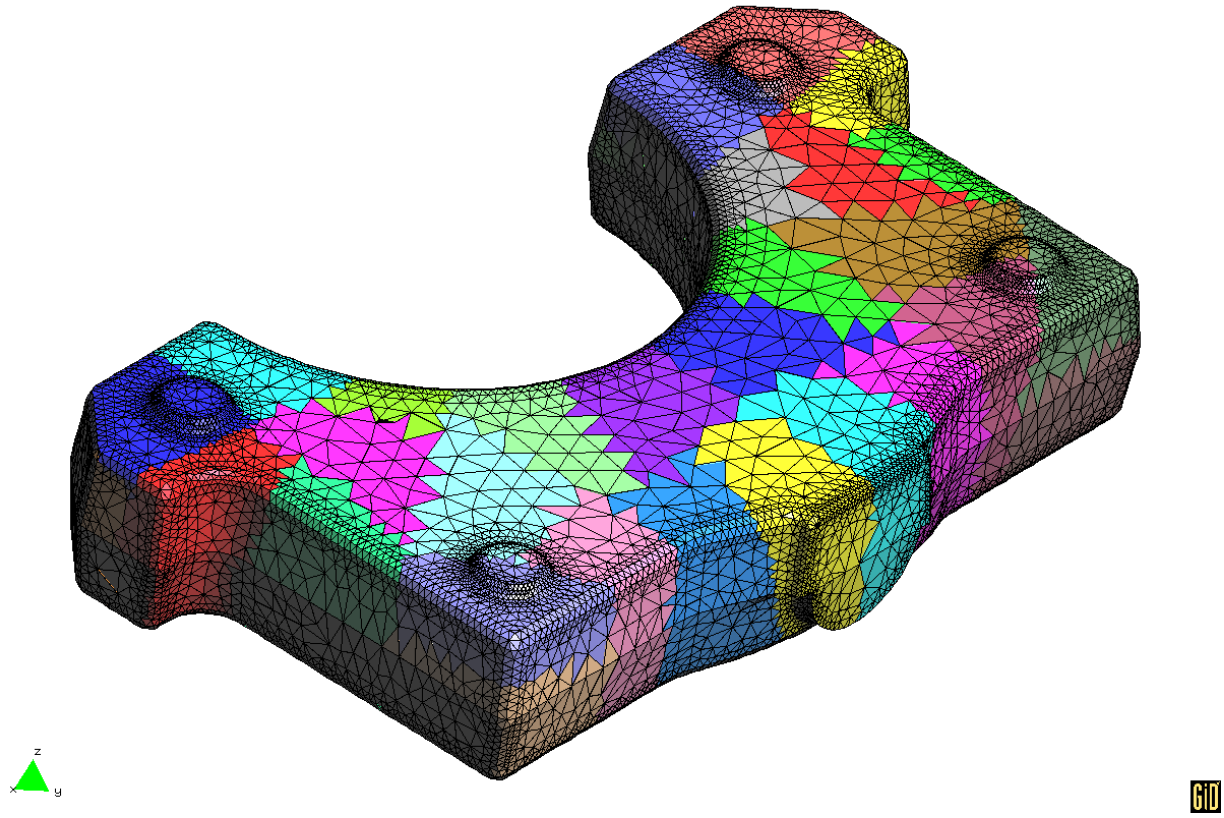


Figure 5.13. Volume mesh of a bearing cap constructed by parallel grid generator.

elements) is possible. For instance, generation of mesh with 200 millions elements on 8 CPUs takes less than 10 minutes.

5.3. Quality of computational mesh

5.3.1. Quality of surface triangulation

It was already mentioned that the mesh quality is an important property to pay attention to. The partitioning algorithm should not adversely affect the quality of the surface mesh. The most undesirable triangles for FEM computations are those with very acute angles. One of the major goals of the decomposition algorithm is to make quality loss minimal.

We assume that the original given surface mesh is of good quality. The only thing that can affect surface mesh quality is construction of triangulation on interface and projection operations with contour nodes.

The two-dimensional triangulator we employ uses Delaunay refinement technique. This is a technique for generating triangular meshes suitable for

use in interpolation, the FEM, and the FVM. The problem is to find a triangulation that covers a specified domain and contains only triangles whose shapes and sizes satisfy constraints: the angles should not be too small or too large, and the triangles should not be much smaller than necessary, nor larger than desired. Delaunay refinement algorithms offer mathematical guarantees that such constraints can be met. They also perform excellently in practice.

The triangles should be relatively «regular» in shape, because triangles with large or small angles can degrade the approximation quality of the numerical solution to a finite element problem. In interpolation, triangles with large angles can cause large errors in the gradients of the interpolated surface. In the finite element method, large angles can cause a large discretization error. The solution may be less accurate than the method would normally promise. Small angles can cause the coupled systems of algebraic equations that the finite element method yields to be ill-conditioned.

The quality grid generator *Triangle* is a hybrid of by L. Paul Chew's and Jim Ruppert's Delaunay refinement algorithm. It adds vertices to the mesh for prevention of angles less than 20^o. Clear, that acute angles in input segment can not be already removed, but it is possible to avoid any other acute angle. There is a possibility to specify desired minimum angle of a triangulation. If the specified angles are 20.7^o or less then convergence is guaranteed. In practice algorithm works with angles up to 33^o and usually does not terminate for angles more than 34^o. For some meshes it might be necessary to reduce minimum angle in order to avoid problems related with accuracy of the calculations.

In Fig. 5.14 and Fig. 5.15 triangles for prosthesis component with angle less than 30^o and for bearing cap with angles less than 40^o are shown before and after decomposition procedure.

5.3.2. Quality of tetrahedral mesh

The main goal of the parallel grid generator is to produce meshes suitable for solving partial differential equations (PDEs) by finite element methods (FEM) and finite volume methods (FVM).

The problem is to generate a tetrahedral mesh conforming to a given (polyhedral or piecewise linear) domain together with certain constraints for the

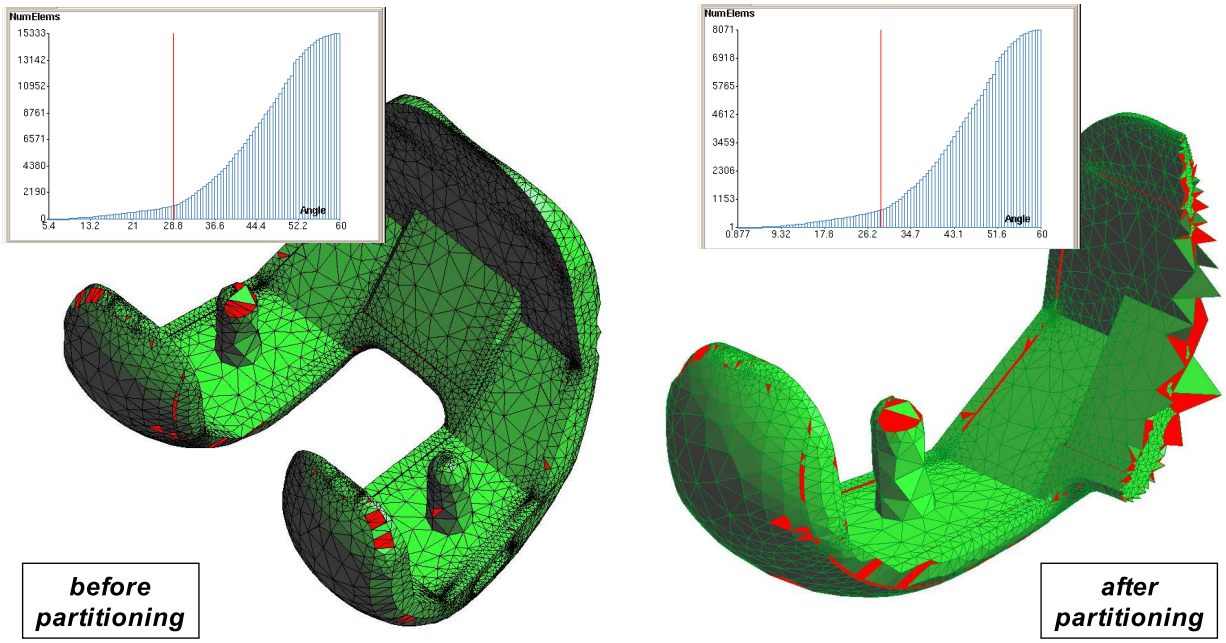


Figure 5.14. Surface quality for mesh of knee prosthesis before and after partitioning. Triangles with angles less than 30^o.

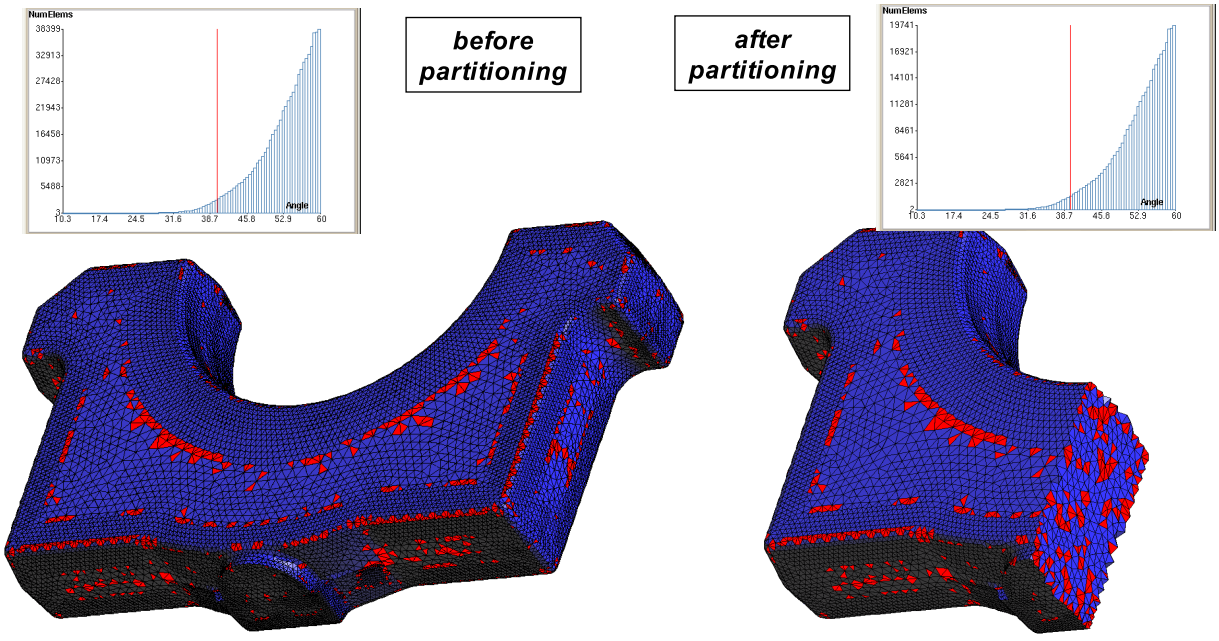


Figure 5.15. Surface quality for mesh of bearing cap before and after partitioning. Triangles with angles less than 40^o.

size and shape of the mesh elements. It is a typical problem of provably good mesh generation or quality mesh generation. The techniques of quality mesh generation provide the «shape» and «size» guarantees on the meshes:

- All elements have a certain quality measure bounded
- The number of elements is within a constant factor of the minimum number.

The approaches to solve quality mesh generation include octree, advancing front, and Delaunay methods.

Delaunay refinement, a Delaunay tetrahedralization is refined by iteratively adding vertices. The placement of these vertices is chosen to enforce boundary conformity and to improve the quality of the mesh. Delaunay refinement was successfully applied to the corresponding two-dimensional problem (see previous section). Such algorithms can be found in the work of Chew, and Ruppert. However, these algorithms do not remove slivers (very flat and nearly degenerate tetrahedra) in three dimensions.

The algorithm *TetGen* implemented to tackle this problem is a Delaunay refinement algorithm from Shewchuk [134]. It is a smooth generalization of Ruppert's algorithm to three dimensions. Given a complex of vertices, constraining segments and facets in three dimensions, with no input angle less than 90^o, this algorithm can generate a quality mesh of Delaunay tetrahedra with radius-edge ratios not greater than 2.0. Tetrahedra are graded from small to large over a short distances. The algorithm generates meshes generally surpassing the theoretical bounds and is effectively in eliminates tetrahedra with small or large dihedral angles.

Except the tetrahedra near small input angles, the sliver is the only type of badly-shaped tetrahedron which could survive after the Delaunay refinement. Several techniques [135–137] have been developed to remove slivers from the mesh. TetGen does a simplified sliver removal step. Slivers are removed by local flip operations and peeling off from the boundary. This strategy is effective to remove most of the slivers but does not guarantee to remove all of them.

There are several quality measures available in literature. For accuracy in the FEM, it is generally necessary that the shapes of elements have bounded aspect ratio. The aspect ratio of an element is the ratio of the maximum side length to the minimum altitude. For a quality mesh, this value should as small as possible. For example, «thin and flat» tetrahedra tend to have large aspect ratio.

A similar but weaker quality measure is radius-edge ratio, proposed by Miller, Talmor, Teng, Walkington, and Wang [138]. A tetrahedron t has a unique circumsphere. Let $R = R(t)$ be that radius and $L = L(t)$ the length of the shortest edge. The radius-edge ratio $Q = Q(t)$ of the tetrahedron is given by:

$$Q = \frac{R}{L}$$

The radius-edge ratio measures the quality of a tetrahedron. For all well-shaped tetrahedra, this value is small (Fig. 5.16(A)), while for most of badly-shaped tetrahedra, this value is large (Fig. 5.16(B)). Hence, in a quality mesh, this value should be bounded as small as possible. However, the ratio is minimized by the regular tetrahedron (in which case the lengths of the six edges are equal, and the circumcenter is the barycenter), that is

$$Q \geq \sqrt{6}/4 \approx 0.612$$

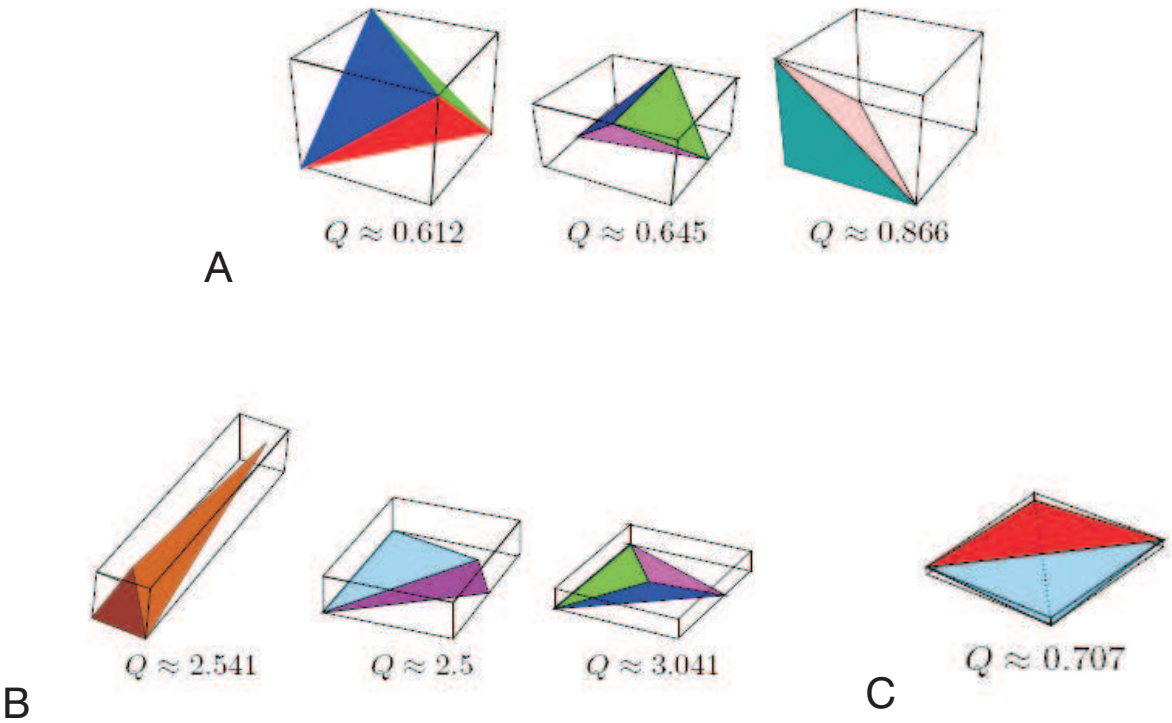


Figure 5.16. The radius-edge ratio for some well-shaped and badly-shaped tetrahedra. A - the radius-edge ratio for some well-shaped tetrahedra; B - the radius-edge ratios for some badly-shaped tetrahedra; C - sliver (special type of badly-shaped tetrahedron).

A special type of badly-shaped tetrahedron is called sliver (see Fig. 5.16(C)), which is very flat and nearly degenerate. Slivers can have radius-edge ratio as small as $\sqrt{2}/2 \approx 0.707$. The radius-edge ratio is not a proper measure for slivers. However, Miller, Talmor, Teng, Walkington, and Wang [138] have pointed out that it is the most natural and elegant measure for analyzing Delaunay refinement algorithms.

It is possible to control quality of the mesh by specifying upper bound of the quality coefficient and volume of tetrahedra. The algorithm performs quality mesh generation by Shewchuk's Delaunay refinement algorithm. It adds vertices to the CDT (Constrained Delaunay Triangulation) or a previously generated mesh to ensure that no tetrahedra have radius-edge ratio greater than 2.0. An alternative minimum radius-edge ratio may be specified. If too small ratio is supplied (smaller than 1.0), algorithm may not terminate.

If no input angle or input dihedral angle (of the Piecewise Linear Complex) smaller than 60^o degree, this algorithm is theoretically guaranteed to terminate with no tetrahedron has radius-edge ratio greater than 2.0. In practice, this algorithm often successes for radius-edge ratio be 1.414 or even smaller.

In Fig. 5.17a example is shown. It is a wing of an airplane and a box enclosed it. The mesh domain is outside the wing and bounded by the box. In Fig. 5.17b the quality mesh (shown below on the right) is generated. Default, the radius-edge ratio of each tetrahedron is bounded below 2.0. You can impose a tight bound, for example 1.2. So in Fig. 5.17c program generates a quality mesh which have more points inserted than the mesh created in above. Another way to refine mesh is to impose a maximum volume constraint on the mesh. Fig. 5.17d shows a quality mesh which has both radius-edge ratio bounded by 1.2 and maximum volume bounded by 0.0001. Tetrahedral grid generator prints the mesh quality statistics after meshing. It looks like the follows. The mesh quality is reported in three quality measures, the «tetrahedronal aspect ratio», «triangular face angle», and «tetrahedronal dihedral angle». Each quality measure is presented in a histogram.

```
Statistics:

Input points: 19202
Input facets: 38400
```

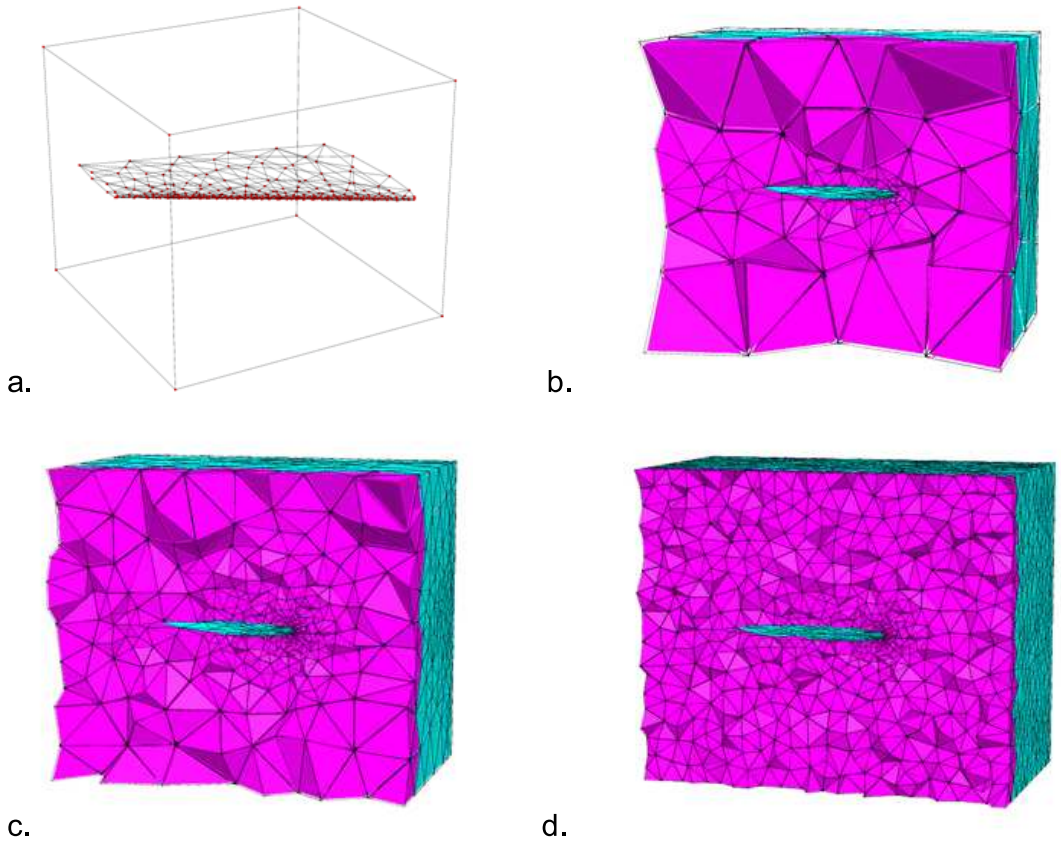


Figure 5.17. Quality control of volume mesh. a. — wing of an airplane enclosed in box; b. — mesh with radius-edge ratio bounded below 2.0; c. — mesh with radius-edge ratio bounded below 1.2; d. — mesh with radius-edge ratio bounded below 2.0 and maximum volume bounded (0.0001).

```
Input holes: 0
Input regions: 0

Mesh points: 72582
Mesh tetrahedra: 298490
Mesh faces: 642353
Mesh subfaces: 93078
Mesh subsegments: 58142

Mesh quality statistics:

Smallest volume:  2.6358e-06| Largest volume:    4.0253
```

```
Shortest edge:      0.014807| Longest edge:      4.9446
Smallest dihedral:    5.0016| Largest dihedral:172.7122

Radius-edge ratio histogram:
       < 0.707:  9717| 1.6 - 1.8:13561
 0.707 - 1    :137380| 1.8 - 2  : 4438
     1 - 1.1  : 25801|   2 - 2.5:   27
   1.1 - 1.2  : 24218| 2.5 - 3  :   13
   1.2 - 1.4  : 49987|   3 - 10 :    5
   1.4 - 1.6  : 33343|   10 -    :    0
(A tetrahedron's radius-edge ratio is its radius of
circumsphere divided by its shortest edge length)

Aspect ratio histogram:
1.1547 - 1.5:     0|      15 - 25    :2382
   1.5 - 2  :     0|      25 - 50    : 286
     2 - 2.5:     7|      50 - 100   :   0
   2.5 - 3  : 14905|     100 - 300   :   0
     3 - 4  :104588|      300 - 1000  :   0
     4 - 6  :138765|     1000 - 10000 :   0
     6 - 10 : 32111|    10000 - 100000:   0
    10 - 15 :  5446|     100000 -       :   0
(A tetrahedron's aspect ratio is its longest edge
length divided by the diameter of its inscribed sphere)

Dihedral angle histogram:
     0 -  5 degrees:    0|  90 - 100 degrees:65684
     5 - 10 degrees: 3598| 100 - 110 degrees:67134
    10 - 30 degrees:66772| 110 - 120 degrees:53257
    30 - 40 degrees:70565| 120 - 130 degrees:30221
    40 - 50 degrees:79999| 130 - 140 degrees:17502
    50 - 60 degrees:67168| 140 - 150 degrees:10387
    60 - 70 degrees:10388| 150 - 170 degrees:10267
    70 - 80 degrees: 3990| 170 - 175 degrees:  226
```

```
80 - 90 degrees:39822| 175 - 180 degrees:     0
```

The quality of the mesh created by the parallel generator should not be much lower than quality of the mesh constructed sequentially. Below, the comparison of different mesh quality measures is given in case of tetrahedral mesh of the bearing cap constructed sequentially and by developed parallel grid generator (see Fig. 5.18).

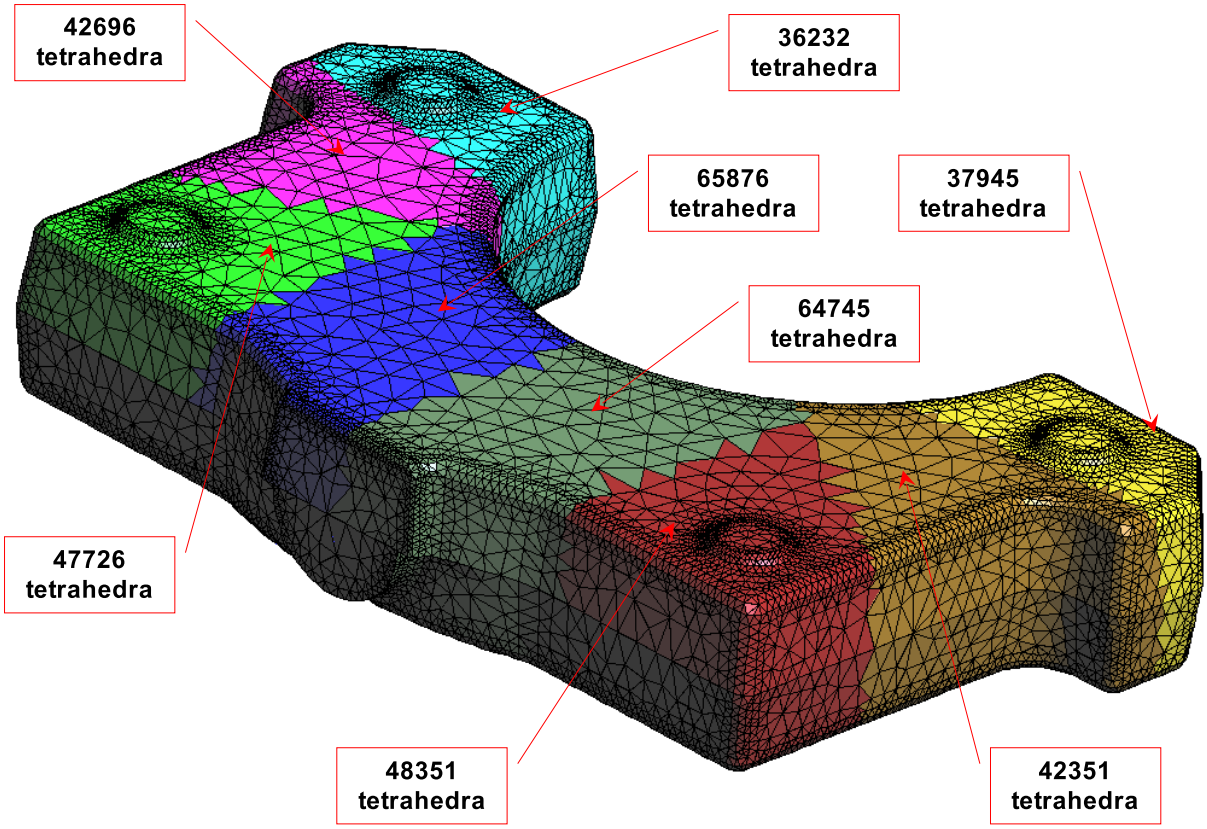


Figure 5.18. Result of decomposition and parallel volume mesh generation on 8 CPUs (shown in different colors). The total number of elements is 430074 .

In Fig. 5.19 - 5.21 comparison of various qualitative measures in normal and accumulated distribution forms are illustrated such as: minimum and maximum dihedral angle, minimum and maximum edge, elements volume, and shape. Where the shape quality of tetrahedra is defined as:

$$Q = \frac{6\sqrt{2} \cdot Vol}{\sum_{i=1}^{6} \ell_i^3},$$

where Vol – tetrahedron volume, ℓ_i — length of its edge i .

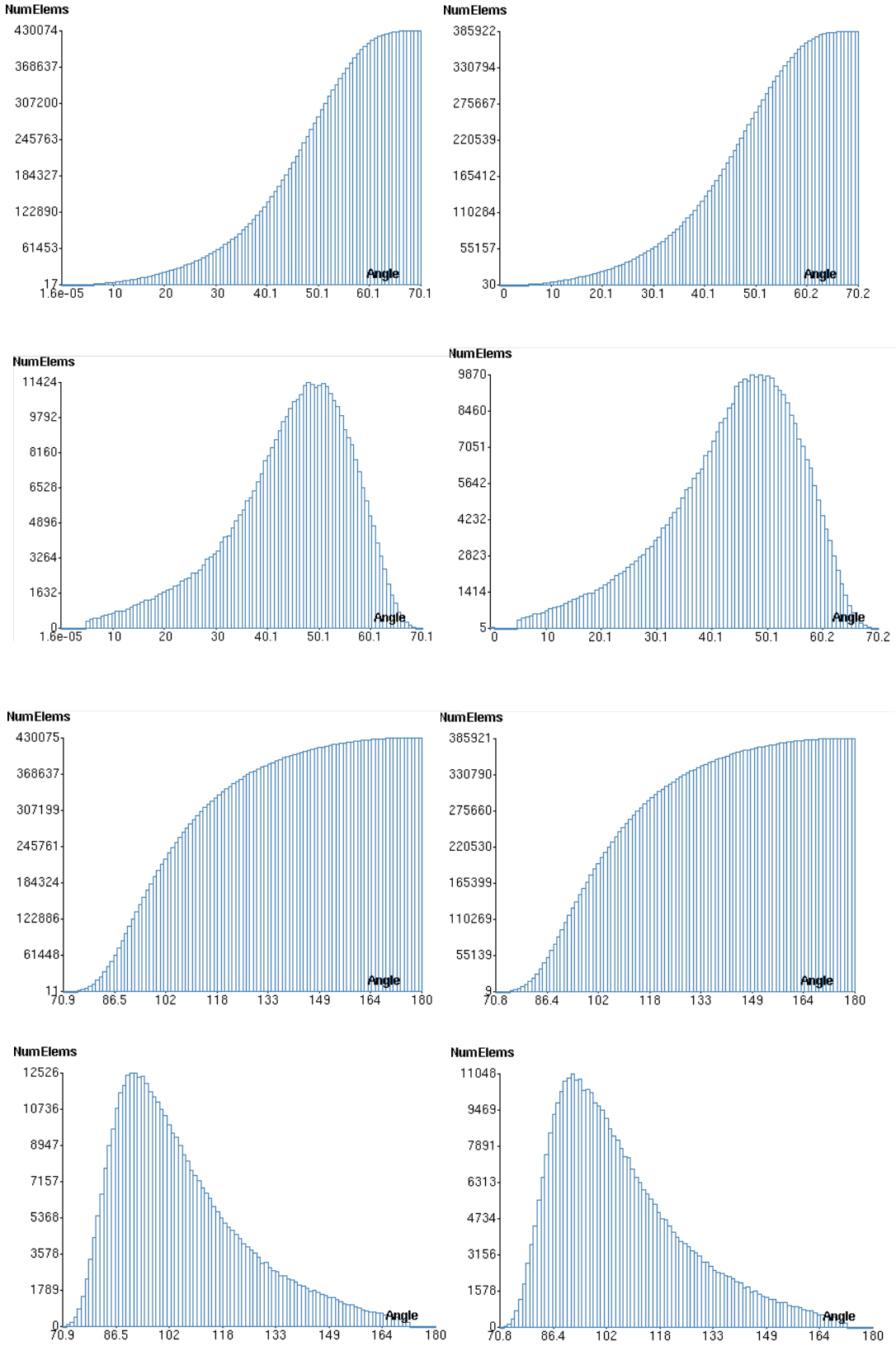


Figure 5.19. Minimum dihedral angle of tetrahedral mesh elements in accumulated (first row) and normal (second row) distribution form for meshes constructed sequentially (left) and by parallel mesh generator (right); Maximum dihedral angle of tetrahedral mesh elements in accumulated (third row) and normal (fourth row) distribution form for meshes constructed sequentially (left) and by parallel mesh generator (right).

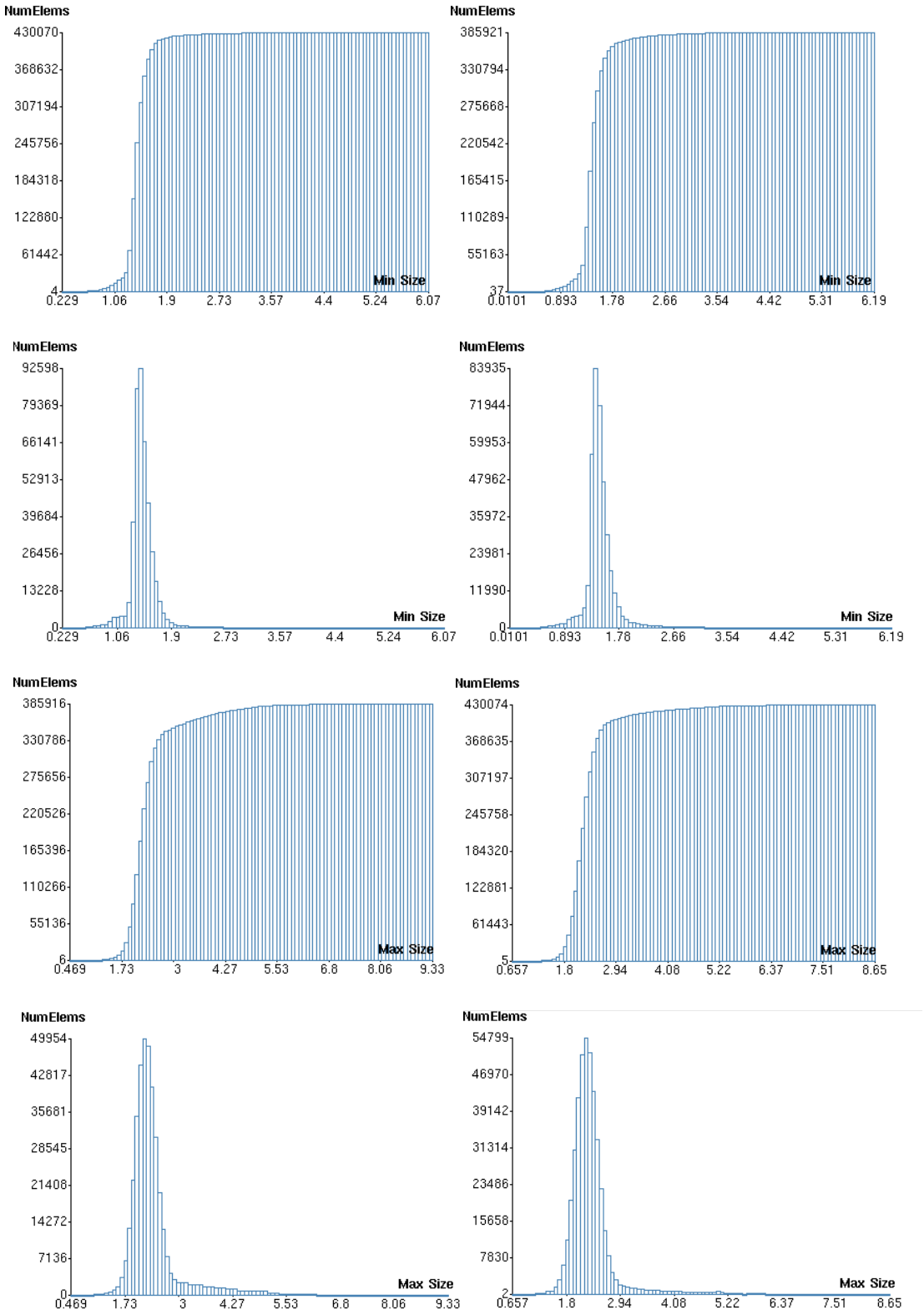


Figure 5.20. Minimum edge of tetrahedral mesh elements in accumulated (first row) and normal (second row) distribution form for meshes constructed sequentially (left) and by parallel mesh generator (right); Maximum edge of tetrahedral mesh elements in accumulated (third row) and normal (fourth row) distribution form for meshes constructed sequentially (left) and by parallel mesh generator (right).

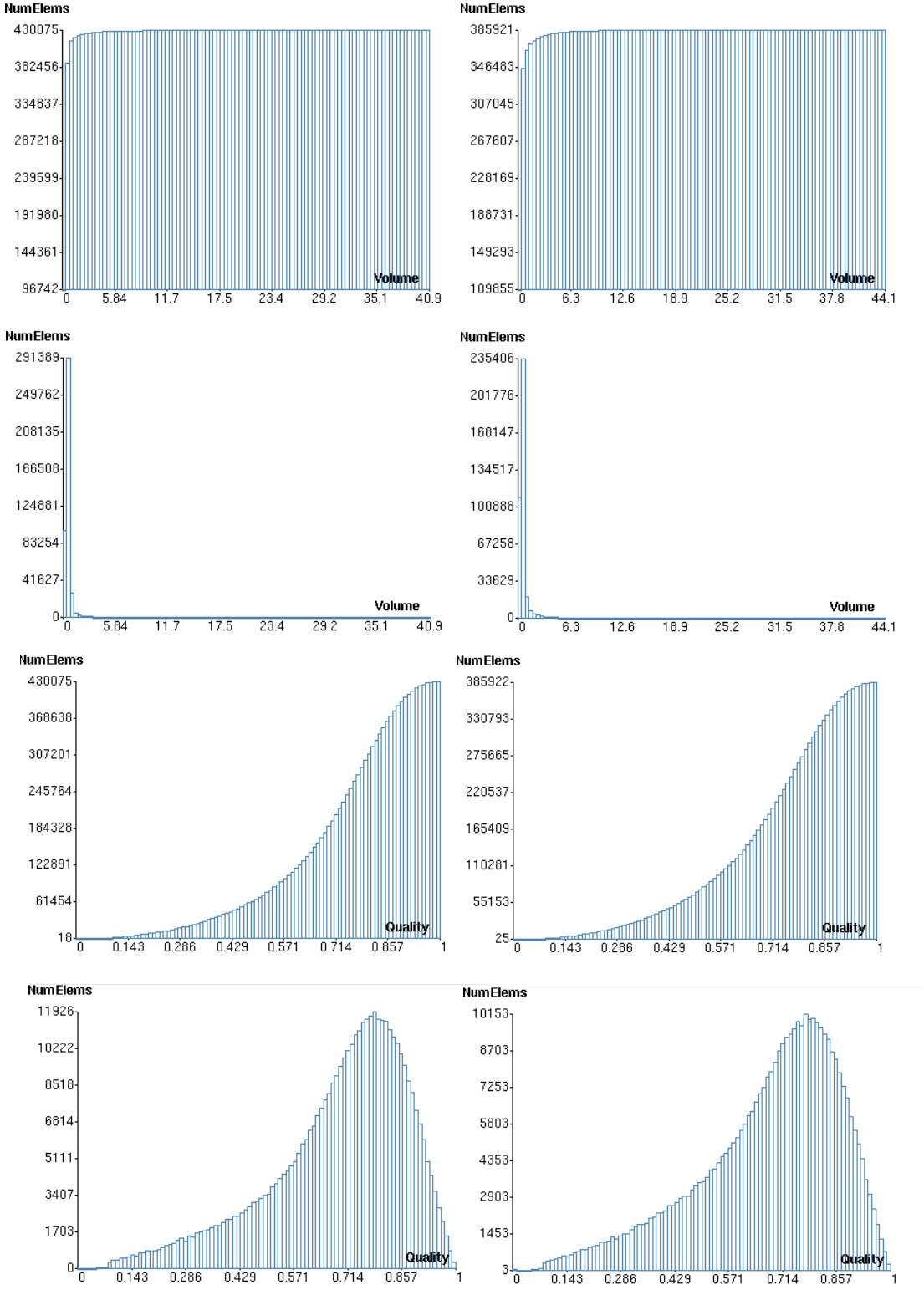


Figure 5.21. Volume of tetrahedral mesh elements in accumulated (first row) and normal (second row) distribution form for meshes constructed sequentially (left) and by parallel mesh generator (right); Shape quality of tetrahedral mesh elements in accumulated (third row) and normal (fourth row) distribution form for meshes constructed sequentially (left) and by parallel mesh generator (right).

For the regular tetrahedron $Q = 1$. With deterioration of volume elements the coefficient value becomes smaller. It can be seen that comparable quality is achieved. In both cases the greatest number of tetrahedra are between $Q = 0.76$ and $Q = 0.78$.

5.4. Computational costs and time reduction

Obviously, in comparison with traditional sequential Delaunay triangulators, the developed parallel generator has two very important advantages: the time required to construct a mesh is much less and memory requirements for each processor are lower. It enables us to work with large meshes. Manipulations with these meshes is not possible in case of traditional mesh generators.

Compare to a posteriori algorithms used by mesh partitioning libraries such as METIS, CHACO, JOSTLE, the developed parallel generator is also advantageous in terms of memory and time, since it first decompose the structure and then construct the volume mesh, while a posteriori methods perform decomposition of already created volume mesh.

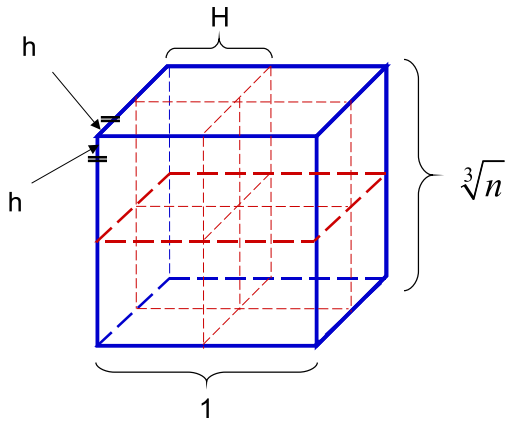
5.4.1. Superlinear scaling effect

TetGen uses the flip-based algorithm. It is incremental and adds points in a sequence of flips. The algorithm has complexity $O(n^2)$ in the worst case. However, such case will rarely happen. In practice, this algorithm has a nearly linear complexity $O(n \log n)$.

For some cases, the following interesting fact can be observe. The speed up of computational time has a superlinear scaling (see chapter III, construction of volume mesh for cylinder). In our case this can be explained in terms of complexity.

If decomposition of object is performed in a balanced way and complexity of calculation associated with volume mesh construction inside is higher than $O(n)$ (in our case it can be $O(n \log n)$) then decomposition, say into N parts, will give you acceleration of time:

$$\frac{n \log n}{\frac{n}{N} \log(\frac{n}{N})} = N \frac{1}{1 - \frac{\log N}{\log n}}.$$



H : *diameter of subdomain*
h : *mesh parameter*
n: *number of nodes in the volume*
N: *number of subdomains*
m: *number of interface nodes*

$$n = \frac{1}{h^3}, \qquad N = \frac{1}{H^3}$$

$$m = 3\left(\sqrt[3]{n}\right)^2\left(\sqrt[3]{N} - 1\right)$$

worst case: $O\left(n^2\right)$
in practice: $O\left(n \log n\right)$

Figure 5.22. Example of cube object for estimation of computational time and total area of interfaces.

The second multiplier here is greater than unit, thus division in N parts could result in time acceleration more than N times for N CPUs. Nevertheless, in practice one can not observe this due to many reasons: not perfectly balanced partitioning, different complexity depending on shape of the object and so on. So for real practical problems one can notice this for small CPU numbers (see speed up in case of two CPUs in case of bearing cap).

Let us consider a cube as an example of computational domain (Fig. 5.22). Knowing algorithm complexity and decomposition principles it is interesting to investigate how the superlinear effect depends on number of subdomains and number of grid points. In Fig. 5.23 the difference between superlinear and linear acceleration Δt and its dependence on number of subdomains and number of nodes is shown.

$$\Delta t(n, N) = N(\log n / \log(n/N) - 1)$$

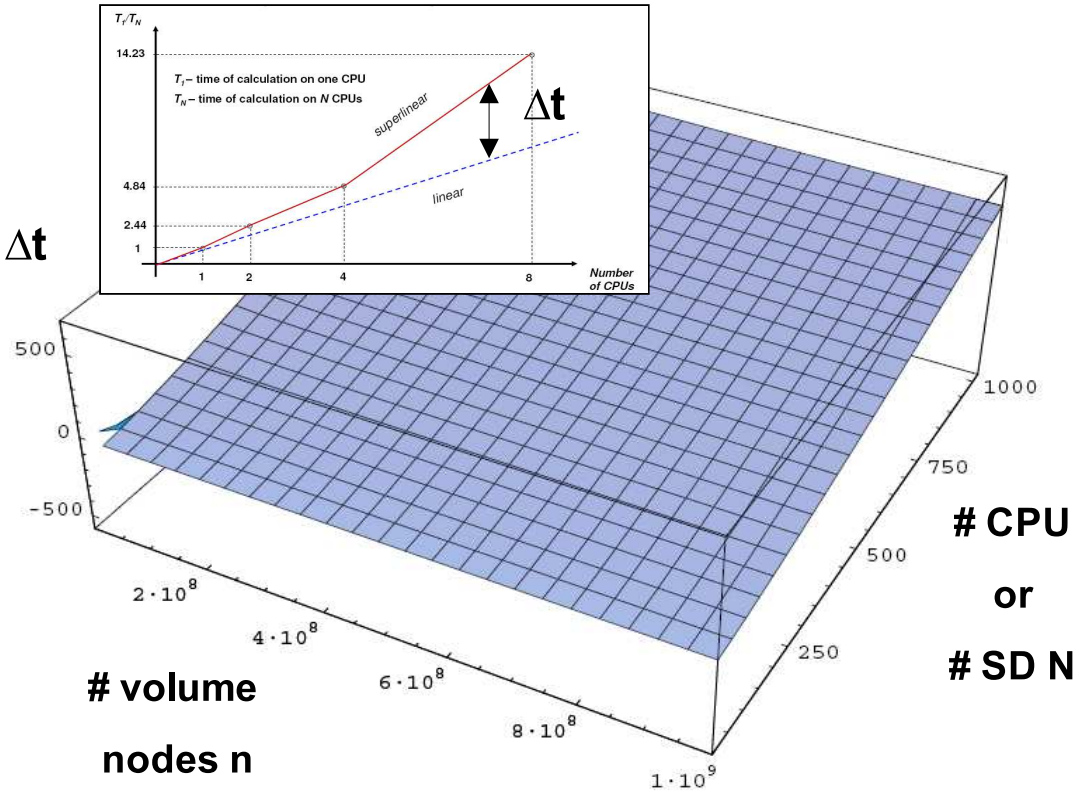


Figure 5.23. Superlinear scaling effect. Difference Δt between superlinear and linear accelerations.

5.5. Estimation of total area of interfaces

The total area of interfaces have an undoubted importance for further use by parallel solver. It is necessary to minimize the total area of interfaces, since it is directly related with amount of data transfered while problem solution procedure, and, therefore influences speed of calculations.

Using the cube object as an example, let us try to estimate the change of total interface area for different numbers of subdomains and grid points. In Fig. 5.24 dependence graph of ratio f between interface nodes and total number of grid points is shown.

$$f(n, N) = 3n^{1/3}(N^{1/3} - 1).$$

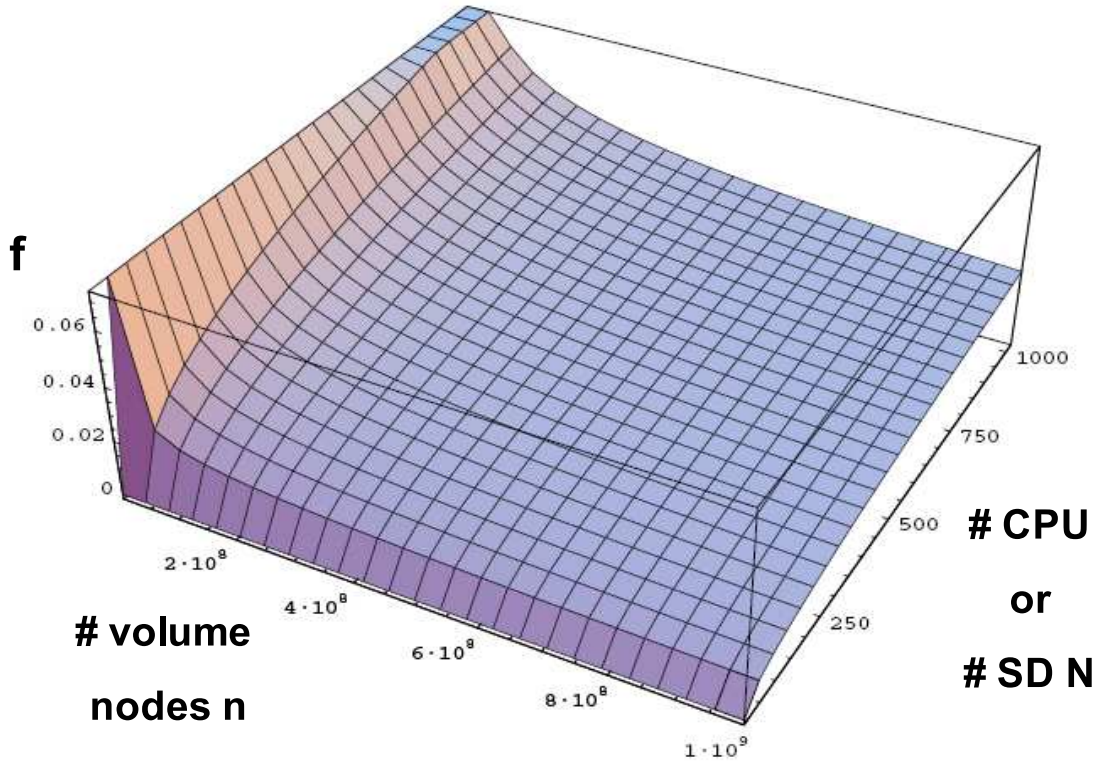


Figure 5.24. Ratio f between interface nodes and total number of grid points.

It is clear that we want to keep this coefficient small. It seems that sometimes it is necessary to decrease number of CPUs in order to reduce the total interface area, but one should always keep in mind that it might not be possible due to the memory limitations reasons.

5.6. Advantages of the developed parallel grid generator

As a result of analysis of developed algorithm and of computational results of real-life problems it is necessary to point out major advantages of developed parallel grid generator compare to other known decomposition and parallel grid generation algorithms.

- Algorithm needs lower computational time and lower memory compared to an a posteriori partitioning method (PARMETIS, CHACO, JOSTLE).

- Algorithm enables us to use well tested and fine-tuned sequential 2D and 3D triangulators.
- Algorithm achieves 100% code re-use and eliminates need communication and synchronization.
- All operations performed in algorithm preserve original surface mesh.
- Algorithm produces almost plane interfaces, i.e. interfaces with small perturbations, between the subdomains.
- Algorithm uses a splitting criterion, which minimizes the interface area and which is sensitive to both the object shape and the grid resolution.

5.7. Prospective directions of further development

There are many tasks that could be mentioned here, but the most important directions of further parallel grid generator development are the following:

- Decomposition of more complex geometrical domains.
- To ensure a priori guarantee of domain size, mesh size, and a quality criterion.
- Improving of load balancing and scalability of parallel generation algorithm.
- Improve programming interface with CAD and FE solvers.
- Perform local adaptive mesh refinement from CAD level based in information provided by solver.

Summary and conclusion

1. Novel algorithm for automatic parallel generation of unstructured three-dimensional mesh by using geometrical decomposition of computational domain has been developed. Proposed algorithm preserves original surface mesh and produces almost plane interfaces, i.e. interfaces with small perturbations, between the subdomains.

2. Different domain decomposition approaches (a posteriori, a priori, along the same direction, recursive, overdecomposition) and criteria (equidistant cutting, volume comparison, moment of inertia) have been investigated. It has been shown that from examined methods recursive cutting by plane according to moment of inertia criterion gives better result since the approach minimizes interface area and is sensitive to both object shape and grid resolution.

3. A software package for parallel grid generation has been created based upon developed algorithm. It includes well tested and fine-tuned sequential 2D and 3D triangulators, achieves 100% code re-use and eliminates need for communication and synchronization. The software is fully automatic and can be used together with FE/FV solvers to overcome memory limitations as well as to reduce computational time.

4. Different surface and volume mesh quality measures comparison have been given for sequentially generated meshes and meshes constructed by the parallel grid generator. It has been demonstrated that comparable quality is achieved, mesh generation time is reduced significantly due to developed parallel grid generator. Efficiency analysis of parallel algorithm and software shows that they are efficient (70% for bearing cap mesh of 40 millions elements on 32 CPUs).

5. Developed algorithm and software were used for solution of real-life problems: construction of mesh for femoral and tibial knee prosthesis components, bearing cap to fix a crank shaft at a motorblock and others. Size

of generated meshes achieved order of $10^7 - 10^8$ elements. Sequential construction of meshes of this size is extremely costly.

Bibliography

1. Thacker W.C. A brief review of techniques for geneational grids // Int. J. Numer. Meth. Engng. - 1980. - V. 15. - №9. - P. 1335–1341.
2. Ho-Le. Finite element mesh generation methods: a review and classification. // Computer Aided Design. - 1988. - V. 20. - P. 27–38.
3. Shephard M.S., Grice K.R., Lot J.A., Schroeder W.J. Trends in automatic three-dimensional mesh generation. // Comput. Strict. - 1988. - V. 30. - №1/2 - P. 421–429.
4. Baker T.J. Prospects and expectations for unstructured methods. // Proceedings of the Surface Modeling, Grid Generation and Related Issues in Computational Fluid Dynamics Workshop. NASA conference publication 3291, NASA Lewis Research Center, Cleveland, OH - 1995. - P. 273–287.
5. Baker T.J. Mesh adaptation strategies for problems in fluid dynamics. // Finite Elements Anal. Design - 1995. - V. 25. – P. 243–273.
6. Field D.A. The legacy of automatic mesh generation from solid modeling. // Comp. Aided Geom. Design - 1995. - V. 12. – P. 651–673.
7. Carey G.F. Computational Grids. Generation, Adaptation, and Solution Strategies, Taylor and Francis, London, 1997.
8. George P.L., Borouchaki H. Delaunay Triangulation and Meshing - Editions Hermes, Paris, 1992.
9. Krugljakova L.V., Neledova A.V., Tishkin V.F., Filatov A.Yu. Unstructured adaptive grids for problems of mathematical physics. // Math. Modeling. - 1998. - V. 10. - №3 - P. 93–116.
10. Thompson J.F., Weatherill N.P. Aspects of numerical grid generation: current science and art. // AIAA Paper 93-3539 - 1993.
11. Skvortsov A.V. Obzor algoritmov postroeniya Delone // Vychislitelnye Metody i programmirovanie - 2002. - V. 3. - P. 14–39.

12. Dirichlet G.L. Über die Reduktion der positiven quadratischen Formen mit drei unterstimmten ganzen Zahlen. // Z. Angew Math. Mech - 1850. - V. 40. - №3 - P. 209–227.

13. Voronoi G. Nouvelles applications des paramétres continus á la théorie des formes quadratiques. Recherches sur les parallélloédres primitifs // Journal Reine angew. Math.- 1908. - V. 134.

14. Green P., Sibson R. Computing Dirichlet tesselation in the plane. // Comput. Journal - 1978. - V. 21. - №3 - P. 168–173.

15. Lawson C.L. Software for C^1 surface interpolation. // Math. Soft., 3, J. Rice ed., Academic Press, New York. - 1977.

16. Hermeline F. Une méthode automatique de maillage en dimension n. // Thése, Université Paris VI, Paris.

17. Bowyer A. Computing Dirichlet tesselation // The Comp. J. - 1981. - V. 24. - №2 - P. 162–167.

18. Watson D.F. Computing the n-dimensional Delaunay Tesselation with applications to Voronoi polytopes // Computer Journal - 1981. - V. 24. - №2 - P. 167–172.

19. Avis D., Bhattacharya B.K. Algorithms for computing d-dimensional Voronoi diagrams and their duals, // Advances in computing research - 1983. - V.1. - P. 159–188.

20. Schoenhardt E. Über die Zerlegung von Dreieckspolyedern in Tetraeder // Mathematische Annalen - 1928. - V.98. - P. 309-312.

21. Bagemihl F. On Indecomposable Polyhedra // American Mathematical Monthly - 1948. - V.55. - P. 411-413.

22. Chazelle B. Convex Partition of Polyhedra: A Lower Bound and Worst-case Optimal Algorithm // SIAM Journal on Computing - 1984. - V.13. - №3. - P. 488-507.

23. Rambau J. On a Generalization of Schoenhardt's Polyhedron // MSRI Preprint - 2003. - V.13.

24. Ruppert J., Seidel R. On the difficulty of triangulating three-dimensional non-convex polyhedra // Discrete and Computational Geometry - 1992. - V.7. - P. 227-254.

25. Shewchuk J.R. Constrained Delaunay tetrahedralizations and provably good boundary recovery // 11th International Meshing Roundtable - 2002. - P. 193-204.

26. Pébay P. A Priori Delaunay-Conformity // 7th International Meshing Roundtable - 1998.

27. Murphy M., Mount D.M., Gable C.W. A point-placement strategy for conforming Delaunay tetrahedralization // Proceedings of the 11th Annual Simposium on Discrete Algorithms - 2002. - P. 67-74.

28. Cohen-Steiner D., de Verdiére E. Yvinec M. Conforming Delaunay triangulations in 3D // Proceedings of the 18th Annual Simposium on Computational Geometry - 2002.

29. Si H., Gaertner K. An algorithm for three-dimensional constrained Delaunay Tetrahedralizations // Proceedings of the 4th International Conference on Engineering Computational Technology - 2004.

30. Liseikin V.D. Grid Generation Methods - Springer-Verlag Berlin Heidelberg - 1999.

31. Avis C.L. Properties of n-dimensional triangulations. // Comp. Aided Geom. Design - 1986. - V.2. - P. 231–246.

32. Henle M. A combinatorial Introduction to Topology. - W.H. Freeman, San Francisco.

33. Steinitz E. Polyeder and Raumeinteilungen. // Enzykl. Mathematischen Wiss. - 1922. - V.3. - P. 163.

34. Klee V. The number of vertices of a convex polytope. // Can. Math - 1964. - V.16. - P. 37.

35. Lee K.D. On finding k-nearest neighbours in the plane. // Tech. Report 76-2216. University of Illinois, Urbana, IL - 1976.

36. Delaunay B.N. Sur la sphere vide. // Bull. Acad. Sci. USSR VII: Class. Sci. Mat. Nat. - 1934. - V.3. - P. 793-800.

37. Delaunay B.N. Peterburg School of Number Theory. // Acad. Sci. USSR, Moscow - 1947.

38. Brostow W., Dussault J.P., Fox B.L. Construction of Voronoi polyhedra. // J. Comput. Phys. - 1978. - V.29. - P. 81-92.

39. Finney J.L. A procedure for the construction of Voronoi polyhedra. // J. Comput. Phys. - 1979. - V.32. - P. 137-143.

40. Tanemura M., Ogawa T., Ogita N. A new algorithm for three-dimensional Voronoi tesselation. // J. Comput. Phys. - 1983. - V.51. - P. 191-207.

41. Sloan S.W., Houlsby G.T. An implementation of Watson's algorithm for computing 2D Delaunay triangulations. // Advances Engng. Software - 1984. - V.6. - №4 - P. 192-197.

42. Fortune S. A sweepline algorithm for Voronoi diagrams. // AT&T Bell Laboratory Report, Murray Hill, NJ - 1984.

43. Zhou J.M., Ke-Ran, Ke-Ding, Quing-Hua. Computing constrained triangulations and Delaunay triangulation: a new algorithm. // IEEE Transactions on Magnetics - 1990 - V.26. - №2 - P. 692-694.

44. Edelsbrunner H. Algorithms in combinatorial geometry. - 1987. - Springer, Berlin, Heidelberg.

45. Du D.-Z., Hwang F. Computing in Euclidian Geometry. - 1995. - World scientific, Singapore.

46. Okabe A., Boots B., Sugihara K. Spatial Tesselations Concepts and Applications of Voronoi Diagrams. - 1992. - Wiley, New York.

47. Preparata F.P., Shamos M.I. Computational Geometry: An Introduction. - 1995. - Springer, New York.

48. Guibas L., Stolfi J. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. // ACM Trans. Graphics - 1985 - V.4 - P. 74-123

49. Baker T.J. Three-dimensional mesh generation by triangulation of arbitrary point sets. // 1987. - AIAA Paper 87-1124-CP.

50. Baker T.J. Automatic mesh generation for complex three-dimensional region using a constrained Delaunay triangulation. // Eng. Comput. - 1989 - V.5. - P. 161-175.

51. Sibson R. Locally equiangular triangulations. // Comput. J. - 1978. - V.21. - №3 - P. 243-245.

52. Lee B.E., Schachter B.J. Two algorithms for constructing a Delaunay triangulation. // Int. J. Comput. Inform. Sci. - 1980 - V.9. - №3 - P. 219-241.

53. Holmes D.J., Snyder D.D. The generation of unstructured triangular meshes using Delaunay triangulation. // In Sengupta S., Hauser J., Eiseman P.R., Thompson J.F. (eds.) Numerical Grid Generation in Computational Fluid Dynamics. - 1988 - P. 643-652.

54. Ruppert J. Results on Triangulation and High Quality Mesh Generation. // PhD thesis, University of California, Berkeley. - 1992.

55. Chew P. Mesh generation, curved surfaces and guaranteed quality triangles. // Technical Report, IMA, Workshop on Modeling, Mesh Generation and Adaptive Numerical Methods for Partial Differential Equations, University of Minnesota, Minneapolis. - 1993.

56. Rebay S. Efficient unstructured mesh generation by means of Delaunay triangulation and Bowyer-Watson algorithm. // J. Comput. Phys. - 1993 - V.106. - P. 125-138.

57. Baker T.J. Triangulations, mesh generation and point placement strategies. // In Caughey D. (ed.). Computing the future. Wiley. New York. - 1994. - P. 1-15.

58. Haman B., Chen J.-L., Hong G. Automatic generation of unstructured volume grids inside or outside closed surfaces. // In Weatherill N.P., Hauser J., Eiseman P.R., Thompson J.F. (eds.) Numerical Grid Generation in Computational Field Simulation and Related Fields. Pineridge, Swansea - 1994 - P.187

59. Anderson W.K. A grid generation and flow solution method for the Euler equations in unstructured grids. // J. Comput. Phys. - 1994 - V.110. - P. 23-38.

60. Lee D.T., Lin A.K. Generalized Delaunay triangulation for planar graphs // Discrete Comput. Geometry - 1986 - V.1. - P. 201-217.

61. Chew L.P. Constrained Delaunay triangulation. // Algorithmica - 1989 - V.4. - P. 97-108.

62. Cline A.K., Renka R.L. A constrained two-dimensional triangulation and the solution of closest node problems in the presence of barriers // SIAM J. Numer. Anal. - 1990 - P. 1305-1321.

63. George P.L., Hecht F., Saltel E. Automatic 3D mesh generation with prescribed meshed boundaries // IEEE Trans. Magn. - 1990 - V.26 - № 2 - P. 771-774.

64. Weatherill N.P. The integrity of geometrical boundaries in the two-dimensional Delaunay triangulation // Commun. Appl. Numer. Meth. Fluids - 1990. - V.8 - P. 101-109.

65. George P.L., Hermeline F. Delaunay's mesh of convex polyhedron in dimension d: application for arbitrary polyhedra // Int. J. Numer. Meth. Engng. - 1992. - V.33 - № 2 - P. 975-995.

66. Field D.A., Nehl T.W. Stitching together tetrahedral meshes // In Field D.A., Komkov V.(eds.): Geometric Aspects of Industrial Design. SIAM Philadelphia, Chapter 3 - 1992. - P. 25-38.

67. Hazlewood C. Approximating constrained tetrahedrizaton // Comput. Aided Geometric Design - 1993. - V.10 - P. 67-87.

68. Weatherill N.P., Hassan O. Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints // Int. J. Numer. Meth. Engng. - 1994. - V.37 - P. 2005-2039.

69. Cavendish J.C., Field D.A., Frey W.H. An approach to automatic three-dimensional finite element mesh generation // Int. J. Numer. Meth. Engng. - 1992. - V.21 - P. 329-347.

70. Shenton D.N., Cendes Z.J. Three-dimensional finite element mesh generation using Delaunay tessellation // IEEE Trans. Magnetics MAG-21 - 1985. - P. 2535-2538.

71. Perronet A. A generator of tetrahedral finite elements for multimaterial objects or fluids // In Sengupta S., Hauser J., Eiseman P.R., Thompson J.F. (eds.) Numerical Grid Generation in Computational Fluid Dynamics. - 1988 - P. 719-728.

72. Jameson A., Baker T.J., Weatherill N.P. Calculation of inviscid transonic flow over a complete aircraft // AIAA Paper 86-0103 - 1992.

73. Weatherill N.P. A method for generating irregular computational grids in multiply connected planar domains // Int. J. Numer. Meth. Fluids - V.8. - 1988 - P. 181-197.

74. DeFloriani L. Surface representations on triangular grids // The Visual Computer - V.3. - 1987 - P. 27-50.

75. Yerry M.A., Shephard M.S. Automatic three-dimensional mesh generation for three-dimensional solids // Comput. Struct. - V.20. - 1985 - P. 31-39.

76. Yerry M.A., Shephard M.S. Automatic three-dimensional mesh generation by the modified-octree technique // Int. J. Numer. Meth. Engng. - V.20. - 1990 - P. 1965-1990.

77. Lohner R. Some useful data structures for the generation of unstructured grids // Commun. Appl. Num. Meth. - V.4. - 1988 - P. 123-135.

78. Shephard M.S., Guerinoni F., Flaherty J.E., Ludwig R.A., Bachmann P.L. Finite octree mesh generation for automated adaptive three-dimensional flow analysis // In Sengupta S., Hauser J., Eiseman P.R., Thompson J.F. (eds.) Numerical Grid Generation in Computational Fluid Dynamics. - 1988 - P. 709-718.

79. Schneiders R., Bunten R. Automatic generation of hexahedral finite element meshes // Commun. Appl. Num. Meth. - V.12. - 1995 - P. 693.

80. Peraire J., Vahdati M., Morgan H., Zienkiewicz O.C. Adaptive remeshing for compressible flow computations // AIAA Paper 88-0032 - 1987.

81. Lohner R. Generation of three-dimensional unstructured grids by the advancing fron method // AIAA Paper 88-0515 - 1988.

82. Formaggia L. An unstructured mesh generation algorithm for three-dimensional aeronautical configurations // In Arcilla A.S., Hauser J., Eiseman P.R., Thompson J.F. (eds.) Numerical Grid Generation in Computational Fluid Dynamics and Related Fields. North-Holland, New York - 1991 - P. 249-260.

83. Merriam M. An efficient advancing front algorithm for Delaunay triangulation // Technical Report, AIAA Paper 91-0792 - 1991.

84. Mavriplis D.J. Unstructured and adaptive mesh generation for high Reynolds number viscous flow // In Arcilla A.S., Hauser J., Eiseman P.R., Thompson J.F. (eds.) Numerical Grid Generation in Computational Fluid Dynamics and Related Fields. North-Holland, Amsterdam - 1991 - P. 79-92.

85. Mavriplis D.J. An advancing front Delaunay triangulation algorithm designed for robustness // AIAA Paper 93-0671 - 1993.

86. Muller J.D., Roe P.L., Deconinck H. A frontal approach for internal node generation in Delaunay triangulations // Int. J. Numer. Meth. Fluids - 1993. - V.17. - № 3. - P. 241-256.

87. Markum D.L., Weatherill N.P. Unstructured grid generation using iterative point insertion and local reconnection // AIAA Journal - 1995. - V.33. - № 9. - P. 1619-1625.

88. Lohner R. Matching semi-structured and unstructured grids for Navier-Stokes calculations // AIAA Paper 933348-CP - 1993.

89. Pirzadeh S. Recent progress in unstructured grid generation // AIAA Paper 92-0445 - 1992.

90. Parthasarathy V., Kallinderis Y. Directional viscous multigrid using adaptive prismatic meshes // AIAA Journal - 1995. - V.33. - № 1. - P. 69-78.

91. Peraire J., Peiro J., Formaggia L., Morgan K., Zienkiewicz O.C. Finite element Euler computations in three dimensions // AIAA Paper 88-0033 - 1988.

92. Lohner R., Parikh P. 3-dimensional grid generation by the advancing front method // Int. J. Numer. Meth. Fluids - 1995. - V.8. - P. 1135-1149.

93. Weatherill N.P., Marchant M.F., Hassan O., Marcum D.L. Adaptive inviscid flow solutions for aerospace geometries on efficiently generated unstructured tetrahedral meshes // AIAA Paper 93-3390 - 1993.

94. Powell K.G., Roe P.L., Quirk J.J. Adaptive mesh algorithms for computational fluid dynamics // In Hussaini M.Y., Kumar A., Salas M.D. (eds.): Algorithmic Trends in Computational Fluid Dynamics. Springer, New York - 1992. - P. 301-337.

95. Holmes D.G., Lamson S.H. Adaptive triangular meshes for compressible flow solutions // In Hauser J., Taylor C. (eds.): Numerical Grid Generation in Computational Fluid Dynamics. Pineridge, Swansea. - 1986. - P. 413.

96. Mavriplis D.J. Adaptive mesh generation for viscous flows using Delaunay triangulation // J. Comput. Phys. - V.90 - 1990 - P. 271-291.

97. Muller J.D. Quality estimates and stretched meshes based on Delaunay triangulation // AIAA Journal - 1994 - V.32. - P. 2372-2379.

98. Pirzadeh S. Structured background grids for generation of unstructured grids by advancing front method // AIAA Journal - V.31 - № 2 - 1993 - P. 257-265.

99. Pirzadeh S. Viscous unstructured three-dimensional grids by the advancing-layers method // AIAA Paper 94-0417 - 1994.

100. Darve E., Löhner R. Advanced Structured-Unstructured Solver for Electromagnetic Scattering from Multimaterial Objects // AIAA-97-0863 - 1997.

101. Morgan K., Brooks P.J., Hassan O., Weatherill N.P. Parallel Processing for the Simulations of Problems Involving Scattering of Electromagnetic Waves // in Proc. Symp. Advances in Computational Mechanics (Demkowicz L. and Reddy J.N. eds) - 1997.

102. Baum J.D., Luo H., Löhner R. Numerical Simulation of a Blast Inside a Boeing 747 // AIAA-93-3091 - 1993.

103. Baum J.D., Luo H., Löhner R. Numerical Simulation of a Blast in the World Trade Center // AIAA-95-0085 - 1995.

104. Jou W. Comments on the Feasibility of LES for Commercial Airplane Wings // AIAA-98-2801 - 1998.

105. Yoshimura S., Nitta H., Yagawa G., Akiba H. Parallel Automatic Mesh Generation Method of Ten-million Nodes Problem Using Fuzzy Knowledge Processing and Computational Geometry // Proc. 4th World Congress on Computational Mechanics (WCCM-IV) (CD-ROM) - 1998.

106. Mavriplis D.J., Pirzadeh S. Large-Scale Parallel Unstructured Mesh Computations for 3-D High Lift Analysis // ICASE Rep. 99-9 - 1999.

107. Löhner R., Cebral J.R. Parallel Advancing Front Grid Generation // Proceedings, 8th International Meshing Roundtable - 1999 - P. 67-74.

108. Löhner R. Three-Dimensional Fluid-Structure Interaction Using a Finite Element Solver and Adaptive Remeshing // Comp. Sys. In Eng. - 1990 - V.1. - № 2 - P. 257-272.

109. Mestreau E., Löhner R., S. Aita TGV Tunnel-Entry Simulations Using a Finite Element Code with Automatic Remeshing // AIAA-96-0798 - 1996.

110. Mestreau E., Löhner R. Airbag Simulations Using Fluid/Structure Coupling // AIAA-96-0798 - 1996.

111. Baum J.D., Luo H., Löhner R., Yang C.,Pelessone D., Charman C. A Coupled Fluid/Structure Modeling of Shock Interaction with a Truck // AIAA-96-0795 - 1996.

112. Kamoulakos A., Chen V., Mestreau E., Löhner R., Finite Element Modeling of Fluid/Structure Interaction in Explosively Loaded Aircraft Fuselage Panels Using PAMSHOCK/PAMFLOW Coupling // Conf. on Spacecraft Structures, Materials and Mechanical Testing - 1996.

113. Löhner R., Yang C., Cebral J., Baum J.D., Luo H., Pelessone D., Charman C. Fluid-Structure-Thermal Interaction Using a Loose Coupling Algorithm and Adaptive Unstructured Grids // AIAA-98-2419 - 1998.

114. Hassan O., Bayne L.B., Morgan K., Weatherill N.P. An Adaptive Unstructured Mesh Method for Transient Flows Involving Moving Boundaries // 5th US Congress on Computational Mechanics - 1999 - P. 662-674.

115. Baum J.D., Luo H., Löhner R. The Numerical Simulation of Strongly Unsteady Flows With Hundreds of Moving Bodies // AIAA-98-0788 - 1998.

116. Baum J.D., Luo H., Mestreau E., Löhner R., Pelessone D., Charman C. A coupled CFD/CSD Methodology for Modeling Weapon Detonation and Fragmentation // AIAA-99-0794 - 1999.

117. Simon H. Partitioning of unstructured problems for parallel processing // Comp. Systems in Eng. - 1991 - V.2. - P. 135-148.

118. Farhat C., Lesoinne M. Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics // Int. J. Numer. Meth. Engng - 1993 - V.36. - P. 745-764.

119. Galtier J., George P.L. Prepartitioning as a Way to Mesh Subdomains in Parallel // 5th International Meshing Roundtable, Sandia National Laboratories - 1996 - P. 107-122.

120. Verhoeven N.A., Weatherill N.P., Morgan K. Dynamic load balancing in a 2D parallel Delaunay mesh generator // Proceedings of the Parallel CFD Conference - 1995.

121. Topping B.H.V., Cheng B. Parallel and distributed adaptive quadrilateral mesh generation // Computers and Structures - 1999 - V.73. - P. 519-536.

122. Laemer L., Burghardt M. Parallel generation of triangular and quadrilateral meshes // Advances in Engineering Software - 2000 - V.31. - P. 929-936.

123. Löhner R., Camberos J., Merriam M. Parallel Unstructured Grid Generation // Comp. Meth. Appl. Mech. Eng. - 1992 - V.95. - P. 343-357.

124. Chew L.P., Chrisochoides N., Sukup F. Parallel Constrained Delaunay Meshing // Proc. Workshop on Trends in Unstructured Mesh Generation - 1997.

125. Chrisochoides N., Nave D. Simultaneous mesh generation and partitioning for Delaunay meshes // Mathematics and Computers in Simulation - 2000 - V.54. - P. 321-339.

126. Okunsanya T., Peraire P. 3-D Parallel Unstructured Mesh Generation // Proc. Joint ASME/ASCE/SES Summer Meeting - 1997.

127. Löhner R. Three-Dimensional Parallel Unstructured Grid Generation // Int. J. Num. Meth. Eng. - 1995 - V.38. - P. 905-925.

128. Said R., Weatherill N.P., Morgan K.,Verhoeven N.A. Distributed Parallel Delaunay Mesh Generation // Comp. Meth. Appl. Mech. Eng. - 1999 - V.177 - P. 109-125.

129. Chrisochoides N. A Survey of Parallel Mesh Generation Methods // Brown University, Providence RI - 2005.

130. Hoppe H., DeRose T., Duchamp T., McDonald J., Stuetzle W. Mesh optimization // ACM SIGGRAPH - 1993 - P.19-26.

131. Snir M., Otto S., Huss-Lederman S., Walker D., Dongarra J. MPI: Complete Reference. - London: MIT Press, 1996. - 350 p.

132. Shewchuk J. R. Triangle: Engineering a 2d quality mesh generator and Delaunay triangulator // Proceedings first workshop on Applied Computational Geometry - 1996. - P. 124-133.

133. Si H., Gaertner K. Meshing Piecewise Linear Complexes by Constrained Delaunay Tetrahedralizations // Proceedings of the 14th International Meshing Roundtable - 2005.

134. Shewchuk J. R. Tetrahedral mesh generation by Delaunay refinement // Proceeding of the 14th Annual Simposium on Computational Geometry - 1998. - P. 86-95.

135. Chew P.L. Guaranteed-Quality Delaunay meshing in 3D // Proceeding of the 13th Annual ACM Simposium on Computational Geometry - 1997. - P. 391-393.

136. Edelsbrunner H., Guoy D. An experimental Study of Sliver Exudation // Engineering with Computers - 2002. - V.18. - P. 229-240.

137. Cheng S.W., Dey T.K., Edelsbrunner H., Facello M.A., Teng S.H. Sliver Exudation // Proceeding of the 15th Annual Simposium on Computational Geometry - 1999.

138. Miller G.L., Talmor D., Teng S.-H., Walkington N., Wang H. A Delaunay Based Numerical Method for Three Dimensions: Generation, Formulation, and Partition // Proceeding of the 27th Annual ACM Simposium on the Theory of Computing - 1999. - P. 683-692.

139. METIS: Multilevel Partitioning Algorithms // http://glaros.dtc.umn.edu/gkhome/views/metis, 2007

140. PETSc: Portable, Extensible Toolkit for Scientific Computation // http://www-unix.mcs.anl.gov/petsc/petsc-2/, 2007

141. H. Andrä, D. Stoyanov Error indicators in the parallel finite element solver for linear elasticity DDFEM // Berichte des Fraunhofer ITWM №83(2006)

142. PERMAS: Numerical Simulation with Finite Elements // http://www.intes.de, 2007

143. GeoDict: Interactive Microstructure Generator // http://wwws.geodict.com, 2007

144. Lima Group Web Site http://www.lima.it/english/prostheses/products/multigen.htm, 2007

145. http://www.gienath.com/english/produkte.htm, 2007

Publications:

1. E. Ivanov, O. Gluchshenko, H. Andrä, A. Kudryavtsev, *Automatic Parallel Generation of Tetrahedral Grids by Using a Domain Decomposition Approach*, Journal of Computational Mathematics and Mathematical Physics, Distributed by **Springer**. Submitted.

2. E.G. Ivanov, *Automatic Parallel Generation of Three-Dimensional Unstructured Grids for Computational Mechanics*, Journal of Computational Technologies, Vol. 11(1), 2006, pp. 3-17.

3. Evgeny G. Ivanov, Heiko Andrä, Alexey N. Kudryavtsev, *Domain Decomposition Approach for Automatic Parallel Generation of Tetrahedral Grids*, International Mathematical Journal Computational Methods in Applied Mathematics, Vol. 6(2), 2006, pp. 178-193.

4. Evgeny Ivanov, Olga Gluchshenko, Heiko Andrä and Alexey Kudryavtsev, *Efficient parallel algorithm and software for automatic generation of 3D computational tetrahedral meshes*, The 2007 World Congress in Computer Science, Computer Engineering and Applied Computing, PDPTA'07 - International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, USA, June 25-28, 2007, pp. 250-256.

5. Evgeny G. Ivanov, Heiko Andrä, Alexey N. Kudryavtsev, *Domain Decomposition Approach for Automatic Parallel Generation of Three-Dimensional Unstructured Grids*, Proceedings of the European Conference on Computational Fluid Dynamics (ECCOMAS CFD 06), Egmond aan Zee, The Netherlands, 2006, Paper No. 295.

6. Evgeny Ivanov, Olga Gluchshenko, Heiko Andrä, Alexey Kudryavtsev, *Parallel Software Tool for Decomposing and meshing of 3D Structures*, Fraunhofer ITWM Report Nr. 110 (2007), pp. 3-17.

7. E.G. Ivanov, H. Andrä, A.N. Kudryavtsev, *Automatic Parallel Generation of Tetrahedral Grids by Using a Domain Decomposition Approach*, Proc. of the Conf. on Numerical Geometry, Grid Generation and High Performance Computing, Moscow, Russia, July 4-7, 2006, pp. 115-124.

8. Evgeny G. Ivanov, Heiko Andrä, Alexey N. Kudryavtsev, *Domain Decomposition Approach for Automatic Parallel Generation of Tetrahedral Grids*, Fraunhofer ITWM Report Nr. 87 (2006), pp. 3-21.

9. E.G. Ivanov, *Automatic Parallel Generation of Unstructured Three-Dimensional Grids*, Conference on Stability and Turbulence of Homogeneous and Heterogeneous Fluids Flows, Novosibirsk, 2005, pp. 79-82.

10. Evgeny G. Ivanov, *Implementation of Preprocessor for CSCM Navier-Stokes code by using Graphic User Interface*, Thesis for M.Sc. in Mechanical Engineering, Yeungnam University, Mechanical Engineering Department, South Korea, 2003, 56 p.

11. Makhsuda Juraeva, Evgeny G. Ivanov, Dong Joo Song, *Teaching Computational Fluid Dynamics via Internet*, Proceedings of the 4th IASTED International Conference on Web-Based Education (Ed. by V. Uskov). Grindelwald, Switzerland, 2005, Paper No. 461-016.

12. Evgeny G. Ivanov, Makhsuda Juraeva, Dong Joo Song, *Implementation of Full Web-based Graphic User Interface Processor for CFD software*, Proceedings of the Conference of Korean Society of Computational Fluids Engineering, South Korea, 2004, pp. 121-125.

13. Evgeny G. Ivanov, Seol Lim, Dong Joo Song, *Implementation of Preprocessor for CSCM Navier - Stokes code by using Graphic User Interface*, Joint Symposium between Sister Universities in Mechanical Engineering (JSSME), Yokohama, Japan, 2004.

14. Evgeny G. Ivanov, Dong Joo Song, *Implementation of Preprocessor for CSCM code by using Graphic User Interface*, Proceedings of the Conference of Korean Society of Computational Fluids Engineering, Taegu, South Korea, 2003, pp. 69-75.

15. Evgeny G. Ivanov, *Designing and Implementation of Visual Components for Nongraphical Part of Amthour Test*, Thesis for B.Sc. in Mathematics, Novosibirsk State Technical University, Faculty of Applied Mathematics and Computer Science, Russia, 2001, 64 p.

Conferences:

- 2007 World Congress in Computer Science, Computer Engineering and Applied Computing, PDPTA'07 - International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, USA, June 25-28, 2007;
- European Conference on Computational Fluid Dynamics (ECCOMAS CFD 06), Egmond aan Zee, The Netherlands, September 5-8, 2006;
- Conference on Numerical Geometry, Grid Generation and High Performance Computing, Moscow, Russia, July 4-7, 2006;
- 4th The International Association of Science and Technology for Development (IASTED) International Conference on Web-Based Education, Grindelwald, Switzerland, February 2005.
- Conference on Stability and Turbulence of Homogeneous and Heterogeneous Fluids Flows, Siberian Branch of Russian Academy of Sciences, Institute of Theoretical and Applied Mechanics, Novosibirsk, Russia, April 2005;
- Conference of Korean Society of Computational Fluids Engineering, Taegu, South Korea, August 2003;
- Joint Symposium between Sister Universities in Mechanical Engineering, Yokohama, Japan, August 2004.

Scientific seminars:

- Seminar «Formulation of Problems, which Allow Parallelization on Multiprocessor Computational Systems», Sobolev Institute of Mathematics SB RAS, Novosibirsk, Russia. (Seminar chairman: Academician S.K. Godunov.)

- Seminar at Erlangen-Nuremberg University, Department of Computer Science (Seminar chairman: Prof. U. Rüde.)

- Seminar of meshing software company DISTENE (INRIA), France.

- Seminar of Institute of Mathematical Modeling RAS, Moscow, Russia. (Seminar chairman: B.N. Chetverushkin , Director of IMM RAS.)

- CFD software company NUMECA (Numerical Mechanics Applications), Belgium. (Seminar chairman: Prof. Charles Hirsch, President of NUMECA Int.)

- Seminar «Mathematical Modeling in Mechanics», Institute of Theoretical and Applied Mechanics SB RAS. (Seminar chairmen: Academician V.M. Fomin , Prof. A.V. Fedorov.)

- Seminar of Techno- Mathematics working group, Technical University of Kaiserslautern, Department of Mathematics, Germany. (Seminar chairman: Prof. A. Klar.)

- Seminar «Aerogasdynamics», Siberian Branch of Russian Academy of Sciences, Institute of Theoretical and Applied Mechanics, Novosibirsk, Russia. (Seminar chairman: Prof. A.A Maslov, Deputy Director of ITAM.)

- Seminar of Competence Center for High Performance Computing and Visualization, Institute for Industrial Mathematics (ITWM), Kaiserslautern, Germany. (Seminar chairman: Dr. Franz-Josef Pfreundt, Head of CC HPC.)

- Seminar «Computational Technologies», Institute of Computational Technologies SB RAS. (Seminar chairman: Academician Yuri Shokin, Director of ICT, Prof. V.M. Kovenya.)
- Seminar «Supercomputer's Software and Architecture», Supercomputer Software Department, Institute of Computational Mathematics and Mathematical Geophysics SB RAS. (Seminar chairman: Dr. V.E. Malyshkin.)
- Seminar «Mechanics of Viscous Fluid and Turbulence», Siberian Branch of Russian Academy of Sciences, Institute of Theoretical and Applied Mechanics, Novosibirsk, Russia. (Seminar chairman: Prof. V.V. Kozlov, Head of Aerophysical Studies of Subsonic Flows Department.)

Acknowledgements:

This work is supported by the Fraunhofer Institute for Industrial Mathematics (Competence Center for High Performance Computing and Visualization), DAAD, TU Kaiserslautern and Institute of Theoretical and Applied Mechanics SD RAS. Author is grateful to A.N. Kudryavtsev, H. Andrä for careful supervision and help, to A. Klar and U. Rüde for discussion of results of the work, to S.K. Godunov for his interest to this work and many comments on subject, to B.N. Chetverushkin for helpful remarks on improving of the algorithm. Special thanks to Franz-Josef Pfreundt for his constant support of this work. Author wants to express his gratitude to all people who helped to accomplish this work in any way: D. Stoyanov, R. Zillich, O. Iliev, E. Teichmann and others.

List of figures

List of tables

Cirriculum Vitae

Personal data

Name:	Evgeny G. Ivanov
Address:	Fraunhofer Institute for Industrial Mathematics (Fraunhofer ITWM), Competence Center for High Performance Computing and Visualization, Fraunhofer Platz 1, D-67663 Kaiserslautern, Germany
E-mail:	ivanov@itwm.fhg.de
Date of Birth:	20 May, 1980

Academic preparation

2003–2008	**Ph.D. in Mathematics** Technical University of Kaiserslautern / Fraunhofer Institute for Industrial Mathematics (Fraunhofer -Institut für Techno- und Wirtschaftsmathematik), Kaiserslautern, Germany
2001–2003	**M.Sc. in Mechanical Engineering** Graduate School of Mechanical Engineering, Yeungnam University, South Korea
1997–2001	**B.Sc. in Mathematics** Novosibirsk State Technical University, Novosibirsk, Russia
1996–1997	Technical Lyceum of Novosibirsk State Technical University, Novosibirsk, Russia